Projektgruppe Apache Hadoop

BAND 1

Projektdokumentation

Projektgruppe Apache Hadoop

Projektdokumentation

Themenersteller:	Prof. Dr.-Ing. habil. Jorge Marx Gómez
Betreuer:	M. Eng. & Tech. Viktor Dmitriyev Dr.-Ing. Andreas Solsbach
Abgabetermin:	30. September 2016

Danksagung

Die Mitglieder der Projektgruppe bedanken sich bei der Abteilung VLBA der Carl-von-Ossietzky-Universität Oldenburg und insbesondere bei ihren Betreuern Prof. Dr.-Ing. habil. Jorge Marx Gómez, Dr.-Ing. Andreas Solsbach und M. Eng. & Tech. Viktor Dmitriyev für die Ermöglichung dieses Projektes und die umfangreiche Unterstützung in allen Bereichen während der gesamten Projektlaufzeit.

Zudem gilt unser Dank der Firma CEWE Stiftung & Co. KGaA. Insbesondere danken wir Dipl.-Ing. Eugen Neigel, M.Sc. Fatih-Mehmet Inel, M.Sc. Wiebke Osmers, Sancho Dauskardt, Dipl.-Ök. Oliver Behrend, BBA Matthias Kersken und Dipl.-Inf. Peter Schütz für die enge Zusammenarbeit und die zur Verfügung gestellten Ressourcen. Das hohe Interesse an den Vorgehensweisen und Ergebnissen unserer Projektgruppe und das uns geschenkte Vertrauen schätzen wir sehr.

Projektgruppe DASH, 30. September 2016

Inhaltsverzeichnis

Abkürzungen

API	Application Programming Interface
ASL2	Apache Software License, Version 2.0
AXT	Agent for XML Transportation (Eigenentwicklung der Projektgruppe)
Bash	Bourne-again shell
BI	Business Intelligence
CEWE	CEWE Stiftung & Co. KGaA
CRISP-DM	Cross Industry Standard Process for Data Mining
CSV	Comma-separated values
DAG	Directed Acyclic Graph
DFM	Digi Fotomaker
DWH	Data Warehouse
ELT	Extract-Load-Transform
ERP	Enterprise Ressource Planning
ETL	Extract-Transform-Load
FP	Frequent Pattern
FTP	File Transfer Protocol
FQDN	Fully Qualified Domain Names
GAG	gerichteter azyklischer Graph
HDFS	Hadoop Distributed Filesystem
HDP	Hadoop Data Platform
HIMP	Hive Importer (Eigenentwicklung der Projektgruppe)

HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
JAR	Java Archive
JDBC	Java Database Connectivity
JDK	Java Development Kit
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
KPI	Key Process Indicator (Kennzahl)
MB	Megabyte
MLlib	Machine Learning Library
NTP	Network Time Protocol
OLAP	Online Analytical Processing
OSF	Onsite-Finishing
RAM	Random Access Memory
RDBMS	Relationales Datenbank Management System
RDD	Resilient Distributed Dataset
REST	Representational state transfer
SaPPy	Sax Parser on Python (Eigenentwicklung der Projektgruppe)
SAX	Simple API for XML
SCRUM	Software Capability Rational Unified Model
SQL	Structured Query Language
SSH	Secure Shell
TC	Technisches Cockpit
UI	User Interface
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
UUID	Universally Unique Identifier

VIS	Vertriebsinformationssystem
VLBA	Very Large Business Applications (Abteilung der Carl von Ossietzky Universität Oldenburg)
VM	Virtuelle Maschine
XML	Extensible Markup Language
Yarn	Yet Another Resource Manager
ZML	Eigens entwickeltes Dateiformat für komprimierte Daten

Abbildungen

Tabellen

1. Einleitung

Dieses Projekt wurde im Rahmen der Masterstudiengänge Informatik und Wirtschaftsinformatik der *Carl von Ossietzky Universität* in Oldenburg durchgeführt. Hierfür bildeten zwölf Studenten für ein Jahr eine Arbeitsgruppe, welche unter Betreuung von Mitarbeitern der Universität eine vorgegebene Aufgabe untersucht und löst. Die Abteilung Wirtschaftsinformatik / Very Large Business Applications (VLBA) bot hierfür in Zusammenarbeit mit der Firma *CEWE* ein Projekt aus dem Themengebiet *Business Intelligence (BI)* an. In dieser Projektdokumentation werden die Planung, das Vorgehen und die Ergebnisse der Gruppe vorgestellt.

1.1. Motivation und Zielsetzung

Daten gewinnen für Unternehmen immer mehr an Bedeutung. Sie werden als vierter Produktionsfaktor neben Arbeitskraft, Rohstoffen und Kapital gesehen [Bit12, S. 7] oder auch als Rohstoff des 21.Jahrhunderts bezeichnet [BDS15, S. 12].

Die großen Datenmengen in Unternehmen bergen enorme Potenziale hinsichtlich der Analyse zur Generierung von Informationen und Wissen. Das Themenfeld Big Data umfasst strategische Ansätze, IT-Architekturen, Methoden und Verfahren, um Informationen und Wissen aus den exponentiell wachsenden und vielfältigen Daten zu generieren. Der Begriff Big Data ist grundlegend durch Datenvolumen (Volume), Datenvielfalt (Variety) und Geschwindigkeit (Velocity) charakterisiert - die sogenannten drei Vs. Mit Datenvolumen sind Volumen in Bereichen von Terabyte bis hin zu Yottabyte gemeint. Die Datenvielfalt beinhaltet strukturierte Daten (bspw. Daten aus relationalen Datenbanken), semistrukturierte Daten (bspw. XML oder JSON) und unstrukturierte Daten (bspw. Videodateien). Mit der Geschwindigkeit ist sowohl die Geschwindigkeit der Datenerzeugung als auch die Geschwindigkeit der Datenanalyse gemeint. [Bit12, S. 19, 21][BDS15, S. 12]

Auch der Unternehmenspartner der Projektgruppe, die CEWE, sucht immer nach neuen Möglichkeiten, um durch den Gewinn von Erkenntnissen und Wissen aus ihren Daten Wettbewerbsvorteile zu generieren. Um das Potenzial der großen und vielfältigen Datenmengen in Unternehmen wie CEWE zu nutzen, sind neue Technologien wie das Hadoop Ecosystem und NoSQL-Datenbanken entstanden [MSH14, S. 2][Bit14, S. 14].

Vor diesem Hintergrund hat CEWE der Projektgruppe ein Anwendungsszenario aus dem Unternehmen gestellt, an dem die Projektgruppe die Einsatzmöglichkeit des Hadoop Ecosystem untersuchen sollte. CEWE besitzt ca. 11.000 Kiosksysteme (DigiFotoMaker kurz: DFM genannt) bei ihren Handelspartnern in ganz Europa, an denen die Kunden Fotoprodukte direkt drucken oder im Labor bestellen können. Bei den Interaktionen zwischen den Kunden und den Digi Fotomakers (DFMs) entstehen täglich ca. 10 GB XML Dateien. Das Ziel der Projektgruppe ist es, mit Hilfe des Hadoop Ecosystems diese XML-Dateien täglich zu verarbeiten, um die darin enthaltenen Informationen für Analysen nutzbar zu machen. Zusätzlich soll die Projektgruppe erste Analysen durchführen, um einen ersten Eindruck über das Potenzial der Daten zu gewinnen.

1.2. Unternehmensportrait: CEWE Stiftung & Co. KGaA

CEWE wurde im Jahre 1961 von Heinz Neumüller gegründet. Um seinen Schwiegervater, Carl Wöltje, zu ehren, übernahm er dessen Initialen mit in die Firmenbezeichnung. Das Hauptgeschäftsfeld dieser Firma ist seither die Fotoentwicklung. Um dem technologischen Wandel und der damit von analogen zu digitalen Fotoprodukten schwankenden Nachfrage Stand zu halten, wurde das CEWE Fotobuch entworfen. Dieses Produkt erhält von allen Artikeln der CEWE Stiftung einen ständig steigenden Absatz. So wurden im Jahr 2015 beispielsweise 6,0 Mio. Exemplare verkauft [CEW16].

Mit ihren 12 Betriebsstätten, 3400 Mitarbeitern und einem Umsatz von 554,2 Mio. € im Jahr 2015 ist CEWE im europäischen Raum Marktführer im Bereich der Fotoentwicklung. Als Industrieunternehmen erhält CEWE ihre Aufträge durch ihre 25.000 belieferten Handelsgeschäfte. Die Endkunden bestellen ihre Produkte daher entweder in einem Geschäft des Handelspartners oder online auf einem für den jeweiligen Handelspartner angepassten Onlineshop.

In der folgenden Abbildung werden die Produktionsstandorte von CEWE auf einer Europakarte durch Punkte dargestellt.

Die rot gekennzeichneten Punkte zeigen zusätzlich Vertriebsstandorte auf. Die Produktionsstätte in Oldenburg ist zugleich der Hauptsitz der Firma. Aus diesem Grund wird diese auf der Abbildung namentlich erwähnt.

Bezeichnend ist für dieses Unternehmen, dass es den drastischen Wandel der Nachfrage im Bereich der Fotoprodukte standhielt. Die analoge Fotografie wurde größtenteils durch digitale Fotoprodukte abgelöst.

Ein weiterer Meilenstein in der Geschichte von CEWE war die Abwehr der Forderungen amerikanischer Hedgefonds-Firmen. Im Jahre 2005 wurden der Vorstand und der Aufsichtsrat unter anderem zu einer Dividende gedrängt, welche

Abbildung 1.1.: Produktionsstandorte von CEWE (Stand: März 2016)

teilweise durch Kredite finanziert werden müsste und der Firma langfristig geschadet hätte. Durch einen Mehrheitsbeschluss der Hauptversammlung wurde dies jedoch abgelehnt.

1.3. Aufbau der Projektdokumentation

Folgend werden die Inhalte der verschiedenen Kapitel in der Reihenfolge ihres Auftretens zusammenfassend vorgestellt.

1. **Einleitung:** Einleitend wird die Motivation und die Zielsetzung der Projektgruppe, sowie der Projektpartner vorgestellt.
2. **IST-Situation des Unternehmenspartners:** Die Ausgangslage vor dem Start des Projektes wird in diesem Kapitel dargelegt. Hierzu wird vorgestellt, wie bisher die für dieses Projekt relevanten Daten entstehen, was diese beinhalten und wie sie bereits analysiert werden.
3. **Soll-Zustand:** Ziele des Projektes werden in diesem Abschnitt zusammengefasst. Hierbei werden technologische Vorgaben erläutert. Aus der Differenz zwischen IST- und Sollzustand ergibt sich hier bereits eine Vorstellung für die Architektur und das Vorgehen der Projektgruppe.

4. Projektmanagement: In diesem Kapitel werden die Projektmitglieder und die Vorgehensweise zur Koordination des Projektes vorgestellt. Zudem werden die jeweiligen Phasen und Meilensteine erläutert.

5. Evaluation verfügbarer Software: Zur erfolgreichen Durchführung des Projektes wurden für unterschiedliche Aufgabenbereiche entsprechende Softwarekomponenten evaluiert. In diesem Abschnitt werden die Teilbereiche und die hierfür erprobten Softwaretools vorgestellt.

6. Praktische Durchführung: Nachdem verfügbare Software evaluiert wurde, kann die konkrete Umsetzung des Projektes durchgeführt werden. Komponenten werden installiert und konfiguriert und Eigenentwicklungen implementiert.

7. Hadoop Ecosystem der Projektgruppe: Die gesamte Architektur der Projektgruppe wird in diesem Abschnitt vorgestellt. Hierbei liegt der Fokus vor allem auf dem ELT-Prozess und allen dazugehörigen Komponenten.

8. Test & Evaluation: Um eine möglichst lauffähige und stabile Software zu entwickeln, wurde ein Testmanagement eingesetzt. In diesem Kapitel wird das Vorgehen hierbei erläutert.

9. Fazit: Die Ergebnisse der Gruppe werden zusammenfassend dargestellt.

2. IST-Situation des Unternehmenspartners

In diesem Kapitel wird das gestellte Anwendungsszenario des Unternehmenspartners CEWE an die Projektgruppe detailliert erläutert. Die Kunden von CEWE können ihre Fotoprodukte über verschiedene Wege bei CEWE bzw. einem der Handelspartner in Auftrag geben. Eine Möglichkeit ist, die Produkte Zuhause zu gestalten und im Labor entwickeln zu lassen. Dies geht sowohl über eine Software, die auf dem Computer installiert wird, mittels einer App auf dem Smartphone oder Tablet sowie online auf der Internetseite des Handelspartners. Nachdem die Produkte im Labor gefertigt worden sind, werden sie dem Kunden dann entweder zugeschickt oder er kann sie in einer Filiale des Handelspartners abholen.

Falls der Kunde nicht warten möchte, bis die Produkte im Labor gefertigt worden sind, kann er auch einige Produkte direkt an einem Kiosk-Gerät, DFM genannt, von CEWE drucken. Es befinden sich ca. 11.000 DFMs in etwa 8.000 Filialen der Handelspartner von CEWE ([OB15], [BNIK15]). Hier können die Produkte direkt am Gerät gestaltet und anschließend vor Ort gedruckt werden. Dieser Prozess nennt sich Onsite-Finishing (OSF). Es besteht weiterhin die Möglichkeit, das gewünschte Produkt über den DFM zu bestellen und im Labor fertigen zu lassen.

2.1. Datenentstehung

Bei der Bedienung eines DFMs werden neben den Auftragsdaten auch das Bedienverhalten und technische Daten des DFM wie beispielsweise Informationen zum Drucker anonymisiert in einer Log-Datei festgehalten. Dies geschieht sowohl beim OSF als auch bei der Bestellung von Labor-Produkten. Eine Log-Datei stellt dabei eine Session dar. Die Session beginnt mit der ersten Interaktion des Kunden mit dem DFM, das heißt der erste Klick um den Bildschirmschoner zu verlassen. Sie endet, wenn der Kunde die Bestellung aufgibt, die Session manuell über das Verlassen-Menü beendet oder das Gerät so verlässt und es wieder in den Ruhe-Modus geht. Anhand der Informationen, die in den Log-Dateien enthalten sind, kann die Bestellsoftware optimiert werden.

Die Log-Dateien sind in einem semi-strukturierten Format als Extensible Mark-

up Language (XML)-Datei gesichert. Die XML-Dateien werden von dem DFM an einen Unternehmenspartner von CEWE übermittelt. Dieser Unternehmenspartner bereitet die Dateien für CEWE auf, sodass die Informationen anschließend in einer strukturierten Form vorliegen. Anhand der aufbereiteten Daten kann CEWE anschließend ihre Analysen durchführen. Hierbei wird derzeit noch nicht das volle Potential der Daten genutzt, da nicht alle Informationen aus der XML-Datei aufbereitet werden.

2.2. Datenanalyse

Um das gesamte Analysepotenzial aus den protokollierten Daten der DFM-Geräte auszuschöpfen, ist es notwendig, diese vollständig zu erfassen und mit weiteren unternehmensinternen und -externen Datensätzen anzureichern. Hierzu müssen die Daten in aufbereiteter Form in Datenbanken vorliegen. Zu Beginn der Projektgruppe im dritten Quartal 2015 hat CEWE bereits Zugriff auf den Großteil dieser Daten. So werden über das CEWE-interne Data Warehouse (DWH), das Vertriebsinformationssystem (VIS) und das Enterprise Ressource Planning (ERP)-System sämtliche Geschäfts-, Auftrags- und DFM-Bestandsdaten sowie DFM-ServiceMeldungen bereitgestellt. Diese Daten lassen sich mit einer Analysesoftware abrufen. In Zusammenarbeit mit einem IT-Dienstleister stehen zudem die Statusinformationen der Online-Drucker in Form von Live-Daten zur Verfügung. Externe Informationen wie beispielsweise Wetterdaten werden zugekauft. Letztlich sind auch die Inhalte der von den DFM erzeugten XML-Daten über einen externen Server des IT-Dienstleisters in aufbereiteter Form für das DWH und mit einem webbasierten Analysetool abrufbar. (siehe Abbildung 2.1) Einige dieser Inhalte werden jedoch nicht in der Datenbank abgelegt. Hierzu zählen unter anderem die Clickstreamdaten. Diese sind relevant für die Optimierung der Bedienoberfläche des DFM.

Genutzt werden die vorhandenen Daten vorrangig für das deskriptive Reporting. Die meisten Analysen und Datenexplorationen werden von Hand mit den zuvor beschriebenen Lösungen sowie Tabellenverarbeitungsprogrammen durchgeführt. Umfangreiche Visualisierungsfunktionen und die Möglichkeit, Analysen mit Daten aus unterschiedlichen Systemen zu kombinieren, fehlen. [OB15] [Beh14]

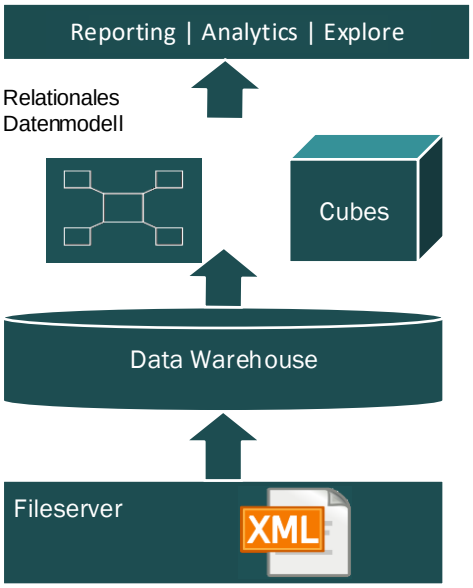


Abbildung 2.1.: XML-Datenweg, eigene Darstellung in Anlehnung an [BNIK15]

2.3. Dateninhalt

In dem Anwendungsszenario von CEWE bilden XML-Dateien die Ausgangsbasis. Die semistrukturierten Daten sollten strukturiert aufbereitet werden, um aus den Daten Informationen und aus den Informationen Wissen zu generieren. Ein wichtiger Schritt war ein Workshop mit CEWE und einem externen Dienstleister von CEWE. Dieser Workshop diente der Projektgruppe dazu, mögliche Inhalte einer XML-Datei zu analysieren, um die relevanten Informationen zu identifizieren. Das Ergebnis des Workshops ist eine Dokumentation der XML-Dateien, welche sich im Anhang A.1 befindet. Mit dem Wissen aus dem Workshop und der XML-Dokumentation wurde abgesteckt, welche Informationen aus den XML-Dateien geparst und strukturiert werden, um sie für Analysen zugänglich zu machen.

Die XML-Dateien sind in die Bereiche Terminal, Ticket, Path und Summary unterteilt. Diese Bereiche beinhalten verschiedene Informationen und bilden auch die allgemeine Aufteilung der XML-Dokumentation.

Zudem wurden im Verlauf des XML-Workshops potenziell nützliche Informationen für die Analysen definiert. Die beinhalteten Informationen der Hauptbereiche einer XML-Datei werden im Folgenden grob erläutert:

Terminal: Das Terminal Tag enthält Informationen über Stammdaten des DFM wie Standort, Händler und Gerätenummer.

Ticket: Im Ticket Bereich stehen relevanten Informationen wie die aktuelle Firmware des DFM.

Path: Der Path Bereich beinhaltet die Informationen über die Interaktion des Nutzers mit dem DFM. Es wird festgehalten, welche Prozessschritte der Software durchgeführt worden sind. Zudem werden relevante Informationen über ausgewählte Produkte, den Drucker und den Druckvorgang festgehalten.

Summary: Der Summary Bereich beinhaltet zusammenfassende Informationen der Session des Nutzers. Wesentliche Informationen sind bspw. der Status der Session (erfolgreich und nicht erfolgreich) sowie die gekauften Produkte und der Gesamtpreis.

3. Soll-Zustand

Der Soll-Zustand des Projektes wird in diesem Abschnitt beschrieben. In Tabelle 3.1 sind die Anforderungen des Projektpartners CEWE und der Projektbetreuer an die Gesamtarchitektur und den Workflow aufgeführt. Ausgehend von den Anforderungen wird in diesem Kapitel der Soll-Zustand der Gesamtarchitektur und des Extract-Load-Transform (ELT)-Prozesses dargestellt.

Kürzel	Beschreibung
GAW-01	Zur Lösung des Anwendungsszenarios soll das Hadoop Ecosystem genutzt werden.
GAW-02	Die Daten sollen täglich verarbeitet werden können.
GAW-03	Es sollen für beliebige Tage in der Historie Wiederholungsläufe möglich sein.
GAW-04	Die Informationen aus den XML-Dateien sollen täglich zur Analyse zur Verfügung stehen.
GAW-05	Steuerung des Workflows durch ein Workflowmanagement Tool.
GAW-06	Für den ELT-Prozess werden Tools benötigt, die den Datenimport, Datentransformation und das Laden der Daten in Hive unterstützt.
GAW-07	Automatisierung des Workflows und Logging-Möglichkeit.
GAW-08	Daten zu Analysezwecken sollen strukturiert und in einem Hive-Schema persistiert werden.
GAW-09	Analyse der XML-Dateien (Deskriptive- und Predictive-Analytics).
GAW-10	Erstellung eines Management Dashboards und eines Technisches Cockpit (TC).
GAW-11	Nutzung verschiedener Visualisierungsformen.

Tabelle 3.1.: Anforderungen an die Gesamtarchitektur und den Workflow

3.1. Gesamtarchitektur

In Abbildung 3.1 ist die grobe Zielarchitektur zu sehen. Die beschriebenen XML-Dateien vom File Transfer Protocol (FTP) Server (Nummer 1) sollen mit Hilfe des Hadoop Ecosystem (Nummer 2) für die Extraktion von Informationen, Wissen und Erkenntnissen (Nummer 3) nutzbar gemacht werden. Dabei ist der entscheidende Baustein der Architektur das Hadoop Ecosystem der Projektgruppe (Nummer 2), das im Laufe des Projektes entwickelt werden soll. Das Hadoop Ecosystem der Projektgruppe soll dazu dienen, die Informationen in den XML-Dateien täglich nutzbar zu machen, um Wissen und Erkenntnisse aus diesen zu generieren. (siehe Anforderungen in Tabelle 3.1)

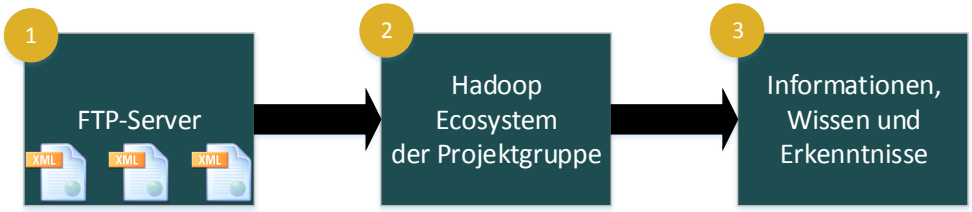


Abbildung 3.1.: Soll Zustand

3.2. Workflow

In diesem Abschnitt wird der ELT-Workflow der Projektgruppe beschrieben. Es wird aufgezeigt, wie die am DFM generierten XML-Daten von einem FTP-Server in das Dateisystem Hadoop Distributed Filesystem (HDFS) geladen werden, anschließend in ein relationales Format transformiert und in der Hive-Datenbank gespeichert werden sollen und zuletzt mit Hilfe eines Visualisierungstools dargestellt werden sollen. Die Abbildung 3.2 zeigt den Workflow.

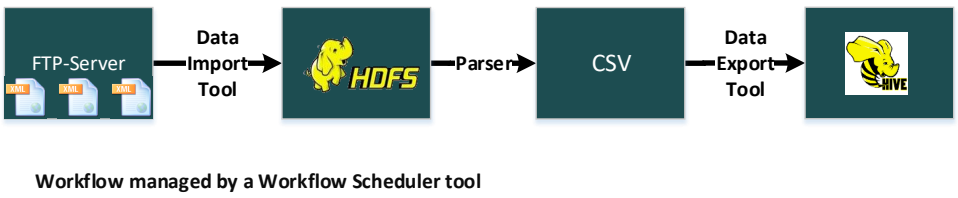


Abbildung 3.2.: Soll-Workflow

Die Abbildung 3.2 zeigt den Workflow im Zielzustand. Der Datenfluss ist von links nach rechts zu betrachten. Für die Steuerung des Prozesses soll ein Workflowmanagement-Tool genutzt werden.

Der ELT-Prozess beginnt beim FTP-Server, auf dem die XML-Dateien hinterlegt sind und soll mit der Übertragung der Daten in Hive-Tabellen enden. Die Dateien sollen zuerst vom Server extrahiert werden (**E**xtract), anschließend in das HDFS geladen (**L**oad) und zuletzt transformiert und in Hive gespeichert werden(**T**ransform).

Für die Arbeitsschritte in diesem Prozess sollen vornehmlich Tools des Hadoop Ecosystem zum Einsatz kommen. Um eine Auswahl der Tools zu treffen werden verschiedene Softwarelösungen aus dem Hadoop-Ecosystem evaluiert. Insbesondere für den Import der Daten vom FTP-Server in das HDFS, sowie für die Transformation der semi-strukturierten XML-Daten in eine strukturierte Zielstruktur sollen verschiedene Tools untersucht werden. Erfüllt kein Tool des Hadoop Ecosystem den gestellten Anforderungen für den jeweiligen Prozessschritt, muss ein alternativer Lösungsweg (ggf. eine Eigenentwicklung) gefunden werden.

Die Zielstruktur soll eine Hive-Datenbank bilden. Diese soll aus verschiedenen Tabellen mit Informationen aus den unterschiedlichen Bereichen der XML-Strukturen bestehen. Wie in Abbildung 3.2 zu erkennen, soll der sogenannte Parser, der die Datentransformation vornehmen soll, Comma-separated values (CSV)-Dateien erzeugen, die über ein *Data Export Tool* in das Hive-Schema überführt werden sollen.

Der Workflow sollte täglich und automatisch zu einer bestimmten Uhrzeit von einem *Workflowmanagement-Tool* ausgeführt und geloggt werden. Sollte ein Fehler auftreten, so muss dieses für den verantwortlichen Anwender ersichtlich sein.

Das Hive-Schema soll die strukturierte Datengrundlage für weitergehende Analysen der DFM-Daten bilden. Um die Analysen durchführen zu können, sollen die relevante Informationen in den Daten identifiziert und ein sinnvolles strukturiertes Datenbankschema entwickelt werden.

Die Ergebnisse der Datenanalyse sollen übersichtlich und leicht verständlich präsentiert werden. Um dieses Ziel zu erreichen, wird ein Visualisierungstool gesucht, mit dem ein Dashboard entworfen werden kann. In diesem Dashboard sollen Analyseergebnisse in Form von Grafiken und Tabellen präsentiert werden.

4. Projektmanagement

4.1. Mitglieder der Projektgruppe

Die Projektgruppe besteht aus zwölf Mitgliedern. Abbildung 4.1 zeigt die Mitglieder und die groben Aufgabenbereiche, in denen die einzelnen Projektgruppenmitglieder mitgewirkt haben.

Abbildung 4.1.: Mitglieder der Projektgruppe mit groben Aufgabenbereichen

Die Rollenvergabe vom Projektmanager, Dokumentationsbeauftragten, Webseitenverantwortlichen und Systemadministratoren waren vorgegebene Rahmenbedingung der Projektbetreuer und wurden zu Beginn des Projektes festgelegt. Die

personelle Besetzung der Rollen ist ebenfalls der Abbildung 4.1 zu entnehmen. Im folgenden werden die Aufgabenbereiche beschrieben.

Projektmanagement: Der Aufwand für das Projektmanagement sollte eine Schwelle von 20% der wöchentlichen Arbeitszeit für die Projektgruppe nicht überschreiten. Zum Projektmanagement zählen Aufgaben wie die Koordination von Terminen mit dem Unternehmenspartner, das Vorbereiten der Termine und das Koordinieren von Aufgaben innerhalb der Projektgruppe.

Dokumentationsbeauftragter: Der Dokumentationsbeauftragte ist für das einheitliche Layout aller Dokumente zuständig. Die wichtigste Aufgabe ist die Koordination und das Betreuen der Abschlussdokumentation.

Systemadministratoren: Die Systemadministratoren sind für die Installation und die Betreuung des Hadoop Clusters in der Universität verantwortlich. Weitere Details sind in Kapitel 6.1 dargestellt.

Scrum Master: Der Scrum Master hat die Aufgabe, den Scrum Prozess der Projektgruppe zu steuern und zu überwachen. Die Kontrolle der Jira-Tickets und das Durchführen der Sprint-Abschlüsse, sowie das Durchführen von Retrospektiven sind wichtige Aufgaben.

Webseitenverantwortlicher: Der Webseitenverantwortliche ist für die Erstellung von Blog-Beiträgen zuständig, um der Außenwelt den Fortschritt der Projektgruppe transparent zu machen.

Bereich Extract & Load im ELT-Prozess: Die Aufgaben in diesem Bereich befassen sich mit dem Import der XML-Dateien in das Hadoop System. Weitere Details sind in Kapitel 5.1 und 6.4 zu finden.

Bereich Transformation im ELT-Prozess: In diesen Bereich fällt das Parsen der XML-Dateien in eine strukturierte Datenhaltung. Der Prozess wird in den Kapiteln 5.2 und 6.5 detailliert beschrieben.

Analytics: Der Bereich Analytics umfasst Aufgaben der deskriptiven Analyse und des Data Minings, um Wissen und Erkenntnisse aus den XML-Dateien zu extrahieren. Die Kapitel 5.3 und 6.6 beschreiben den Bereich Analytics im Detail.

Automatisierung des ELT-Prozesses: Die Automatisierung des ELT-Prozesses ist eine zentrale Aufgabe. Der Prozess soll automatisiert gestartet und ggf. wiederholt werden können. Weitere Details sind in den Kapiteln 5.5 und 6.8 zu finden.

Testmanagement: Der Bereich Testmanagement umfasst die vollständige Aufgabe, alle Bereiche der Projektgruppe zu testen und einer Qualitätssicherung zu unterziehen. Das Vorgehen in diesem Bereich wird näher in Kapitel 8 erläutert.

4.2. Projekt-Stakeholder

CEWE: Als Partnerunternehmen für die Projektgruppe hat CEWE der Projektgruppe einige Mitarbeiter als Unterstützung zur Verfügung gestellt. Hauptansprechpartner und Auftraggeber der Projektgruppe sind Eugen Neigel und Fatih-Mehmet Inel. Als Ansprechpartner für Themen zum DFM und zu den XML-Dateien stehen Oliver Behrend und Matthias Kersken (beide CEWE), sowie Sancho Dauskardt (Hertz) zur Verfügung. Wiebke Osmers und Malte Prang sind Ansprechpartner für das Beta-System bei CEWE.

Projektbetreuer der Universität: Die Projektbetreuer sind Viktor Dmitriyev und Andreas Solsbach. Beide sind während der gesamten Projektlaufzeit Ansprechpartner für alle relevanten Themen und standen der Projektgruppe zu jeder Zeit mit Rat und Tat zur Seite. Auch die inhaltliche und organisatorische Unterstützung erfolgte durch die Projektbetreuer.

4.3. Projektphasen und Meilensteine

In Abbildung 4.2 sind die Projektphasen und die wesentlichen Meilensteine der Projektgruppe zu sehen. Der Projektplan war das Rahmenwerk, um den Projektmitgliedern und den Stakeholdern zu jederzeit einen aktuellen Projektstand liefern zu können. Zudem diente der grobe Projektplan der Team- und Aufgabenpaketbildung innerhalb der Projektgruppe. Ein weiterer wichtiger Punkt waren die Meilensteine, um den Erfolg des Projektes zu gewährleisten und die Projektgruppe zusätzlich anzuspornen. Der Projektplan ist in zwei Teilen entstanden, der erste Teil von Vorgang eins bis acht im ersten Semester und der zweite Teil von Vorgang neun bis fünfzehn in Semester zwei der Projektgruppenlaufzeit. Die zweistufige Planung gab der Projektgruppe Zeit, sich mit der Thematik vertraut zu machen und so die zweite Phase zu einem späteren Zeitpunkt detaillierter planen zu können. Die Projektphasen und die Meilensteine werden im Folgenden erläutert:

1. Seminarphase und Einarbeitung Hadoop: Die erste Projektphase diente allen Projektgruppenmitgliedern zur Einarbeitung in die Thematik der Projektgruppe. Hierzu wurde an jedes Projektgruppenmitglied ein Seminararbeitsthema mit Bezug zur Projektgruppe vergeben. Jedes Projektgruppenmitglied hatte zwei Monate Zeit eine Ausarbeitung und eine Präsentation zum jeweiligen Thema anzufertigen.

2. Anforderungsanalyse und Grobkonzept: Bereits während der Seminarphase wurde damit begonnen die Anforderungen von CEWE aufzunehmen und ein Grobkonzept zu erstellen.

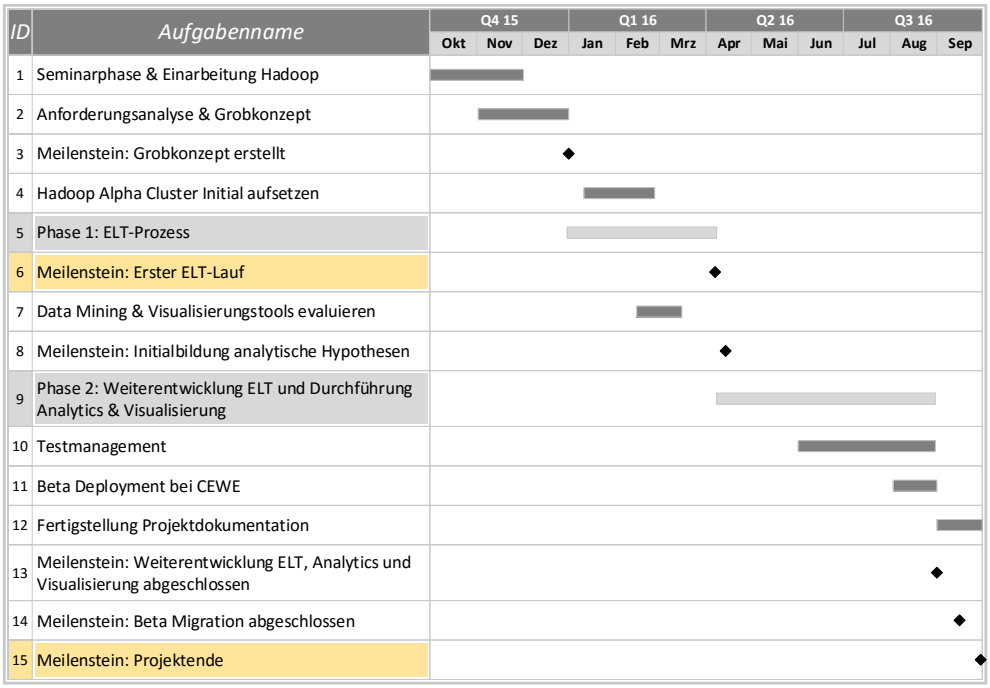


Abbildung 4.2.: Grobe Projektphasen und wesentliche Meilensteine

3. **Meilenstein Grobkonzept erstellt:** Ein erster wichtiger Meilenstein war die Abnahme des Grobkonzeptes seitens CEWE und der Projektbetreuer, um sicher zu gehen, dass das Projekt den richtigen Weg eingeschlagen hat.
4. **Hadoop Alpha Cluster initial aufsetzen:** Für die Umsetzung des Projektes ist ein Hadoop Cluster in der Universität vorgesehen. In dieser Projektphase wurde das sogenannte Alpha Cluster in der Universität initial installiert.
5. **Phase 1 - ELT-Prozess:** Diese Projektphase ist eine der längsten und wichtigsten im gesamten Projekt gewesen. In dieser Projektphase ist der ELT-Prozess initial konzeptioniert und entwickelt worden, für den sich bei der Grobkonzeption entschieden wurde.
6. **Meilenstein - Erster ELT-Lauf:** Zu diesem Meilenstein sollte der ELT-Prozess (manuell oder bereits teilautomatisiert) durchlaufen worden sein.
7. **Data Mining und Visualisierungstools evaluieren:** Nach der ersten Hälfte der Projektlaufzeit wurden Data Mining- und Visualisierungstools des Hadoop Ecossytem evaluiert, um die analytischen Anforderungen umzusetzen.
8. **Meilenstein: Initialbildung analytische Hypothesen:** Dieser Meilenstein sollte dazu dienen, initial die analytischen Fragestellungen aufzustellen und gemeinsam mit den Projektstakeholdern zu priorisieren.
9. **Phase 2 - Weiterentwicklung ELT und Analytics und Visualisierung:** Die

zweite wichtige und lange Projektphase war die Weiterentwicklung, Optimierung und Finalisierung des ELT-Prozesses, der Beginn mit den analytischen Aufgaben und die Visualisierung der gewonnenen Erkenntnisse.

10. **Testmanagement:** Im letzten Drittel der Projektlaufzeit ist mit dem Testmanagement begonnen worden, um die Ergebnisse des Projektes umfassend qualitativ abzusichern.
11. **Beta Deployment bei CEWE:** Zum Ende des Projektes wurde das entwickelte System auf das Hadoop Cluster des Unternehmenspartners CEWE migriert.
12. **Fertigstellung Projektdokumentation:** Der letzte Monat des Projektes dient der Fertigstellung der Projektdokumentation.
13. **Meilenstein: ELT, Analytics und Visualisierung abgeschlossen:** An diesem Meilenstein wurde gemessen, ob alle aufgestellten Anforderungen innerhalb des Projektes umgesetzt wurden.
14. **Meilenstein: Beta Migration abgeschlossen:** Dieser Meilenstein diente zur Messung der Migration des Systems zum Hadoop Cluster bei CEWE.
15. **Meilenstein: Projektende** Der letzte Meilenstein steht für das Projektende am 30.09.2016.

Im Gantt-Diagramm in Abbildung 4.2 sind die Vorgänge sechs und neun gelb und die Vorgänge fünf und fünfzehn grau hinterlegt. Die grau hinterlegten Vorgänge betonen die entscheidenden und arbeitsintensivsten Phasen des Projektes. An diesen Projektphasen wurde eine lange Zeit und mit viel Personalaufwand gearbeitet. Genauso entscheidend war auch der Meilenstein für den ersten ELT-Lauf in Vorgang sechs. Dieser Meilenstein konnte im Projekt nicht gehalten werden und wurde ca. sechs Wochen später erreicht. Dadurch konnte weniger Zeit als geplant in die Durchführung der Analysen, ein Teil von Vorgang dreizehn, investiert werden. Der Meilenstein des Projektendes und der Abgabe sämtlicher Unterlagen wurde erfolgreich eingehalten und das Projekt erfolgreich beendet.

4.4. Projektvorgehen

Ein Projekt mit zwölf Mitgliedern zu koordinieren und zu organisieren ist eine Herausforderung. Noch schwieriger wird es, wenn es ein Projekt mit Teilzeit-Präsenz ist. Die Projektgruppenmitglieder haben eine Wochenarbeitszeit von ca. 14 Stunden von denen ca. 7-8 Stunden zusammen in einem Projektgruppenraum gearbeitet wurden. Hier kann von einem Projekt in Teilzeit-Präsenz gesprochen werden. Um die Koordination möglichst optimal und somit das gesamte Projekt effizient und effektiv zu gestalten, wurden Aufgabenbereiche definiert, die von kleinen Teams mit ein bis fünf Personen bearbeitet wurden. Diese groben Aufga-

benbereiche sind in Abschnitt 4.1 beschrieben worden. Nähere Details zum Vorgehensmodell im Projekt, zu organisatorischen Rahmenbedingungen, zur Kommunikation und zur Unterstützung durch Software werden in den nächsten Abschnitten erläutert.

4.4.1. SCRUM

Die Projektgruppe hat sich zu Beginn des Projektes für das agile Projektvorgehen SCRUM entschieden. Alle Projektgruppenmitglieder hatten während des Bachelor Studiums bereits erste positive Erfahrungen mit SCRUM erlebt. Die theoretischen Grundlagen zu SCRUM können im Anhang unter nachgelesen werden.

Nachdem die Seminararbeitsphase abgeschlossen war, waren die notwendigen theoretischen Grundlagen für den Projektstart geschaffen. Die wesentlichen Anforderungen sind bereits in ersten Kickoff-Meetings mit CEWE und den Projektbetreuern festgelegt worden. Durch die geringe Erfahrung mit einem derartigen Projekt und der verwendeten Technologie Hadoop, ist der Backlog über die ganze Projektlaufzeit in Bewegung geblieben. Die Sprints sind in zwei wöchigen Zyklen durchlaufen worden. Sprintstart und Sprintende waren immer an einem Donnerstag, dem Tag an dem die Projektgruppe ihre Präsenzzeit im Projektgruppenraum hat. Zur Unterstützung des SCRUM Vorgehens wurde die Software JIRA benutzt (siehe 4.4.3).

4.4.2. Kommunikationswege und Rahmenbedingungen im Projekt

Für eine optimale Koordination und Kommunikation im Projekt wurden feste Kommunikationswege zum Anfang des Projektes festgelegt und einige organisatorische Spielregeln für die Zusammenarbeit bestimmt. Der Kommunikationsweg von der Projektgruppe zu den Projektbetreuern war jederzeit gegeben, dennoch gab es ein offizielles wöchentliches Treffen zwischen der Projektgruppe und den Projektbetreuern, um sich über den aktuellen Stand des Projektes auszutauschen. Mit dem Unternehmenspartner CEWE fanden im Rhythmus von 6-10 Wochen regelmäßige Statusmeetings statt, in denen der aktuelle Projektstatus vorgestellt und diskutiert worden ist. In diesen Statusmeetings konnte der Unternehmenspartner Einfluss auf die Entwicklung des Projektes nehmen. Bei Fragen oder Problemen konnten diese auch per Telefon oder E-Mail mit Eugen Neigel oder Fatih Inel beantwortet und gelöst werden.

Nachfolgend sind die Spielregeln der Projektgruppe aufgelistet:

- Jedes Projektgruppenmitglied hat 15 Tage Urlaub ggf. für Klausurenphasen und Sommerurlaub.

- Das offizielle Meeting mit den Projektgruppenbetreuern ist verpflichtend.
- Die wöchentliche Präsenzzeit der Projektgruppe ist ebenfalls verpflichtend.
- Urlaub und Krankheit sind so früh wie möglich anzukündigen und der Projektgruppe mitzuteilen.
- Im offiziellen Meeting und während der Präsenzzeit gibt es einen Protokollanten, der das Protokoll zeitnah (nach 2 Tagen) an alle Projektgruppenmitglieder verteilt.
- Bei Abwesenheit ist das Weekly-Scrum an den Protokollanten zu schicken.

4.4.3. Software-Werkzeuge

Damit das Projekt optimal beendet wird, wurde für die Unterstützung zur Kommunikation und zum Wissensaustausch auf Software von Atlassian zurückgegriffen. Hier benutzte die Projektgrupppe das Tool Jira, das die Implementierung des SCRUM-Vorgehens und die Aufgabenverwaltung unterstützt, sowie das Tool Confluence für die Wissensspeicherung und den Wissensaustausch. Durch die Software-Unterstützung konnte die Projektgruppe die Kommunikation und den Wissensaustausch trotz der wenigen Präsenzzeiten verbessern.

In Abbildung 4.3 ist ein Screenshot des Confluence der Projektgruppe zu sehen. Hier wurden beispielsweise vom Dokumentationsbeauftragten alle nützlichen Tipps und Tricks rund um Latex dokumentiert und alle Regeln beim Erstellen von Beiträgen in der Gesamtdokumentation. Jedes Projektgruppenmitglied konnte somit alle notwendigen Informationen zum Erstellen eines Beitrages für die Gesamtdokumentation im Confluence der Projektgruppe finden.

Abbildung 4.3.: Screenshot Atlassian Confluence

Für das Projektvorgehen mit SCRUM nutzte die Projektgruppe JIRA. (siehe Ab-

bildung 4.4) In der Abbildung sind die vier Phasen, die eine Aufgabe durchlaufen soll (Aufgabe, Wird Ausgeführt, Testen, Fertig (getestet)), abgebildet. In dieser Ansicht kann jedes Projektgruppenmitglieder über alle Aufgaben und über alle seine eigenen Aufgaben den aktuellen Zustand sehen. Zudem können alle Tickets kommentiert werden, sodass auch ein Informationsaustausch unter den Projektgruppenmitgliedern in den Tickets stattfinden kann.

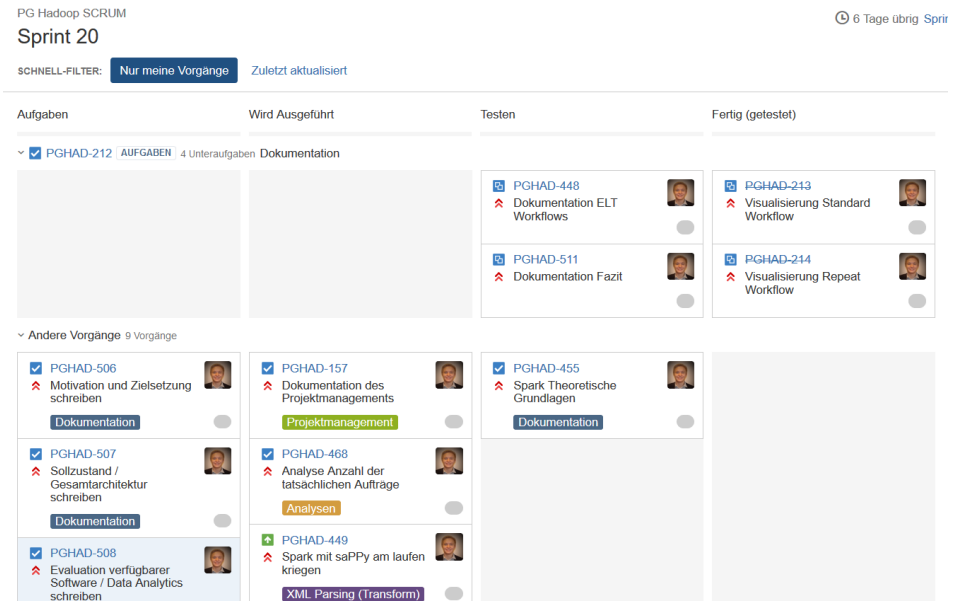


Abbildung 4.4.: Screenshot Atlassian Jira

4.4.4. Lessons learned Projektmanagement

Dieser Abschnitt soll dazu dienen ein kurzes Fazit für das Projektmanagement zu ziehen. Zwölf Projektgruppenmitglieder über ein Jahr mit einem wöchentlichen Workload (Arbeitszeit) von 14 Stunden effektiv und effizient zu koordinieren, ist eine Herausforderung. Durch klare Spielregeln in der Zusammenarbeit, durch das gewählte Projektvorgehen SCRUM und durch unterstützende Software wie Jira und Confluence konnte das Projektmanagement gut durchgeführt werden. Besser wäre es gewesen, wenn das Projektmanagement und die Rolle des Scrum-Masters von einer Person durchgeführt worden wäre. Zudem war der veranschlagte wöchentliche Aufwand von ca. drei Stunden für das Projektmanagement rückblickend zu wenig. Bei einer Projektgruppengröße von 12 Mitgliedern sollte ein Projektmanager 60-80% seiner Zeit mit dem Verteilen, Koordinieren, Kommunizieren und Abstimmen von Aufgaben, Zielen und Inhalten beauftragt werden.

5. Evaluation verfügbarer Software

Nach dem Entwurf des Workflows und der Zielarchitektur werden in diesem Kapitel fertige Softwarelösungen zur Implementierung des Zielbildes evaluiert. Dabei werden Tools in fünf verschiedenen Bereichen untersucht. In den Abschnitten 5.1 Data Import, 5.2 Data Transformation und 5.5 Workflowmanagement werden Tools zur Implementierung des ELT-Workflows untersucht und in den Abschnitten 5.4 Data Visualization und 5.4 Data Visualization Tools zur Unterstüzung von Analyseverfahren evaluiert.

Für jeden Bereich werden Anforderungen erhoben und tabellarisch zusammengefasst. Diese sollten durch die Softwarekomponenten abgedeckt werden. Dabei kann es vorkommen, dass für einen Aufgabenbereich mehrere Tools in Frage kommen und sogar zusammen verwendet werden. Ebenso ist es möglich, dass alle untersuchten Komponenten nicht die gewünschte Funktionalität erfüllen und eine eigene Entwicklung durchgeführt werden soll.

In einem abschließenden Fazit zu jedem Aufgabenbereich wird der gewählte Lösungsweg dargestellt.

5.1. Data Import

Die bereits in Kapitel 2 beschriebenen XML-Dateien bilden die Grundlage zukünftiger Analysen. Neben diesen semistrukturierten Dateien können auch Daten aus bestehenden relationalen Datenbank für Analysezwecke herangezogen werden. Da die Analyse mittels Hadoop durchgeführt werden soll, müssen die XML-Dateien, die zur Zeit auf Servern von CEWE liegen, in das HDFS importiert werden. Gleiches gilt für die relationalen Daten aus den bestehenden Datenbanken. Der Vorgang des Übertragens der semistrukturierten und relationalen Daten in das HDFS wird in der Projektgruppe als Datenimport bezeichnet.

Es kann abstrahiert werden, dass im Rahmen des Datenimports, Daten von einem Quellsystem in ein Zielsystem zu importieren sind. Die Server stellen die XML-Dateien per FTP zum Abruf bereit. Das HDFS bietet die Möglichkeit über eine Hypertext Transfer Protocol (HTTP) Representational state transfer (REST) Application Programming Interface (API), Befehle an das Dateisystem zu senden. Diese API bietet das gesamte Interface des Dateisystems an und wird als *WebH-*

DFS bezeichnet (vgl. [Apa13b]). Eine weitere Möglichkeit besteht in der Nutzung der Java API von Hadoop (vgl. [Apa16d]). Zusätzlich wird auch eine C Bibliothek bereitgestellt (vgl. [Apa13a]). Beide bieten die Möglichkeit direkt mit dem Dateisystem bzw. mit Hadoop zu kommunizieren.

Für den Import von Daten in das HDFS existieren unterschiedliche Tools am Markt. Um eine Auswahl von Tools für den Einsatz in der Projektgruppe zu treffen, wurde zunächst recherchiert welche Tools sich grundsätzlich eignen. Aus der Recherche ergab sich folgende Liste von Tools, die sowohl eine Anbindung an das HDFS, als auch einen Eingang für FTP besitzen.

- Apache Sqoop
- Apache Storm
- Apache Flume
- Logstash
- Apache Kafka
- Bash
- Gobblin
- Fluentd

Jedes dieser Tools wurde von der Projektgruppe einzelnen evaluiert und getestet. Um eine Bewertung für die Eignung des jeweiligen Tools durchführen zu können werden in Abschnitt 5.1.1 Anforderungen definiert. Das Vorgehen der Evaluation und Tests ist in den nachfolgenden Abschnitten detailliert dokumentiert. Dabei wird jedes Tool zunächst kurz vorgestellt. Anschließend wird die Installation und wenn nötig die Konfiguration des Tools beschrieben. Jeder der Unterabschnitte schließt mit einem Fazit und einer Empfehlung ob das Tool die Anforderungen des Anwendungsfalls erfüllt.

5.1.1. Anforderungen

Grundsätzlich lassen sich die Aufgaben der zu evaluierenden Tools für den Data Import in zwei Bereiche einteilen. Zum einen sollen Daten aus bestehenden relationalen Datenbanken importiert werden können und zum anderen müssen die XML-Dateien in das HDFS importiert werden. Somit werden auch unterschiedliche Anforderungen an die Tools gestellt. Die folgende Tabelle 5.1 fasst die Anforderungen an den Import der XML-Dateien und der Daten aus relationalen Datenbanken zusammen.

Anforderungen - Import relationaler Daten	
TG1DI-01	Es sollen Lese- und Schreibprozesse in MySQL und Hive durchgeführt werden können.
TG1DI-02	Massendaten sollen direkt aus einer MySQL-Datenbank in Hive-Tabellen geladen werden können.
TG1DI-03	Metadaten der Tabellen sollen ausgelesen und übergeben werden können.
TG1DI-04	Es sollen neue Tabellen anhand von Metadaten erstellt werden können.
TG1DI-05	Die Daten in den den Hive-Tabellen sollen wiederholt und gesteuert überprüft und aktualisiert werden können.

Anforderungen - Import der XML-Dateien	
TG1DI-06	Es muss eine Datenschnittstelle per FTP vorhanden sein, um Dateien zu beziehen.
TG1DI-07	Es muss eine Datenschnittstelle zum HDFS vorhanden sein, um Dateien hochzuladen.
TG1DI-08	Konfiguration der zuvor genannten Schnittstellen muss möglich sein.
TG1DI-09	Dateien sollen gemerget bzw. konkateniert werden können.
TG1DI-10	Metadaten (z.B. der Dateiname oder die Dateigröße) sollen in den einzelnen Dateien ergänzt werden können.
TG1DI-11	Sämtliche Aktivitäten des Tools sollen geloggt werden.

Tabelle 5.1.: Anforderungen an die Tools zum Data Import

5.1.2. Apache Sqoop

Apache Sqoop (Abkürzung für *„SQL-to-Hadoop“*) ist ein Open Source Tool, welches den Transfer von Daten zwischen Apache Hadoop und relationalen Datenbanken mittels der MapReduce-Technologie ermöglicht. Die Daten können in das HDFS selbst, in relationale Schemata von Hive oder in die NoSQL-Datanbank HBase übertragen werden. Die Abbildung 5.1 zeigt den Aufbau des Tools. [Sur13]

Im Folgenden wird zunächst erklärt, wie die Daten mit Hilfe von Sqoop in das HDFS importiert bzw. exportiert werden können. Anschließend wird auf die Installation, Konfiguration und Nutzung des Tools eingegangen.

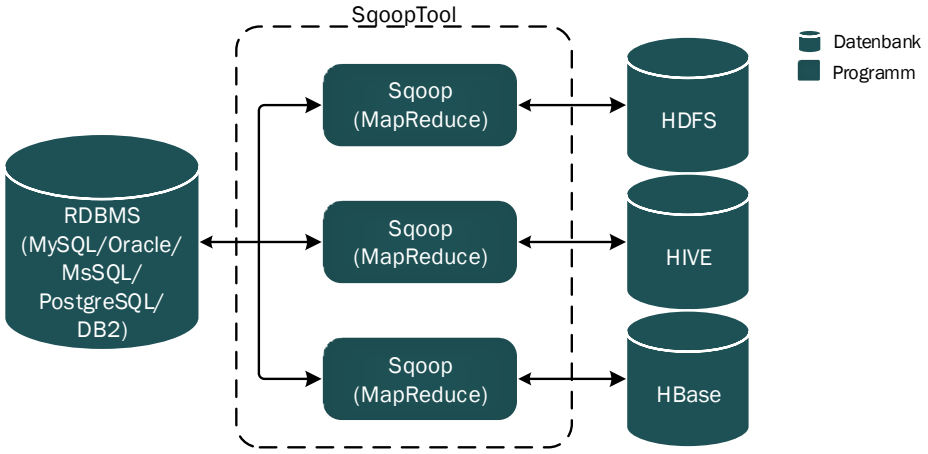


Abbildung 5.1.: Grobarchitektur von Apache Sqoop, eigene Darstellung in Anlehnung an [Sur13]

Im- und Export von Daten

Import in das HDFS: Die Tabellen können sowohl zeilenweise, als auch vollständig als Textdateien oder binär im Arvo-Schema als Sequence-Dateien im HDFS abgelegt werden. Die Struktur wird aus den Metadaten der Datenbank entnommen.

Export aus dem HDFS: Die zu exportierende Datenbank oder Teile von dieser müssen bereits in der Zieldatenbank existieren. Im „update mode“ ersetzt das Tool Daten mittels des UPDATE Statements, im „call mode“ werden Daten durch INSERT-Statements hinzugefügt. [Sev14] [Ico15]

Installation auf dem Hadoop Cluster

Der Sqoop-Server kann an jedem Ort installiert werden, der als Hadoop-Client fungieren kann. Voraussetzung ist hierbei, dass Hadoop installiert und konfiguriert ist, inklusive der Hadoop Libraries. Es ist jedoch nicht notwendig, dass weitere Hadoop-Services auf der Maschine laufen. Der Server steuert beliebig viele Sqoop-Clients, die auf den einzelnen Name-Nodes des Hadoop-Clusters installiert werden, an. Die Clients können auch eigenständig für eine Single-Node-Konfiguration genutzt werden. [Apa13c]

Nachfolgend werden die einzelnen Schritte der Installation erläutert und mit Codebeispielen hinterlegt. Zunächst wird Sqoop heruntergeladen, entpackt und im Zielverzeichnis (bspw. */usr/lib/sqoop*) abgelegt. Der folgende Codeblock zeigt die Ausführung mit der Linux-Commandshell.

Bash 5.1: Sqoop-Archiv entpacken und verschieben

```
$ sudo tar –zxvf sqoop–<VERSIONSBEZEICHNUNG>.tar.gz
$ sudo mv sqoop–<VERSIONSBEZEICHNUNG> /usr/lib/sqoop
```

Anschließend wird die Datei bash.rc um die globalen Variablen von Sqoop erweitert, gespeichert und ausgeführt.

Bash 5.2: bash.rc erweitern und ausführen

```
$ export SQOOP_HOME= usr/lib/sqoop
$ export PATH=$PATH:$SQOOP_HOME/bin
$ source ~/.bashrc
```

Wird bei der Ausführung des folgenden Kommandos die gewählte Sqoop-Version angezeigt, war die Installation erfolgreich.

Bash 5.3: Überprüfung der Sqoop-Installation

```
$ cd $SQOOP_HOME/bin
$ sqoop–version
```

Konfiguration

Nach der Installation muss Sqoop für den jeweiligen Einsatzkontext konfiguriert werden. Dabei spielt die Definition der globalen Variablen im Bash Skript *Sqoop-env.sh* eine große Rolle. Das Template-File wird kopiert, um es als Vorlage für die spezifische Konfiguration von Sqoop zu nutzen. Die kopierte Datei wird um die folgenden Zeilen erweitert:

Bash 5.4: Variablen setzen

```
$ export HADOOP_COMMON_HOME=/usr/local/hadoop
$ export HADOOP_MAPRED_HOME=/usr/local/hadoop
```

Für die Verbindung mit den Relationales Datenbank Management System (RDBMS) müssen zunächst die entsprechenden Treiber zur Kommunikation der Systeme heruntergeladen werden. Diese werden als jar-Dateien im Lib-Verzeichnis von Sqoop abgelegt. Das nachfolgende Beispiel zeigt das Herunter-

laden, Entpacken und Ablegen des MySQL-Connectors.

Bash 5.5: MySQL-Connector entpacken und verschieben

```
1   Download: http://ftp.ntu.edu.tw/MySQL/Downloads/Connector-J/
2   $ tar -zxf mysql-connector-java-5.1.30.tar.gz
3   $ mv mysql-connector-java-5.1.30-bin.jar /usr/lib/sqoop/lib
```

Starten

Der Sqoop-Server wird über *./bin/sqoop.sh server start* gestartet und über *./bin/sqoop.sh server stop* angehalten. Der Server nutzt standardmäßig Port 12000 und 12001. Die genutzten Ports können unter server/bin/setenv.sh durch verändern der Werte *SQOOP_HTTP_PORT* und *SQOOP_ADMIN_PORT* angepasst werden.

Der Sqoop-Client wird mittels folgenden Code gestartet:

Bash 5.6: Sqoop-Client starten

```
1   $ bin/sqoop.sh client
```

Ausführung

Sqoop-Jobs werden über einen Command-Line Befehl gestartet. Dabei werden alle Einstellungen durch Parameter übergeben. Alternativ lassen sich diese Parameter auch als Text-Datei speichern und anschießend durch Sqoop auslesen. Die allgemeine Befehlsstruktur sieht hierbei folgendermaßen aus:

Bash 5.7: Sqoop Befehlsstruktur

```
1   $ sqoop <tool> <Argumente> <toolspezifische Argumente>
```

Die wählbaren Funktionen werden in der Tabelle 5.2 dargestellt.

Alle weitergehenden Parameter der einzelnen Tools und sämtliche Einzelheiten zu der Nutzung von Sqoop können im Sqoop User Guide nachgelesen werden. [Apa16j]

Befehl	Beschreibung
import	Import einer Tabelle aus RDBMS ins HDFS als Text- oder SequenceFile.
import-all-tables	Alles Tabellen einer Datenbank werden separat in Ordnern im HDFS abgelegt.
export	Die zu exportierende Datenbank oder Teile von dieser müssen bereits in der Zieldatenbank existieren. Im textitupdate mode ersetzt das Tool Daten mittels *UPDATE* Statement, im *call mode* werden Daten durch *INSERT* Statements hinzugefügt.
validation	Vergleicht Tabellen mittels zählen der Zeilen (Import oder Export) auf Gleichheit oder prozentuale Übereinstimmung. Kann auch Logmessages (warning/error) ausgeben.
job	Gespeicherte Jobs können hierdurch ausgeführt werden, um wiederkehrende Aufgaben zu erledigen, bspw. um kontinuierlich die neusten Einträge einer DB zu übertragen.
metastore	Zugriff auf gemeinsamen Pool von Job-Files (read/write) von mehreren Usern. Jobfiles werden über textitsqoop job angelegt.
merge	Kombiniert Daten aus zwei Datenbanken, wobei die neueren Daten die älteren überschreiben. Beide werden separat im HDFS abgelegt und enthalten jeweils die neusten Einträge.
codegen	Ermöglicht Erstellung von Meta-Daten Java-Klassen. Beispielsweise bei Verlust dieser Daten.
create-hive-table	Erstellt aus einer Datenbank die bereits ins HDFS importiert wurde oder werden soll, die Metadaten für die Hive-Tabelle. Entspricht dem Hive-Import des import-Tools ohne Übermittlung der Daten.
eval	Gibt einfache SQL-Querys in der Konsole aus. Nützlich zum Abgleich der Erwarteten Resultate mit den Tatsächlichen, bevor die Querys-Ergebnisse importiert werden.
list-databases	Gibt alle Datenbanken des RDBMS zurück.
list-tables	Gibt alle Datenbanken des RDBMS zurück.
help	Sqoop-Wiki für die Konsole wird ausgegeben.
version	Gibt die Sqoop-Version aus.

Tabelle 5.2.: Sqoop-Funktionen

Beispiele

Das erste Beispiel zeigt einen Befehl zum Lesen und Ausgeben aller Tabellen einer MySQL-Datenbank. Dabei werden die Serveradresse sowie die Login-Daten direkt im Statement übergeben.

Bash 5.8: Tabellen einer Datenbank ausgeben

```
$ sqoop list-tables --connect jdbc:mysql://localhost/<
    DatabaseName> --username <mysql-user> --password <PW>
```

Mit dem folgenden Befehl lassen sich sämtliche Tabellen von einer MySQL-Datenbank direkt in das HDFS transferieren. Wie bei der zuvor dargestellten Leseoperation wird auch hier eine Verbindung zur relationalen Datenbank aufgebaut.

Bash 5.9: Alle Tabellen einer Datenbank ins HDFS importieren

```
$ sqoop import-all-tables --connect jdbc:mysql://localhost/<
    DatabaseName> --username <Username> --password <PW>
```

Sqoop erlaubt zudem die Eingabe vollständiger MySQL-Querys. Die Ergebnisse können anschließend in der Console dargestellt oder optional auch im HDFS abgelegt werden.

Bash 5.10: SQL-Anfragen in der MySQL-Datenbank

```
$ sqoop eval --connect jdbc:mysql://db.example.com/corp --query
    "SELECT * FROM employees LIMIT 10"
$ sqoop eval --connect jdbc:mysql://db.example.com/corp -e "
    INSERT INTO foo VALUES(42, 'bar')"
```

Im Kontext der Projektgruppe kann ein Tabellen-Import (MySQL) in Hive nach dem folgenden Schema durchgeführt werden:

Bash 5.11: MySQL-Tabelle in Hive importieren

```
$ sudo sqoop import --connect jdbc:mysql://localhost/<
    DatabaseName>  \
--username <Username> -P --table <TableName> --hive-import \
--hive-table <DatabaseName>.<TableName>
```

Bei Eingabe des Passworts wird aus Sicherheitsgründen empfohlen, das Passwort nicht direkt mit *–password <Passwort>* in das Statement zu schreiben und stattdessen mit dem Argument *-P* eine Eingabeaufforderung der Konsole zu nutzen, wie im vorangegangenen Beispiel. Bevor eine Tabelle in Hive importiert werden kann, muss manuell eine Datenbank mit folgendem Befehl angelegt werden:

Bash 5.12: Hive-Database anlegen

```
1   $ hive> create database <DB_Name>;
```

Sqoop 2

Bei der zweiten Version von Sqoop handelt es sich im Gegensatz zu Sqoop 1 um eine serverbasierte-Lösung. Die Konfiguration und Integration von Connectoren/-Treibern geschieht auf dem Server. Der Datenzugriff wird durch Mapper und nicht wie zuvor über MapReduce-Jobs durchgeführt. Sqoop 2 bietet zudem neben der Steuerung über die Konsole ein Web User-Interface an. [Sri13] Da die Nachfolgerversion noch in der Entwicklungsphase ist, wird empfohlen, bis auf weiteres auf Sqoop 1 zurückzugreifen. [Clo16a]

Fazit

Sqoop ermöglicht es, Daten aus relationalen Datenbanken in das HDFS hinzuzufügen oder diese zu extrahieren. Die Daten können direkt in Hive Tabellen eingefügt werden. Somit können Analysen auf Daten aus dem HDFS um externe Daten, wie beispielsweise Wetterdaten, angereichert werden. Das Tool Sqoop erfüllt alle in Abschnitt 5.1.1 gestellten Anforderungen an den Import relationaler Daten und eignet sich somit für den Einsatz in der Projektgruppe.

5.1.3. Apache Storm

Apache Storm ist ein verteiltes Berechnungs Framework für Event Stream Processing. Initiiert wurde das Projekt von Backtype. Nachdem Twitter im Jahr 2011 das Unternehmen kaufte, wurde das Projekt als Open-Source Projekt über GitHub verfügbar gemacht. Schließlich wurde es zu Apache Incubator hinzugefügt. Seit September 2014 ist Storm ein Apache-Top-Level-Projekt. [Mar14]

In der offiziellen Dokumentation wird Storm wie folgt definiert: „Storm makes it easy to reliably process unbounded streams of data, doing for realtime processing what Hadoop did for batch processing“[Apa15g]. Durch die Kombination von Apache Storm und Hadoop mit Yarn ist ein Hadoop Cluster in der Lage, eine vollständige Palette von Workloads abzudecken. Diese reicht von Echtzeit bis zur interaktiven Batch Verarbeitung.

Storm wurde für massive Skalierbarkeit konzipiert und unterstützt die Fehlertoleranz mit einem, „fail fast, auto restart“-Ansatz für Prozesse. Dadurch bietet Strom die Garantie, dass jedes Tupel verarbeitet wird [Apa15f, Apa15e]. Bei ei-

nem Ausfall des Storm Daemons, Nimbus und des Supervisors, werden diese neu gestartet ohne Auswirkung auf das Cluster oder die Topologien.

Storm ist vor allem in Java und Clojure geschrieben [Apa11] und ist so konzipiert, dass die Unterstützung der Vernetzung von Spouts (Eingangsströme) und Bolts (Verarbeitung und Ausgabe-Module) zusammen als ein gerichteter azyklischer Graph (GAG) betrachtet werden kann. Die Abbildung 5.2 zeigt die Storm Topologie. Topologien laufen auf dem Cluster, wo der Storm-Scheduler die Arbeit auf die Knoten innerhalb des Clusters verteilt. Die Verteilung basiert auf der Topologie-Konfiguration. [Apa14d]

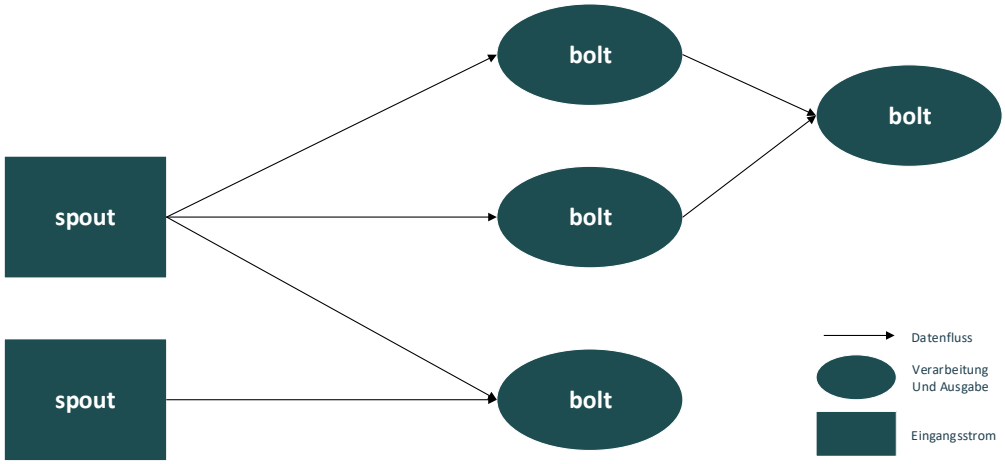


Abbildung 5.2.: Storm Topologie - eigene Abbildung nach [Hor13]

Eine Topologie kann als MapReduce Job in Hadoop angesehen werden, mit dem Unterschied dass sie nicht automatisch terminiert. Ein manuelles Beenden ist jedoch möglich. Dies ist mit dem Fokus auf Echtzeit-Stream basierte Verarbeitung zu begründen. Sobald eine Topologie gestartet wird, bringen die Spouts Daten in das System und geben die Daten an Bolts, wo die Hauptrechenarbeit erledigt wird. Solange die Bearbeitung fortschreitet, können ein oder mehrere Bolts die Daten in eine Datenbank oder ins Dateisystem schreiben. Weiterhin kann auch eine Nachricht an einen anderes externes System gesendet werden. [Apa14c]

Ein Vorteil von Apache Storm ist eine reiche Auswahl an verfügbaren Spouts, die spezialisiert sind, Daten aus verschiedenen Quellen zu empfangen. So gibt es beispielsweise bereits fertige Spouts um Daten über die Twitter-Streaming-API oder von Apache Kafka zu empfangen. Zudem besteht die Möglichkeit, eigenen Spouts und Bolts zu entwickeln [Apa14c]. Im Rahmen dieser Entwicklung werden mehrere Programmiersprachen unterstützt, einschließlich nicht JVM Sprachen [Apa14e]. Es gibt Adapter, die eine einfache Integration mit dem HDFS-

Dateisystem ermöglichen, sodass ein Zusammenspiel beider Technologien problemlos einzurichten ist [Apa14e].

Installation und Konfiguration

In diesem Abschnitt wird die schnelle Installation und Konfiguration von Apache Storm beschrieben. Die Installation erfordert keine `sudo`, Logs oder andere von ZooKeeper und Storm gepflegten Daten. Storm nutzt Zookeeper um das Cluster zu koordinieren [Apa16l].

Zunächst wird ein neues Verzeichnis erstellt in dem Storm heruntergeladen wird.

Bash 5.13: Erstellen und Öffnen eines Verzeichnis für Storm

```
$ mkdir storm
$ cd storm
```

Im nächsten Schritt wird ein Datenverzeichnis für Zookeeper erstellt.

Bash 5.14: Erstellen eines data directorys

```
$ mkdir -p datadir/zookeeper
```

Anschließend kann Zookeeper heruntergeladen und entpackt werden.

Bash 5.15: Herunterladen und Entpacken von ZooKeeper

```
$ wget http://apache.mirrors.spacedump.net/zookeeper/current/
    zookeeper-3.4.6.tar.gz
$ tar -xvf zookeeper-3.4.6.tar.gz
```

Nachdem ZooKeeper heruntergeladen wurde, muss die Konfiguration *Zookeeper-3.4.6/conf/zoo.cfg* angepasst werden.

Config 5.16: Konfigurieren von ZooKeeper

```
tickTime=2000
dataDir=/home/username/datadir/zookeeper
clientPort=2181
```

In *apache-storm-0.9.5/conf/storm.yaml* muss folgendes eingefügt bzw. auskommentiert werden:

Config 5.17: Konfigurieren von Storm

```
storm.zookeeper.servers:
- "127.0.0.1"
nimbus.host: "127.0.0.1"
storm.local.dir: "/home/username/storm/datadir/storm"
supervisor.slots.ports:
- 6700
- 6701
- 6702
- 6703
```

Nun kann ZooKeeper gestartet werden.

Bash 5.18: Starten von ZooKeeper

```
$ zookeeper-3.4.6/bin/zkServer.sh start
```

In einer separaten Shell oder im Hintergrund wird Nimubs gestartet.

Bash 5.19: Starten von Nimbus

```
$ apache-storm-0.9.5/bin/storm nimbus
```

Auch der Storm Supervisor wird über eine separate Shell oder im Hintergrund gestartet:

Bash 5.20: Starten von supervisor

```
$ apache-storm-0.9.5/bin/storm supervisor
```

Gleiches gilt auch für das Starten des User Interface (UI).

Bash 5.21: Starten der UI

```
$ apache-storm-0.9.5/bin/storm ui
```

Wenn der Zugriff auf `127.0.0.1:8080` klappt, ist Storm erfolgreich installiert. Dieses Verfahren erzeugt vier Worker. Wenn mehr oder weniger Worker gebraucht werden, muss `supervisor.slots.ports` entsprechend angepasst werden.

Fazit

Apache Storm ist ein skalierbares, schnelles, fehlertolerantees Open-Source-System für verteilte Berechnungen, mit dem Fokus auf Datenstromverarbeitung. Storm zeichnet sich bei der Ereignisverarbeitung und inkrementellen Berechnungen aus. Storm bietet die Möglichkeit verschiedenste Funktionen aufzurufen und kann daher theoretisch verwendet werden, um fast jede verteilte Berechnung durchzuführen. Die Stärke von Storm liegt aber eindeutig in der Verarbeitung von Ereignisströmen.

Storm erfüllt einige der Anforderungen, die in Kapitel 5.1.1 aufgestellt wurden. Insbesondere bietet Storm Schnittstellen zum HDFS, sowie Module für den Zugriff auf FTP. Da in der Projektgruppe aber nicht mit Datenströmen gearbeitet wird, sondern mit XML Dateien die täglich in das HDFS übertragen werden sollen, wird Storm im Rahmen dieser Projektgruppe dennoch nicht weiter betrachtet.

5.1.4. Apache Flume

Apache Flume ist ein Top-Level Projekt der *Apache Software Foundation* und ist eine Komponente des Hadoop Ecosystems. Flume wurde entwickelt, um große Mengen an Logdaten zu sammeln, zu aggregieren und weiterzuleiten. [Arv11]
Im Folgenden wird evaluiert, ob Flume die in Abschnitt 5.1.1 gestellten Anforderungen an ein Tool zum Datenimport im Rahmen der Projektgruppe erfüllt. Um Daten von einer Quelle zum Ziel übertragen zu können, werden intern unterschiedliche Komponenten verwendet. Diese Komponenten werden im folgenden Abschnitt näher erläutert.
Für den Datentransfer mit Flume wird in einer Konfigurationsdatei die Datenquelle (Source), der Übertragungskanal (Channel) und das Ziel (Sink) bestimmt. Ein Flume Agent bildet den Prozess zum Datentransfer ab und beinhaltet die Source, den Channel und die Sink. In der folgenden Abbildung 5.3 ist der Flume Agent mit seinen Komponenten abgebildet. Es wird der Prozess dargestellt, wie die Daten aus der Source über den Channel in die Sink gelangen.
Die Abbildung zeigt, dass für einen Agenten mehrere Zielstrukturen konfiguriert werden können. Für das weitere Verständnis werden die Komponenten, die an einem diesem Prozess beteiligt sind, näher erläutert.

Event: Ein *Event* ist ein Flume-Objekt, das die Daten repräsentiert, die von der

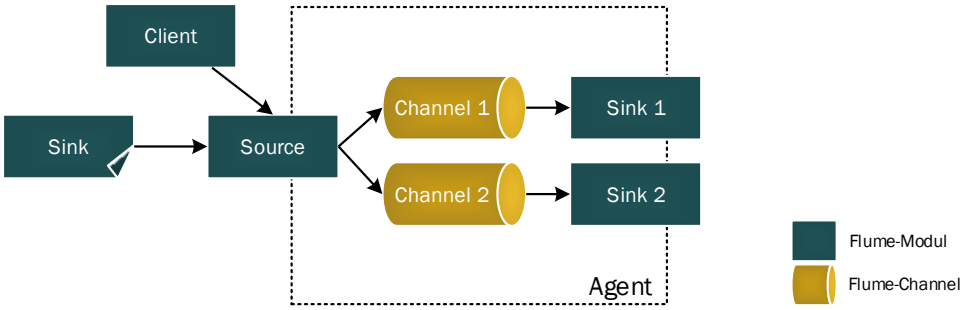


Abbildung 5.3.: Flume Agent [Lea11]

Quelle zum Ziel übertragen werden. Das Objekt beinhaltet sowohl Meta-Informationen über die Daten als auch die Daten selbst.

Agent: Dies ist ein unabhängiger Prozess, welcher die Übertragung vom Quell- in das Zielsystem steuert. Durch einen *Agenten* werden die Daten durch die verschiedenen Flume-Komponenten geleitet. Sobald die Daten im Zielsystem hinterlegt sind, wird der Flume-Prozess beendet.

Client: Ein *Client* ist ein externer Prozess, der die Ausführung eines Flume-Agenten starten kann. Flume bietet Schnittstellen für die Anbindung an verschiedene Software und Protokolle. Hierzu zählen unter anderem *Avro*, *log4j*, *syslog* und das *Http POST* Protokoll.

Source: Die *Source* ist eine Komponente, die auf Events von den *Clients* wartet. Hierfür werden verschiedene Schnittstellen zur Verfügung gestellt. Durch eine *Source* kann ein *Agent* gestartet werden.

Channel: Der *Channel* ist eine vorübergehende Datenablage. Nachdem die Daten aus der *Source* importiert wurden, werden diese im *Channel* hinterlegt, bis die *Sink* bereit für eine Aufnahme ist.

Sink: Die *Sink* repräsentiert die Zieldatenstruktur. Die *Sink* ist in der Lage, die *Events* aus dem *Channel* zu entfernen und die Daten in die tatsächliche Zielstruktur zu übertragen.

Konfiguration

Die Konfiguration von Flume erfolgt durch die Anpassung der drei Config-Dateien: *flume.conf*, *flume-env.sh* und *log4j.properties*. In der folgenden Tabelle 5.3 werden die Funktionen dieser Dateien kurz erläutert.
Die Anforderung TG1DI-06, die in Abschnitt 5.1.1 festgelegt wurde, fordert einen FTP-Server als Source. Dafür wurde das Plug-In *keedio/flume-ftp-source* genutzt [Kee15]. Damit pro Event eine Datei generiert wird, wurde ein *Deserializer* imple-

Datei	Beschreibung
flume.conf	In dieser Datei werden die Komponenten *Source*, *Client*, *Channel* und *Sink* deklariert. Es werden hierfür Ports festgelegt, auf welchen Flume lauscht, um gegebenenfalls einen *Agent* zu starten.
flume-env.sh	Ist das Starterskript für Flume. In diesem können weitere Konfigurationen durch Laufzeitparameter übergeben werden.
log4j.properties	Durch diese Datei kann das *Logging* von Flume deklariert werden.

Tabelle 5.3.: Flume Config-Dateien

mentiert. Der Deserializer wurde selber entwickelt und muss in der Config-Datei angegeben werden. Weiterhin wird das FTP-Plug-in von *Keedio* von GitHub heruntergeladen[1], in den Unterordner *flume-test/lib* kopiert und gebaut. Dies Plugin wird ebenfalls in der Config-Datei angegeben.
Im Folgenden wird die Config-Datei vorgestellt, die im Rahmen der Evaluation erstellt wurde.

Config 5.22: Config-Datei „flume-conf“

```
1  # agent components
2  sandbox.sources = ftps1
3  sandbox.sinks = sink1
4  sandbox.channels = chan1
5
6  # ftps source
7  sandbox.sources.ftps1.type = org.keedio.flume.source.ftp.source.
       Source
8  sandbox.sources.ftps1.client.source = ftps
9  sandbox.sources.ftps1.name.server = 134.106.13.145
10 sandbox.sources.ftps1.port = 21
11 sandbox.sources.ftps1.user = ftpuser
12 sandbox.sources.ftps1.password = dash_elefant
13 sandbox.sources.ftps1.folder = /var/log/flume
14 sandbox.sources.ftps1.file.name = ftps1-status-file.ser
15 sandbox.sources.ftps1.security.enabled = true
16 sandbox.sources.ftps1.security.cipher = TLS
```

[1]Die Quelldateien sind unter folgendem Link abrufbar: https://github.com/keedio/flume-ftp-source/blob/flume_ftp_dev/src/main/resources/example-configs/flume-ng-ftp-source-SFTP.conf

```
17  sandbox.sources.ftps1.security.certificate.enabled = false
18  sandbox.sources.ftps1.deserializer = xmlDeserializer.
        DfmXmlDeserializer$Builder
19  # discover delay, each configured milisecond directory will be
        explored
20  sandbox.sources.ftps1.run.discover.delay = 1000
21
22  # configuring the sink
23  sandbox.sinks.sink1.type = hdfs
24  sandbox.sinks.sink1.hdfs.path = /tmp/flume_sink
25  sandbox.sinks.sink1.hdfs.useLocalTimeStamp = true
26  sandbox.sinks.sink1.hdfs.batchSize = 10000
27  sandbox.sinks.sink1.hdfs.fileType = DataStream
28  sandbox.sinks.sink1.hdfs.filePrefix = ceweData
29  # creating log file in hdfs based on the roll size
30  sandbox.sinks.sink1.hdfs.rollCount = 0
31  sandbox.sinks.sink1.hdfs.rollSize = 10737418240
32  sandbox.sinks.sink1.hdfs.rollInterval = 216000
33
34  # configuring the channel
35  sandbox.channels.chan1.type = file
36  # checkpoint directory
37  sandbox.channels.chan1.checkpointDir = /var/tmp/flume_checkpoint
        /.flume/file-channel/checkpoint
38  # log files created in data directory while running flume-agent
39  sandbox.channels.chan1.dataDirs = /var/tmp/flume_checkpoint/.
        flume/file-channel/data
40
41  # bind the source and sink to the channel
42  sandbox.sources.ftps1.channels = chan1
43  sandbox.sinks.sink1.channel = chan1
```

Die Config-Datei ist in fünf Teilbereiche gegliedert.
Im ersten Bereich werden die Komponenten des Flume Agents definiert.
Im zweiten Teil wird der Source definiert und weitere Config-Details zum Source gepflegt. Hierbei wird vor allem FTP als Source definiert und die zugehörigen Details bezüglich Authentifizierung festgehalten. Weiterhin wird in diesem Teilbereich der selbstentwickelte Deserializer aufgenommen. Auf diesem wird im weiteren Verlauf dieses Kapitels eingegangen. Im dritten Teilbereich der Config-Datei wird der Sink definiert. In diesem Fall werden alle Eigenschaften zum HDFS gepflegt. Dazu gehört vor allem der Pfad zum Zielordner und die abzulegenden Dateinamen.
Im vierten Teil der Config-Datei wird der Channel definiert, durch den die Daten vom Source zum Sink gelangen. Hier wird beispielsweise der Typ als „file“

gefplegt. Eine weitere Variante wäre beispielsweise der Typ „memory“.
Im letzten Teil wird die Source mit dem Sink über den Channel verbunden.

Um die Einsetzbarkeit von Flume für die Projektgruppe zu testen, wurde ein Funktionstest auf einer Virtuelle Maschine (VM) durchgeführt. Dazu wurde zunächst ein Quellordner angelegt, aus welchem Flume die Dateien entnehmen soll. Außerdem wurde im HDFS ein Zielordner erstellt, in den Flume die Dateien entsprechend wieder ablegt. Dieser wurde wie folgt über die Shell angelegt und mit entsprechenden Lese- und Schreibrechten ausgestattet:

Bash 5.23: Anlegen eines Verzeichnisses im HDFS als Flume-Sink

```
1   $ hadoop fs -mkdir -p /tmp/flume_sink
2   $ hadoop fs -chmod -R 777 /tmp/flume_sink
```

Nun können die Flume-Agenten mit folgendem Befehl gestartet werden:

Bash 5.24: Start der Flume-Agenten

```
1   $ flume-ng agent -n sandbox -c conf -f /etc/flume/conf/sandbox/
        flume.conf -C /flume-test/lib/flume-ftp-source-2.0.8.jar
```

Nach Start der Agenten beginnt Flume damit, die Dateien aus der Quelle in die Zielstruktur zu übertragen. Die folgende Tabelle 5.4 erläutert die Run-Parameter.

Parameter	Beschreibung
flume-ng	Der Flume Dienst
agent	Der Flume Agent.
-n sandbox	Ausführung im Sandbox Modus
-c	Nutzung einer KonfigurationsDatei
-f	Pfad der KonfigurationsDatei

Tabelle 5.4.: Flume Run-Parameter

Probleme

Zu Beginn wurde das *„keedio/flume-ftp-source“*, ohne Deserializer benutzt. Es wurde festgestellt, dass durch die Übertragungsvarianten von Flume, die gelesenen Dateien nicht als vollständige Datei behandelt werden. Flume behandelt Dateien

entweder Zeileweise oder Chunkweise. Eine kurze Erläuterung ist im Folgenden dargestellt.

- *Zeilenbasiert*: Die XML-Datei wird Zeile für Zeile gelesen und es werden hierfür mehrere Events generiert.
- *Chunkbasiert*: Bei bis zu einem Kilobyte wird jeweils ein Event generiert und übertragen. Dies geschieht bis alle XML-Dateien gelesen wurden.

Es konnte festgestellt werden, dass durch diese Art der Verarbeitung die XML-Dateien getrennt wurden und somit kein valides XML mehr vorhanden ist. Um dem entgegen zu wirken, wurde der bereits erwähnte Deserializer implementiert. Durch ihn, ist es möglich eingelesene Stücke zu sammeln und als komplette Datei in einem Event weiter zu geben. Druch das Rollover-Verhalten von Flume, ist diese Lösung aber nicht vollständig. Es existiert in der Konfigurationsdatei von Flume ein Parameter (rollSize) der angibt wie groß die Zieldatei im HDFS werden muss, bis eine neue Datei erstellt wird. Beispielsweise ist diese Größe mit 100MB definiert. Die XML-Dateien werden solange zu einer Datei zusammengefasst, bis diese Größe erreicht wird. In einigen Fällen konnten XML-Dateien unzerstückelt übertragen werden, in anderen Fällen wiederum nicht. Da sie zu einem rollover führen und die zweite Hälfte der XML-Datei in einer neuen Datei im HDFS landet.

Fazit

Wie der vorangestellte Abchnitt zeigt, ist es der Projektgruppe nicht gelungen, kopmlette, valide XML-Dateien über Flume in das HDFS zu übertragen, da Flume Dateien nur Zeilenweise bzw. Chunkweise berbeitet und pro Zeile beziehungsweise Chunk ein Event generiert. Durch die Eigenentwicklung des Deserializers, konnte erreicht werden, dass pro Datei ein Event generiert wird. Durch das rollover Verhalten wurden dennoch ungültige XML-Datien im HDFS erzeugt. Somit erfüllt Flume die Anforderung TG1DI-06, aus Abschnitt 5.1.1 nicht und wird im Rahmen der Projektgruppe nicht weiter betrachtet.

5.1.5. Logstash

Im Rahmen der Evaluierung möglicher Tools zum Datenimport wurde Logstash betrachtet. Es wird zunächst kurz dargestellt, aus welchem Umfeld Logstash entstand und wie es heute eingesetzt wird. Daran anknüpfend wird die Architektur von Logstash dargestellt. Im Anschluss daran wird die Installation und Konfiguration dieses Tools beschrieben. Schließen wird dieser Abschnitt mit einem Fazit, ob das Tool für den Anwendungsfall genutzt werden kann.

Logstash kommt aus dem Umfeld der Log-Aggregierung und -Verarbeitung.

Mittlerweile gibt es allerdings über 200 Plug-Ins die es ermöglichen, neben Log-Dateien, auch Dateien anderer Art zu verarbeiten. Weiter bietet Logstash eine API an, die es Entwicklern ermöglicht, eigene Plug-Ins für Logstash zu entwickeln. [Ela16b]

Grundsätzlich besteht eine Logstash Instanz aus drei Komponenten:

- Das *Input Plug-In* definiert, welche Daten von welcher Quelle wohin geleitet werden müssen.
- Das *Filter Plug-In* beschreibt, wie bestimmte Daten verarbeitet oder gegebenenfalls auch angereichert werden müssen und an welchen Output sie gehen sollen.
- Das *Output Plug-In* nimmt die verarbeiteten Daten des *Filter Plug-Ins* entgegen und schreibt sie entsprechend der Konfiguration in ein Zielsystem.

Wie aus der Beschreibung des Input Plug-Ins und des Output Plug-Ins abzuleiten ist, existiert ein Quell- und ein Zielsystem.

Die Architektur der beschriebenen Komponenten wird in Abbildung 5.4 dargestellt. Hierbei dient die Datasource als Input und Elasticsearch exemplarisch als Output. Da das Filter Plug-In eine optionale Komponente darstellt, wurde es in der Abbildung blau hervorgehoben.

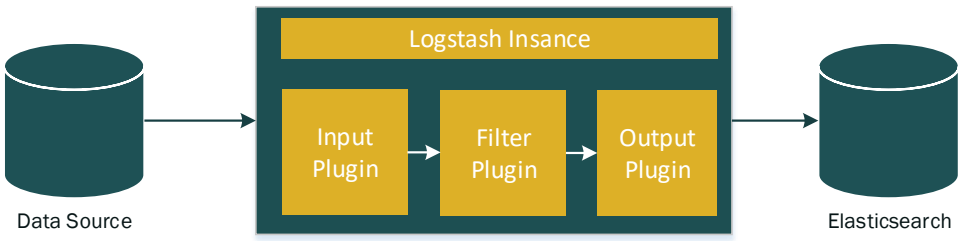


Abbildung 5.4.: Logstash Architektur eigene Darstellung in Anlehnung an [Ela16a]

Installation und Konfiguration

Um Logstash installieren zu können, muss zunächst der *public signing key* heruntergeladen und dem *package repository* hinzugefügt werden. Nun kann das Programm über *apt* heruntergeladen und installiert werden. Der Folgende Code-Block zeigt diese Schritte und die zugehörigen Bash-Befehle:

Bash 5.25: Herunterladen und installieren von Logstash

```
$ sudo wget -qO - https://packages.elastic.co/GPG-KEY-
    elasticsearch | sudo apt-key add -
$ sudo echo "deb http://packages.elastic.co/logstash/2.1/debian
    stable main" | sudo tee -a /etc/apt/sources.list
$ sudo apt-get update && sudo apt-get install logstash
```

Logstash ist nun installiert und kann konfiguriert werden. Eine Konfiguration kann über zwei Wege an Logstash übergeben werden. Der erste Weg, ist über das direkte beschreiben einer Konfiguration (als String) beim Aufruf von Logstash als Parameterwert. Alternativ kann eine Konfigurationsdatei angelegt werden, die beim Aufruf von Logstash als Parameter übergeben wird. Eine Logstash Konfiguration besteht dabei aus den zuvor genannten Komponenten. Es werden die Inputs, Filter (optional) und Outputs definiert. Da es mehrere Instanzen dieser Komponenten geben kann, wird eine direkte Konfiguration beim Aufruf von Logstash schnell unübersichtlich und nicht wartbar, weswegen der Weg der Konfigurationsdatei gewählt wurde.

In der Standardinstallation von Logstash ist kein Output Plug-In für das HDFS vorhanden. Allerdings existiert ein Community Plug-In, welches diesen Output definiert. Dies muss der Logstash Konfiguration mittels folgendem Befehl hinzugefügt werden (der Befehl wird aus dem Installationsordner von Logstash ausgeführt):

Bash 5.26: Herunterladen und installieren von Logstash

```
$ sudo bin/plugin install logstash-output-webhdfs
```

Als erster Test, soll nun eine lokale Datei durch Logstash in das HDFS einer *HortonWorks Sandbox* geschrieben werden. Dazu wurde eine Konfigurationsdatei erstellt, deren Inhalt nachfolgend zu sehen ist.

Bash 5.27: Logstash Konfiguration

```
input {
  file {
    path => "/var/fpt_home/*.xml"
    start_position => "beginning"
  }
}
output {
  webhdfs {
```

```
9         host => "10.0.1.3"
10        path => "/foo/test.xml"
11        user => "admin"
12      }
13    }
```

Erkennbar ist eine Konfiguration des Inputs, die sämtliche Dateien in dem Ordner *fpt_home* einlesen soll. Der Output ist so konfiguriert, dass die Dateien über das WebHDFS auf dem definierten Host in den entsprechenden Ordner geschrieben werden. Anzumerken ist, dass kein Filter Plug-In verwendet wird, da es sich in diesem Anwendungsfall lediglich um den Transport der Dateien handelt, nicht aber um eine mögliche Verarbeitung. Die Konfigurationsdatei kann syntaktisch mit dem Befehl mittels *–configtest* überprüft werden. Dabei werden auch mögliche Netzwerkverbindungen getestet. Bei erfolgreichem Test wird die Zeile *Configuration OK* in der Kommandozeile angezeigt.

Fazit

Trotz erfolgreichem Konfigurationstest ist es nicht möglich gewesen, die Dateien in das HDFS zu transportieren. Auch nach mehrfachem Testen und neuer Konfiguration konnte das Problem nicht gefunden und somit nicht gelöst werden. Weiterhin konnte Logstash in der Testphase keine neu hinzugefügten Dateien im Ordner erkennen, was für den vorliegenden Anwendungsfall jedoch essenziell ist. Durch die aufgezeigten Probleme, ist Logstash als mögliches Tool im Rahmen der Projektgruppe ausgeschieden und wird nicht weiter betrachtet.

5.1.6. Apache Kafka

In diesem Abschnitt wird das Tool Apache Kafka vorgestellt. Kafka wurde ebenfalls im Rahmen der Suche nach einem Tool zum Import der XML-Dateien in das HDFS untersucht und seine Nutzbarkeit im Rahmen der Projektgruppe evaluiert. Zunächst wird das Tool vorgestellt, anschließend wird auf die Installation und Konfiguration in einer Testumgebung eingegangen. Auf Basis der Ergebnisse dieses Testszenarios wird die Nutzbarkeit abschließend in einem Fazit zusammengefasst.

Apache Kafka wurde durch das Karrierenetzwerk LinkedIn entwickelt. Es ist seit 2011 ein Apache-Incubator-Projekt, was bedeutet, dass es noch zahlreiche Änderungen bis zur finalen Version 1.0 geben wird. Kafka ist ein in Scala implementierter, verteilter Open-Source-Message-Broker, der auch eine eigene Java-API bietet und somit auch mit dieser Programmiersprache für die Anwendungsentwick-

lung verwendet werden kann. Es besticht durch die persistente Speicherung der Nachrichten, einen hohen Transaktionsdurchsatz und eine gute Skalierbarkeit. Mit Kafka ist es möglich, offline und online mit zeitnaher Verarbeitung das Bindeglied zwischen Extraktion und Laden umzusetzen. Interessant ist hierbei vor allem sein Einsatz im Hadoop-Ecosystem, für das es zahlreiche Plugins mitbringt. [Sha14]

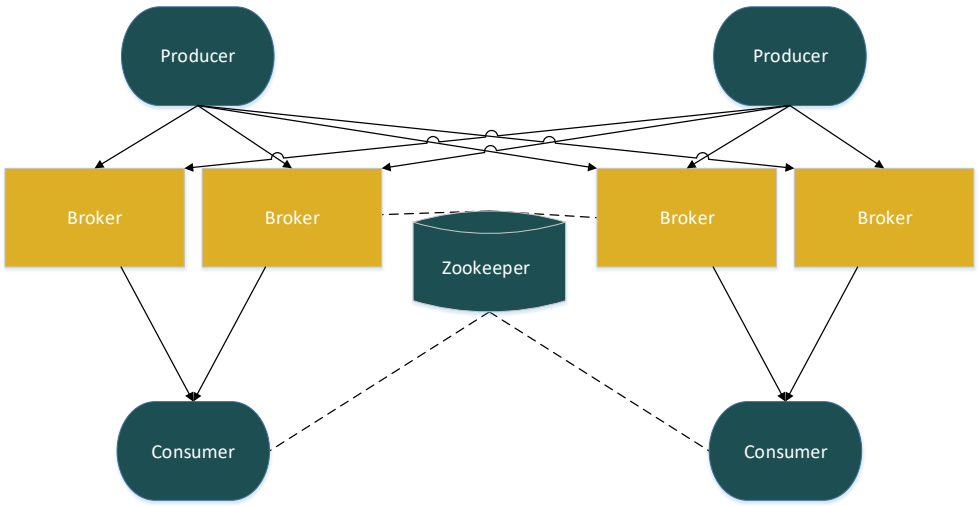


Abbildung 5.5.: Apache Kafka-Architektur, eigene Darstellung in Anlehnung an [Rao13]

Kafka besteht aus drei Komponenten, deren Zusammenhänge in Abbildung 5.5 dargestellt sind. Jede der Komponenten kann auf ein oder mehrere physikalische Systeme verteilt sein [Maz15]. Im Folgenden wird die Funktionsweise kurz beschrieben:

1. *Producer* erfassen die Daten und publizieren diese zu einem gewählten Topic beziehungsweise Thema.
2. *Broker* nehmen anschließend die Daten entgegen.
3. Die *Consumer* rufen die Daten eines gewählten Topics ab und führen die eigentliche Verarbeitung durch.

Apache Zookeeper ist ein zentralisierter Service für die Erstellung und Verwaltung von Kafka-Clustern. So registriert sich jede Instanz des Kafka-Brokers bei diesem Dienst, über den er dann Konfigurationsinformationen erhält und synchronisiert. Insbesondere ist Zookeeper für die Verteilung von Nachrichten im Cluster verantwortlich und instruiert Kafka dahingehend. [RK15]

Installation und Konfiguration

Bei der Installation von Kafka werden zwei Möglichkeiten unterschieden. Die erste Variante der Installation besteht darin, eine reine Apache-Version von Kafka herunterzuladen und zu entpacken. Diese enthält nur die Basiskomponenten, die jedoch keinerlei Unterstützung für die Anbindung an das HDFS enthalten.

Bash 5.28: Herunterladen von Apache Kafka

```
$ curl -O https://www.apache.org/dyn/closer.cgi?path=/kafka
    /0.9.0.0/kafka_2.11-0.9.0.0.tgz
$ mkdir /opt/kafka && cd /opt/kafka
$ tar xfvz kafka_2.11-0.9.0.0.tgz
```

Die zweite Variante der Installation ist die Verwendung der Confluent Data Platform, welche zusätzlich zu den normalen Kafka-Komponenten einige Erweiterungen beinhaltet. Insbesondere die Erweiterung von Kafka-Connect, die eine Schnittstelle zum HDFS benötigt ist im Rahmen der Projektgruppe nützlich.

Bash 5.29: Herunterladen der Confluent Data Platform

```
$ curl -O http://packages.confluent.io/archive/2.0/confluent
    -2.0.1-2.11.7.zip
$ mkdir /opt/kafka && cd /opt/kafka
$ unzip confluent-2.0.1-2.11.7.zip
```

Nach dem Entpacken ist Kafka installiert und kann gestartet werden. Über einen eigenen Kafka-User wird er Zookeeper-Server gestartet. Sobald der Zookeeper bereit ist, können eine oder mehrere Instanzen von Kafka gestartet werden.

Bash 5.30: Kafka starten

```
$ su - kafka
$ cd /opt/kafka/
$ bin/zookeeper-server-start -daemon config/zookeeper.properties
$ bin/kafka-server-start -daemon config/server.properties
```

Es gibt zwei Möglichkeiten, um Daten von Kafka in das HDFS zu transportieren [Pla15]. Die erste Möglichkeit ist es, das Standarddateiformat JavaScript Object Notation (JSON) zu benutzen:

Bash 5.31: Kafka Connect mit JSON starten

```
1   $ bin/connect-standalone etc/kafka/connect-standalone.properties
        etc/kafka-connect-hdfs/quickstart-hdfs.properties
```

Die zweite Möglichkeit besteht darin, das binäre Dateiaustauschformat Apache Avro zu verwenden:

Bash 5.32: Kafka Connect mit Avro starten

```
1   $ bin/connect-standalone etc/schema-registry/connect-avro-
        standalone.properties etc/kafka-connect-hdfs/quickstart-hdfs
        .properties
```

Probleme

Die Verwendung von Apache Kafka mit XML bereitet einige Probleme. So gibt es zwar ein Unterprojekt namens Kafka Connect. Dieses unterstützt jedoch nur die Verwendung von entweder dem textbasierten Format JSON oder dem binärbasierten Avro. Beide Dateiformate sind bei der Verwendung mit Kafka jedoch strukturiert und daher nicht kompatibel mit einer schemalosen beziehungsweise semistrukturierten XML-Datei.

Fazit

Die Vorteile von Apache Kafka sind zahlreich. Es ist Open-Source und von den in der Projektgruppe präferierten Programmiersprachen Java sowie Scala aus einfach nutzbar. Jeder Server und jedes Tool kann voneinander separiert werden, so dass jede Komponente von der anderen unabhängig veränderbar wird. Größere Verarbeitungscluster lassen sich auch einfach realisieren, da Kafka umfangreiche Worker-Strategien mitbringt.

Nachteilig ist jedoch, dass es sich um eine nicht stabile API (0.9.0.0 ...) handelt. Da es sich noch im Incubator der Apache Foundation befindet, könnte es sich noch beträchtlich bis zur Version 1.0 verändern. Zudem ist auch für kleine Installationen die etwas umständliche Handhabung von Apache Zookeeper zwingend erforderlich.

Durch die nicht gegebene Unterstützung des Transports von XML-Dateien ist Kafka zum jetzigen Zeitpunkt nicht für die Projektgruppe einsetzbar und wird daher nicht weiter betrachtet.

5.1.7. Bash (Linux Shell)

Gemäß Ockhams Rasiermesser wurde, neben der Evaluation bestehender Tools, nach einer möglichst einfachen Lösung für den Dateitransfer vom FTP-Server in das HDFS gesucht. Dabei entstand ein Skript, welches die folgenden vier Linux-Tools verwendet: Die Bash-Shell, das Dateisystem SSHFS, die Schnittstelle INotify und den Hadoop-Client.

Bash ist eine Shell für Unix-basierte Systeme wie zum Beispiel das Betriebssystem Ubuntu Linux. *SSHFS* ist ein Dateisystem für Linux, welches eine Möglichkeit darstellt, entfernte Dateisysteme in das eigene einzubinden. *INotify* ist eine Linux-Kernel-Schnittstelle zur Überwachung von Verzeichnissen und Dateien. Sie wird im Skript verwendet, um den FTP-Server auf neu hochgeladene Dateien hin zu überwachen. Der *Hadoop-Client* bietet zwei Methoden, um eine neue XML-Datei per Kommandozeile in das HDFS hochzuladen: *PUT* oder *APPEND*. Eine einzelne Datei wird mit Hilfe von PUT im HDFS gespeichert, während auch mehrere Dateien mit APPEND in eine große kombiniert werden können, zum Beispiel in eine pro Tag.

Installation und Konfiguration

Der Administrator kann den Benutzernamen, das überwachte sowie das Zielverzeichnis ändern (siehe Variablen) und das Skript auf den Hadoop-MasterNode kopieren. Das Skript kann dann entweder manuell im Vordergrund oder auch automatisch wie jeder andere Service in Linux im Hintergrund gestartet werden. Im ersten Schritt stellt das in Config 5.33 dargestellte Skript eine Verbindung zwischen dem FTP-Server und dem Master-Node her.

Bash 5.33: Verbindung des Clusters zum FTP-Server herstellen

```
1  #!/bin/sh
2  sshfs fhaidary@134.106.13.145:/var/www/ftp/ ~/myftp
```

Config 5.34 zeigt den Quellcode für den automatischen Upload der Dateien in das HDFS.

Bash 5.34: Skript für die Überwachung des FTP-Servers und Upload ins HDFS

```
1  #!/bin/bash
2  FOLDER=xml_dateien_neu
3  REMOTE=/var/www/ftp/$FOLDER/
4  LOCAL=/opt/hadoop/myftp/$FOLDER
```

```
5   DEST=/cewe/
6   ssh fhaidary@134.106.13.145 /home/fhaidary/watch -mrq -e create
        --format %f $REMOTE | while read FILE
7   do
8   /opt/hadoop/bin/hadoop fs -put $LOCAL/$FILE $DEST
9   NOW=$(date +"%Y-%m-%d %T.%3N")
10  echo "$NOW - Upload von $FILE nach $DEST erfolgreich" >>upload.
        log
11  done
```

In den Zeilen zwei bis 5 werden zunächst Variablen definiert. Die Variable *FOLDER* enthält den Speicherort der XML-Dateien auf dem FTP-Server. In der Variablen *REMOTE* ist das Wurzelverzeichnis des FTP-Servers hinterlegt. In *LOCAL* wird der lokale Einhängepunkt definiert. Schließlich wird in *DEST* das Zielverzeichnis im HDFS angegeben.

Nach den Variablendeklarationen folgt der Start des *watch*-Skript zur Überwachung des FTP-Servers per Secure Shell (SSH)-Client auf dem entfernten System. Der Parameter *m* steht für monitor, *r* für rekursiv, *q* für quiet, *e* für Event und *format* für die Ausgabe. Es wird also das Remoteverzeichnis rekursiv auf das create-Event hin überwacht und dabei nur der Name der jeweils neu erstellten Datei in der Variable *FILE* abgelegt.

Innerhalb der folgenden *while*-Schleife wird jedes Mal der HDFS-Client aufgerufen, um die neue Datei in den *cewe*-Ordner hochzuladen. Schließlich wird durch die Erzeugung eines Zeitstempels in Kombination mit dem Dateinamen eine Logzeile in der Logdatei *upload.log* angefügt.

Probleme

Die INotify-Schnittstelle ist eine Linux-spezifische Angelegenheit und muss daher bei Portierung auf ein anderes Betriebssystem angepasst werden. Dieser Part des Skripts ist zudem nur erforderlich, wenn der FTP-Server nicht direkt bei jedem Upload ein Skript ausführen kann. Der Rest des Skriptes kann trotzdem weiterverwendet werden.

Fazit

Das Tool ist zwar geeignet für den Transport einzelner oder mehrerer XML-Dateien ins HDFS, bietet jedoch wenig Möglichkeiten zur Konfiguration und Skalierung. Ebenso ist keine Ergänzung von Metadaten zu den Dateien möglich, wie es in Abschnitt 5.1.1 gefordert wird. Daher wird auch diese Lösung im Rahmen der Projektgruppe nicht weiter betrachtet.

5.1.8. Gobblin

In diesem Abschnitt wird das Tool Gobblin vorgestellt. Dabei wird das Tool zunächst kurz vorgestellt und anschließend auf Aspekte der Installation und Konfiguration eingegangen. Im Anschluss werden Probleme, die bei der Nutzung aufgetreten sind beschrieben und ein Fazit zum Gebrauch in der Projektgruppe gezogen.

Gobblin ist ein universelles Framework zum Extrahieren, Transformieren und Laden von Massendaten aus verschiedenen Datenquellen in Hadoop. Datenquellen können unter anderem Datenbanken, REST APIs oder FTP Server sein [Lin15a].

Das Java-basierte Tool wurde von LinkedIn entwickelt und befindet sich aktuell noch im Entwicklungsstatus. Die für die Tests verwendete und aktuellste Version ist 0.6.2 [Lin15b].

Gobblin erledigt eine Reihe an Routineaufgaben, die für den ELT-Prozess erforderlich sind. Hierzu zählen unter anderem das Scheduling von Jobs, die Partitionierung von Tasks, Fehlerbehandlung, Zustandsverwaltung, Datenqualitätsprüfung und das Speichern in ein Zielverzeichnis. Gobblin überträgt Daten aus verschiedenen Datenquellen in das gleiche Zielsystem und verwaltet die Metadaten der verschiedenen Quellen zentral an einem Ort [Lin15b]. Gobblin kann entweder im Standalone-Modus oder im MapReduce-Modus laufen [Lin15c].

Ein Gobblin Job besteht aus einer Reihe von Komponenten, die für die einzelnen Arbeitsschritte des ELT-Prozesses zuständig sind. Alle Komponenten können durch die Job-Konfiguration einzeln hinzugefügt oder bestehende Jobs durch einzelne Komponenten erweitert werden [Lin15c].

Jeder Gobblin Job beinhaltet minimal mehrere Konstrukte, z.B. *Source*, *Extractor*, *Datawriter* und *DataPublisher*. Dabei definiert *Source* die Quelle, von der die Daten entnommen werden. Der *Extractor* implementiert die Logik, um Datensätze zu extrahieren. Der *Datawriter* definiert die Art und Weise, wie die extrahierten Datensätze ausgegeben werden, und der *DataPublisher* schreibt die Daten in das Zielverzeichnis [Lin15c]. Dieser Prozess wird durch die Abbildung 5.6 verdeutlicht.

Ein Job hat optional ein oder mehrere *Converter*, die die extrahierten Datensätze umwandeln, sowie einen oder mehrere *QualityChecker*, der die Qualität der extrahierten Datensätze prüft und feststellen soll, ob diese bestimmten Richtlinien entsprechen [Lin15c].

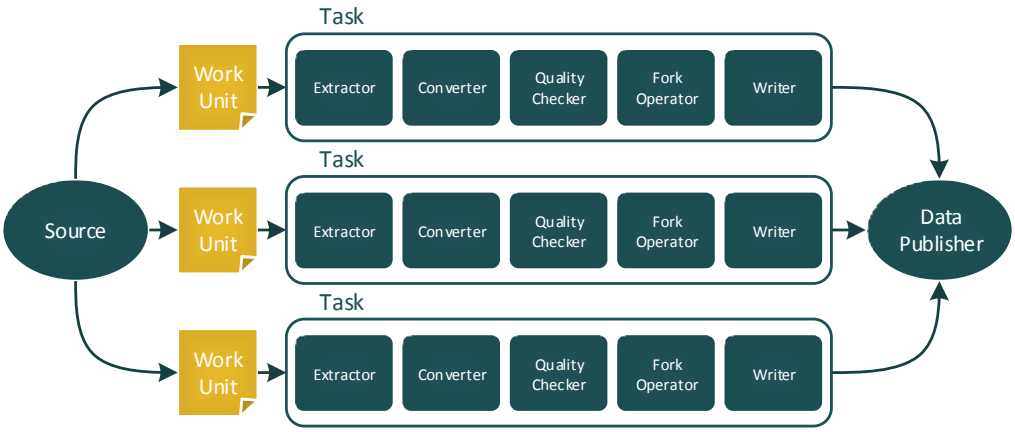


Abbildung 5.6.: Gobblin Constructs, eigene Darstellung in Anlehnung an [Lin15e]

Installation und Konfiguration

Nachfolgend wird beschrieben, wie Gobblin im MapReduce-Modus unter Hadoop 2.6.0 installiert und konfiguriert wird.

Bash 5.35: Checkout Gobblin Repository

```
1    $ git clone https://github.com/linkedin/gobblin.git
```

Bash 5.36: Build Gobblin: Gobblin wird mit Gradle gebuildet

```
1    $ cd gobblin
2    $ ./gradlew clean build
```

Zum Builden für Hadoop ab Version 2.X soll `-PuseHadoop2` hinzugefügt werden. Um Unit-Tests zu überspringen, wird zusätzlich `-x test` eingefügt.

Bash 5.37: Gobblin Distribution entpacken

```
1    $ tar -zxvf gobblin-dist-0.6.2.tar.gz
2    $ cd gobblin-dist
```

Vor dem Start eines Gobblin Jobs via Hadoop MapReduce soll der Referenzwert-Wert `fs.uri` in der Konfigurationsdatei unter `conf/gobblinmapreduce.properties` entsprechend der verwendeten Filesystem Uniform Resource Identifier (URI) definiert werden.

Config 5.38: File-System URI definieren

```
fs.uri=hdfs://<NameNode Host_Namen>:port/
```

Im Folgenden ist eine Zusammenfassung der Umgebungsvariablen, die unter Hadoop MapReduce für den Einsatz eingestellt werden können [Lin15d]:

- `HADOOP_BIN_DIR` Diese Variable definiert den Pfad zum Verzeichnis *bin* des Hadoop-Installationsverzeichnises.
- `GOBBLIN_WORK_DIR` Diese Variable definiert den Pfad zum *Gobblin Workdirectory*. Es muss im Dateisystem unter fs.uri erstellt werden.

Um einen Gobblin Job auf Hadoop MapReduce zu starten, muss das Startskript unter `bin/gobblin-mapreduce.sh` ausgeführt werden.

Zusätzlich muss zuvor eine Job-Konfigurationsdatei erstellt werden. Bei der Job-Konfigurationsdatei handelt es sich um eine Textdatei mit der Endung `.pull` oder `.job`. Hier werden Job-Eigenschaften definiert, die in einem Java-Properties-Objekt geladen werden. Gobblin verwendet Commons-Konfigurationsvariablen, um Variablen-Ersetzungen in Job-Konfigurationsdateien zu ermöglichen. Nachfolgend wird beispielhaft die, in der Test-Phase genutzte Job-Konfigurationsdatei, dargestellt:

Config 5.39: SftpDistcp.pull

```
job.name=SftpDistcp
job.group=Distcp
job.description=Job to copy data from sftp to hdfs

# Source properties
source.filebased.fs.uri=sftp://username@host_name:port
source.class=gobblin.data.management.copy.CloseableFsCopySource
source.conn.private.key=~/.ssh/id_rsa
source.conn.username=username
source.conn.password=password
source.conn.host=host_name
source.conn.port=port

# Dataset properties
```

```
15  gobblin.dataset.pattern=/path/to/source/data
16  # Publisher properties
17  #data.publisher.type=gobblin.data.management.copy.publisher.
        CopyDataPublisher
18  data.publisher.final.dir=/path/to/destination/on/HDFS
19
20  # Writer properties
21  writer.builder.class=gobblin.data.management.copy.writer.
        FileAwareInputStreamDataWriterBuilder
```

Bash 5.40: Start des Gobblin-Jobs im MR-Modus

```
1  $ gobblin-mapreduce.sh [OPTION] --conf <Job-Konfigurationsdatei>
```

Probleme

Während der Test-Phase sind eine Reihe an Fehlern und Bugs aufgetreten. Beispielsweise konnte die Jar-Datei `Gobblin-Metastore-0.6.2-133-g84c12a0.jar` nicht gefunden werden, obwohl diese in `/gobblin/gobblin-dist/lib/` existierte. Dieser Umstand ist dem frühen Entwicklungsstandes des Tools geschuldet.

Fazit

Gobblin ist in der verwendeten Version 0.6.2 zu instabil und unzuverlässig. Die Menge an Errors und Bugs und dessen Korrektion erschweren die Arbeit mit dem Tool und machen es für die Projektgruppe nicht praxistauglich. Deshalb wird Gobblin nicht weiter als potenzielles Tool in der Projektgruppe zu berücksichtigen.

5.1.9. Fluentd

Das Tool Fluentd wurde von Treasure Data Inc. als Open Source Lösung zur Vereinheitlichung der Verarbeitung und Nutzung von Log-Daten entwickelt [Tre16g]. Es ist in die drei Komponenten Input, optionale Filter und Output unterteilt. Über den Input lassen sich verschiedene Datenquellen wie Datenbanken, System- oder Programm-Logs definieren. Über den Output wird das Zielsystem, wie beispielsweise Datenbanken oder Programme gewählt. Neben dem reinen Transport der Daten bietet das Tool eine Reihe weiterer Funktionen an, die über eine Plug-In Struktur eingebunden werden können. Hierzu zählt die Möglichkeit, Daten zu filtern, also den Datenkontext zu untersuchen und inhaltsabhängig Aktionen zu

starten. Das Tool bietet außerdem eine Bufferfunktion um Datenverlust auszuschließen. Hierbei werden die Daten zwischengespeichert, bis eine Rückmeldung für eine fehlerfreie Übertragung eintrifft. So wird sichergestellt, dass die Daten korrekt in das Zielsystem übertragen wurden. Bei einem Übertragungsfehler werden die betroffenen Daten erneut gesendet. Zudem kann die Frequenz bei der Übertragung von Daten mit dem Buffer-Plug-In beeinflusst und gesteuert werden [Tre16a].

Über Parser-Plug-Ins ist eine Transformation der Input-Daten ausführbar. Diese kommen zum Einsatz, wenn die Struktur von keinem Input-Plug-In verarbeitet werden kann. Formatter-Plug-Ins auf der anderen Seite können die Datenstruktur vor dem Output entsprechend der Anforderungen der Nutzer anpassen, beispielsweise um eine Ausgabe im JSON-Format zu realisieren [Tre16b].

Die Input- und Output-Schnittstellen werden ebenfalls über Plug-Ins realisiert. Insgesamt sind über 500 Plug-Ins verfügbar. Diese wurden von Treasure Data oder der Fluentd-Community entwickelt und kostenlos bereitgestellt. [Tre16c]

Fluentd wird als ressourcensparend beworben, da es mit etwa 30-40 Megabyte (MB) Hauptspeicher auskommt und dabei bis zu 13.000 Events pro Sekunde pro Kern verarbeiten kann [Tre16g]. Um bei der Ausführung von Fluentd mehr als einen Rechenkern der CPU nutzen und somit die Verarbeitungsleistung maximieren zu können, muss ein Plug-In installiert werden, das ermöglicht, mehrere Input-Prozesse parallel zu starten [Tre16e].

Installation und Konfiguration

Das Tool kann unter Linux oder MacOS installiert werden. Die Entwicklung eines Windows-Clints ist noch nicht abgeschlossen. Es lässt sich zudem zwischen einer Basisinstallation der Fluentd-Engine und dem TD-Agent auswählen. Der Agent ist vorkonfiguriert und sendet alle Daten neben dem definierten Outputsystem standardmäßig an die Datencloud von Treasure Data [Tre16c]. Aus datenschutzrechtlichen Gründen ist der Einsatz des TD-Agents daher für die Projektgruppe ungeeignet. Zudem besteht die Zielsetzung der Projektgruppe darin, die Daten ausschließlich in das HDFS zu laden.

Fluentd kann auf verschiedene Arten installiert werden. Je nach Betriebssystem und ob die Basisversion oder der TD-Agent installiert werden soll, kann es mit RPM/DEB/dmg Package (TD-Agent), Ruby Gem oder als Sourcecode von GitHub geladen und anschließend mit *rake* gebuildet und als Gem in der Basisversion installiert werden. Nach der Installation von Fluentd können Input, Output und optionale Filter über das Config-File (unter etc/fluent/fluent.conf) initialisiert werden [Tre16d]. Ein einfaches Config-File für die Übertragung von Syslog-NG

Nachrichten auf eine Mongo Datenbank ist in der folgenden beispielhaften Struktur definiert:

Bash 5.41: Config-Beispiel

```
<source>
  @type syslog
  port 5140
  bind 0.0.0.0
  tag system.local
</source>

<match **>
  @type mongo
  database <db_name> #notwendig
  collection <collection_name> #optional, default: "untagged"
  host <hostname> #optional, default: "localhost"
  port <port> #optional, default: 27017
</match>
```

Über *<source>* werden beispielsweise alle Input-Plug-Ins definiert. Das *@type* definiert, welches Input-Plug-In eingesetzt werden soll. Darauf folgen die spezifischen Konfigurationsparameter. Es lassen sich je Config-File auch mehrere Plug-Ins für Input, Output und Filterfunktionen wählen.

Fazit

Zum aktuellen Zeitpunkt existiert kein Plug-In für den Transport von XML-Dateien [Tre16f]. Fluentd ist ausgerichtet auf die Verarbeitung von Log-Dateien oder Log-Streams. Da Fluentd somit die in Abschnitt 5.1.1 gestellten Anforderungen nicht erfüllt, wurde auf eine tiefer gehende Untersuchung der Konfiguration und Ausführung des Tools verzichtet.

5.1.10. Ergebnis

In diesem Abschnitt wurden diverse Tools zum Transport von Daten in das HDFS untersucht. Dabei wurde ihre Nutzbarkeit im Rahmen der Projektgruppe evaluiert. Die Anforderungen in Abschnitt 5.1.1 bildeten zwei verschiedene Bereiche. Es wurden Tools gesucht, die XML-Dateien über FTP von einem Server beziehen und in das HDFS laden können. Zudem wurden Tools zur Anbindung von relationalen Datenbanken an das HDFS untersucht.

Im Bereich der Anbindung relationaler Datenbanken an das HDFS erwies sich

Apache Sqoop als geeignet. Der Einsatz von Sqoop im Rahmen der Projektgruppe wird in Abschnitt 6.4.1 näher erläutert.

Die Suche nach Software für den Transport von XML-Dateien in das HDFS lieferte kein Ergebnis. Apache Storm ist durch seine Fokussierung auf Datenströmen für den Anwendungsfall der Projektgruppe nicht geeignet. Auch Apache Flume ist nicht geeignet, da die Dateien von Flume zeilenweise eingelesen und verarbeitet werden und dabei die Struktur der originalen Dateien verloren geht und somit ungültiges XML produziert wird. Im Rahmen der Evaluation konnte auch Logstash als mögliche Lösung ausgeschlossen werden, da nach mehreren Tests keine Verbindung zum HDFS hergestellt werden konnte. Apache Kafka wurde ebenfalls von der Projektgruppe ausgeschlossen, da es keine zufriedenstellende Unterstützung für das Arbeiten mit XML Dateien bietet. Dadurch das sich Gobblin in einer sehr frühen Entwicklungsphase zum Zeitpunkt der Evaluation befand, traten diverse Fehler auf die das Arbeiten mit Gobblin erschwert haben und somit wurde das Tool aufgrund von Instabilitäten ebenfalls verworfen. Ähnlich wie Apache Kafka, fehlt auch Fluentd die Funktionalität mit XML Dateien umzugehen weswegen auch dieses Tool nicht im Rahmen der Projektgruppe eingesetzt werden kann.

Lediglich mit einem eigens entwickelten Bash Skript war es möglich, die Log Dateien per FTP vom Server zu holen und in das HDFS zu übertragen. Keines der bestehenden Tools am Markt konnte die in Abschnitt 5.1.1 gestellten Anforderungen erfüllen. Aufgrund der beschränkten Möglichkeiten der Wartbarkeit, der Erweiterbarkeit und der Konfigurationsmöglichkeiten von Bash Skripten hat sich die Projektgruppe entschieden diese Möglichkeit jedoch ebenfalls auszuschließen.

Dennoch zeigte sich der Ansatz einer Eigenentwicklung als vielversprechend, so dass die Entwicklung eines eigenen Tools in einer höheren Programmiersprache angestrebt wurde. Die Implementierung und der Einsatz der Eigenentwicklung wird in Abschnitt 6.4.2 detailliert beschrieben.

5.2. Data Transformation

Im Rahmen der Projektgruppe wird die Extraktion und Transformation der Informationen aus semistrukturierten Dateien in eine Zielstruktur als Data Transformation definiert. Im Anwendungsfall der Projektgruppe werden Informationen aus semistrukturierten XML-Dateien extrahiert und so transformiert, dass sie in einen relationalen Datenbanklayer importiert werden können. Der Inhalt der XML-Dateien wird in Abschnitt 2.3 erläutert. Das Zielschema wird in Abschnitt 6.5.1 vorgestellt.

Das Ziel der Datentransformation im Rahmen der Projektgruppe ist es, die

XML-Daten in ein relationales Format zu übersetzen und im Server-Cluster der Apache Hadoop Installation zu speichern. In diesem Zusammenhang werden in Abschnitt 5.2.1 Anforderungen erhoben. Im Hadoop Ecosystem existieren bereits Komponenten, mit denen die Transformation durchgeführt werden könnte. In den folgenden Abschnitten werden die Tools Apache Hive (Abschnitt 5.2.2) und Apache Pig (Abschnitt 5.2.3) evaluiert, um zu entscheiden, ob eines der Tools die Anforderungen der Projektgruppe erfüllen kann. Anschließend werden mit Apache Spark (Abschnitt 5.2.4) und Hadoop Streaming (Abschnitt 5.2.5) zwei Ausführungsmöglichkeiten näher analysiert. Der Abschnitt 5.2.6 fasst das Ergebnis der Evaluation zusammen.

5.2.1. Anforderungen

Um eine Auswahl von geeigneten Data Transformation Tools aus dem Hadoop Ecosystem treffen zu können, wurden Anforderungen an den gesamten Aufgabenbereich gestellt, welche die Softwarelösungen erfüllen sollen. Diese Anforderungen sind in Tabelle 5.5 aufgelistet.

Kürzel	Beschreibung
TG2DT-01	Die semi-strukturierten XML-Dateien sollen in eine der Analyse zugänglichen Struktur transformiert werden können.
TG2DT-02	Die Daten sollen für das Laden angepasst und vorbereitet werden können.
TG2DT-03	Fehlerhafte oder fehlende Werte, Redundanzen, veraltete Werte sollen beseitigt werden können.
TG2DT-04	Erst nach erfolgreichem Abschluss der Datentransformation soll das Laden der transformierten Daten durchgeführt werden.
TG2DT-05	Alle Daten sollen in ein einheitliches Datenformat überführt werden können.
TG2DT-06	Jedes XML-Tag und jedes Attribut soll transformiert werden können.
TG2DT-07	Das XML muss nach der Transformation reproduzierbar sein.
TG2DT-08	Der Transformationsvorgang soll geloggt werden können.
TG2DT-09	Ein fehlerhafter Transformationsvorgang soll abgefangen und wiederholt werden können.
TG2DT-10	Aufkommende Datenspitzen sollen skalierbar sein.

Tabelle 5.5.: Anforderungen an Data Transformation Tools

5.2.2. Apache Hive

Apache Hive ist eine Komponente aus dem Apache Hadoop Ecosystem. Es wird als Data Warehouse Software bezeichnet. Mittels Apache Hive lassen sich große und verteilte Datenmengen abfragen und bearbeiten. Hierzu wird eine SQL ähnliche Sprache namens HiveQL verwendet. HiveQL kann über MapReduce, Tez oder Apache Spark ausgeführt werden. [Apa14a], [Apa16f], [Apa15a]

Apache Hive unterstützt folgende Aktionen bei der Bearbeitung von Daten:

- Extrahieren der Daten aus einem Quellsystem
- Transformieren der Daten
- Bereinigen der Daten
- Laden der Daten

Insgesamt führen diese Aktionen dazu, dass sich die Daten in eine vorgegebene Struktur bringen lassen. Apache Hive kann sowohl auf Daten zugreifen, die im HDFS liegen, als auch auf Daten aus anderen Datenspeichersystemen wie zum Beispiel HBase.

Analyse Apache Hive

Hive bietet mit *XPath* eine native Möglichkeit, XML-Daten zu parsen. In diesem Abschnitt wird untersucht, ob mit Hive eine Datentransformation im Rahmen der Anforderungen der Projektgruppe möglich ist.

Damit XPath auf die Informationen einer XML-Datei zugreifen kann, wird die gesamte Datei in eine Tabellenzelle geladen. Hierfür muss zuerst manuell der Zeilenumbruch (dargestellt durch den String '/n') aus der XML-Datei entfernt werden. Anschließend wird die bearbeitete XML-Datei manuell in Apache Hadoop und in eine Hive-Tabelle (xmltest) geladen, wobei die Hive-Tabelle das Attribut xml vom Typ String hat. Danach können die folgenden exemplarischen Queries ausgeführt werden. Diese HiveQL-Queries zeigen, wie mittels Apache Hive XML-Dateien geparst werden können.

Zum Ermitteln der UUID und aller leave label wird dieses Query verwendet:

SQL 5.42: Parsen der UUID und leave-label

```
1  select xpath(xml,"//*[local-name()='tracker']/@uuid") uuid,
2  leave_label
3  from xmltest
4  Lateral view explode(xpath(xml,"//*[local-name()='leave']/@label
       ")) test as leave_label;
```

Ein weiteres Beispiel ist das folgende Query. Hier wird die UUID und ein String Array mit allen angegeben Attributen ermittelt. Dieses Vorgehen könnte zum Parsen der Daten verwendet werden.

SQL 5.43: Parsen weiterer Attribute

```
1   select xpath(xml,"//*[local-name()='tracker']/@uuid") uuid,
2   xpath(xml,"//*[local-name()='enter']/@label") label,
3   xpath(xml,"//*[local-name()='enter']/@script") script,
4   xpath(xml,"//*[local-name()='enter']/@step") step,
5   xpath(xml,"//*[local-name()='enter']/@when") zeitpunkt,
6   xpath(xml,"//*[local-name()='enter']/@nest") nest,
7   xpath(xml,"//*[local-name()='enter']/@class") class,
8   xpath(xml,"//*[local-name()='enter']/@fullscreen") fullscreen,
9   xpath(xml,"//*[local-name()='enter']/@header") header,
10  xpath(xml,"//*[local-name()='enter']/@widgets") widgets,
11  xpath(xml,"//*[local-name()='enter']/@ProductSelected")
12  ProductSelected,
13  xpath(xml,"//*[local-name()='enter']/@ActualProduct")
14  ActualProducted,
15  xpath(xml,"//*[local-name()='enter']/@view") view_
16  from xmltest;
```

Ergebnisse Apache Hive

Mit Apache Hive ist es nicht oder nur sehr schwer möglich, eine schleifenartige Iteration über die XML-Datei durchzuführen. Diese wird jedoch für das Path-Element der XML-Dateien zwingend benötigt. In diesem Element können nicht zuerst alle Unterelemente des einen Typs (z.B. `<enter>`) und anschließend alle Unterelemente des anderen Typs (z.B. `<leave>`) ermittelt werden. Für die gestellten Anforderungen ist es zwingend erforderlich, dass die Reihenfolge der Unterelemente innerhalb des Path-Elements beibehalten beziehungsweise rekonstruiert werden kann.

Die hier manuell getätigte Vorarbeit zum Entfernen des Zeilenumbruchs könnte in Zukunft automatisiert werden.

Generell ist Apache Hive zum Parsen von XML-Dateien geeignet. Da die XML-Dateien im Anwendungsfall der Projektgruppe jedoch aufgrund der Verschachtelung und einer benötigten Einhaltung der Reihenfolge sehr komplex sind, ist Apache Hive keine sinnvolle Lösung für dieses Anwendungsszenario. Es ist entschieden worden, dass Apache Hive nicht zum Transformieren der Daten verwendet wird.

5.2.3. Apache Pig

Mit Apache Pig wurde ein weiteres potentielles Tool zur Data Transformation aus dem Hadoop Ecosystem untersucht. In diesem Abschnitt wird zunächst Apache Pig vorgestellt. Danach wird auf die Analyse des möglichen Einsatzes von Apache Pig eingegangen. Abschließend folgt eine kurze Zusammenfassung der Untersuchungsergebnisse von Apache Pig.

Apache Pig dient zum Analysieren von großen Datenmengen. Hierbei wird die Skriptsprache Pig Latin verwendet. Pig Skripte werden automatisch in MapReduce Jobs umgewandelt und können parallel auf dem HDFS ausgeführt werden. [Apa16e]

Analyse Apache Pig

Es gibt zwei Möglichkeiten eine XML-Datei mit Apache Pig zu parsen. In beiden Fällen wird die API Piggybank verwendet. Dabei kann auf zwei Funktionen zugegriffen werden: *XPath* und *Extract_All*. In diesem Abschnitt wird die Funktion XPath untersucht.

Zur Evaluierung der Funktion *XPath*, wird die Hortonworks Sandbox 2.3.2 als VM verwendet. Für Testzwecke wurden XML-Dateien im HDFS bereitgestellt. Das Parsen der Dateien erfolgt über ein Skript, dessen Aufbau im Folgenden beschrieben wird.

Zunächst wird die benötigte Parser-Funktion definiert, damit diese verwendet werden kann.

Bash 5.44: Definieren der Parser-Funktion

```
1   DEFINE XPath org.apache.pig.piggybank.evaluation.xml.XPath();
```

Danach wird das Haupt-Tag einem Typ zugewiesen. Ein Haupt-Tag ist beispielsweise das `<Path>`-Element, da es mehrere Unter-Tags enthält. Vor der Typisierung wird die XML-Datei noch mit dem LOAD-Befehl aus dem HDFS geladen.

Bash 5.45: Laden und Typisieren

```
1   A = LOAD 'Test.xml' using org.apache.pig.piggybank.storage.
        XMLLoader('TAG') as (x:chararray);
```

Anschließend wird der Inhalt des jeweiligen Unter-Tags für jedes Tag-Element generiert.

Bash 5.46: Inhalte generieren

```
1   B = FOREACH A GENERATE XPath(x, 'TAG/Untertag1'), XPath(x, 'TAG/
        Untertag2');
```

Am Ende werden die Ergebnisse ausgegeben.

Bash 5.47: Ergebnisse ausgeben

```
1   dump B;
```

Zudem besteht die Möglichkeit auf die geparsten Elemente ein Schema anzuwenden, um weitere Transformationen vorzunehmen und den gesuchten Inhalt auszugeben. Mit dem *STORE*-Befehl lässt sich das Ergebnis in dem jeweiligen Format in das HDFS ablegen.

Ergebnisse Apache Pig

Die Vorteile von Apache Pig gegenüber Apache Hive sind das prozedurale Vorgehen, die gute Aufteilung der Schritte in den drei Phasen LOAD, Transformation und DUMP sowie die Möglichkeit zur Ausgabe der Ergebnisse. Deswegen wird Apache Pig häufig auch im Zusammenhang mit Apache Hive verwendet. Dabei bereitet Apache Pig die Daten vor, damit anschließend einfache Analyseverfahren mit Apache Hive durchgeführt werden können.

Es gibt zwei Möglichkeiten eine XML-Datei mit Apache Pig zu parsen. In beiden Fällen wird die API Piggybank verwendet. Beide Varianten funktionieren. Die Funktion *XPath* ist aber einfacher zu realisieren. Dennoch trat bei der Analyse ein Problem auf. Mit Apache Pig ist es nicht möglich, Tags nach Attributen zu filtern. Aufgrund der Beschaffenheit der vorliegenden XML-Dateien wird diese Funktion jedoch unbedingt benötigt, um die Informationen vollständig parsen zu können. Aufgrund dieser Tatsache wird Apache Pig im Rahmen der Projektgruppe nicht weiter betrachtet.

5.2.4. Apache Spark

In diesem Abschnitt wird Apache Spark vorerst kurz vorgestellt. Anschließend wird auf die Analyse des möglichen Einsatzes von Apache Spark im Rahmen der Projektgruppe eingegangen. Darauf folgt eine kurze Zusammenfassung der Untersuchungsergebnisse von Apache Spark.

Apache Spark ist ein Framework für Cluster Computing. Spark kann sowohl Standalone als auch mit YARN oder Mesos betrieben werden. Es besteht aus den Komponenten Spark Core, Spark SQL, Spark Streaming, Machine Learning Library (MLlib) und GraphX die teilweise voneinander abhängigen sind. Die Komponenten und deren Abhängigkeiten sind in Abbildung 5.7 dargestellt. Der Unterschied zu Apache Hadoop besteht vor allem darin, dass Apache Spark eine In-Memory Technologie ist. Apache Spark kann auf das HDFS oder HBase als Datenquelle zugreifen. Spark Anwendungen können durch die Nutzung verschiedener APIs in Java, Scala oder Python implementiert werden. [Apa15a]

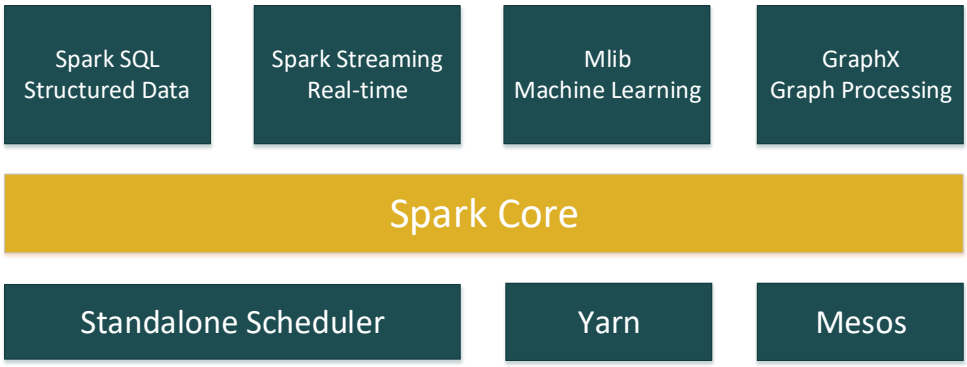


Abbildung 5.7.: Spark-Komponenten eigene Darstellung in Anlehnung an [KKWZ15]

Laut einer Umfrage aus dem Jahr 2015 sind die häufigsten Gründe für den Einsatz von Apache Spark die gute Performance, das leichte Programmieren, die einfache Implementierung sowie die Vielzahl der Analyse-Methoden [Dat15]. Die Nutzung der verwendeten APIs von Apache Spark verteilt sich wie folgt:

- Java hat einen geringen Anteil von 31%. Die Benutzung von Java bei Apache Spark ist eher historisch bedingt und geht weiter zurück.
- Scala macht mit 71% den größten Teil aus. Es ist rein objektorientiert und kann mit Java integriert werden.
- Python hat mit 58% den zweitgrößten Anteil. [Dat15]

Da Scala und Python von mehr Programmierern verwendet werden als Java, gibt es hier eine größere Community, sodass bei Problemen schneller Lösungen gefunden werden können. Dies war auch einer der Hauptgründe, dass sich die Projektgruppe für Scala und Python entschieden hat. Zudem gelten Scala und Python als leicht zu erlernende Programmiersprachen.

Architektur

Grundsätzlich ist Spark ein Framework, das die Verarbeitung großer Datenmengen bis in den Petabyte Bereich ermöglicht. Die wichtigsten Schlüsselfaktoren des Frameworks sind die Performance und die Skalierbarkeit. Spark kann sehr komplexe Berechnungen in kurzer Zeit über großen Daten bewältigen. [SK15]

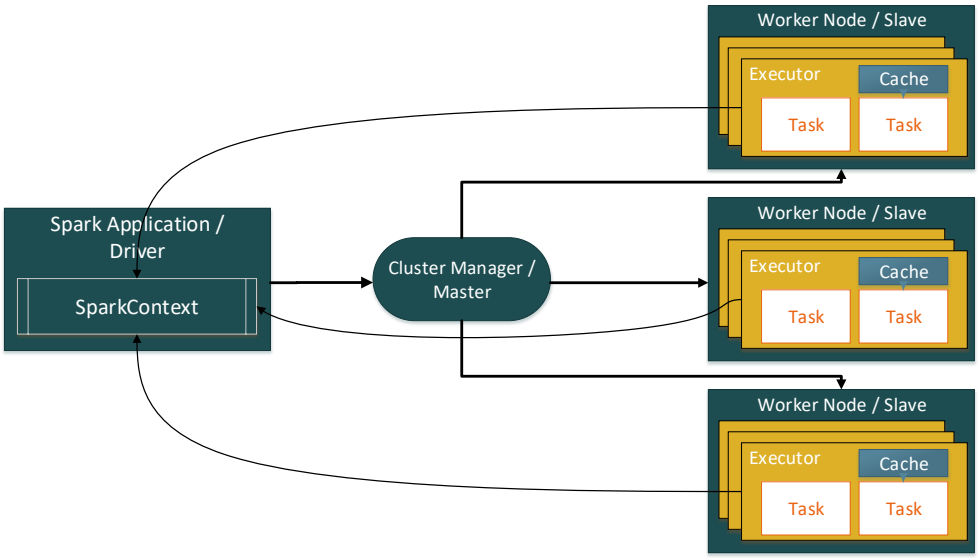


Abbildung 5.8.: Spark-Archetiktur - eigene Darstellung in Anlehnung an [KKWZ15]

Die Abbildung 5.8 zeigt die Architektur von Spark. Spark verwendet eine Master/Worker Architektur. Der Driver kommuniziert mit einem einzigen Koordinator (auch Master genannt), der die Ausführung der Worker während der Laufzeit des Programmes verwaltet. Jede Spark Anwendung besteht aus folgenden Bestandteilen [KKWZ15]:

Driver Ein Spark-Driver ist ein JVM Prozess, der den SparkContext der Spark-Anwendung enthält. Der Spark-Driver teilt die Spark-Anwendung in Tasks auf und verteilt sie für die Ausführung an die Executor. Ein Driver koordiniert die Worker und die Durchführung der Tasks.
Zwei wesentliche Konfigurationsmöglichkeiten für einen Spark Driver ist die zu verwendende Größe des Arbeitsspeichers und die zu verwendende Anzahl der CPU-Cores. Konfigurationen für den Driver können entweder über den aufrufenden Shell-Befehl als Parameter übergeben oder im SparkContext als Parameter gesetzt werden.

Master Der Spark Master, oder auch Cluster Manager, ist eine laufende Spark-Instanz, die sich zum Cluster verbindet, um die erforderlichen Ressourcen der Spark Anwendung zu verwalten.

Worker Ein Worker (auch bekannt als Slave) ist ein Server des Clusters, der je nach Ressourcenverfügbarkeit eine Anzahl von Executorn ausführt, die wiederum die Tasks des Programmes ausführen. Jeder Worker hat einen Block Manager, der den Workern ermöglicht, untereinander zu kommunizieren.

Executor Executor sind die Prozesse der Worker, die die Tasks des Spark Jobs ausführen. Executor senden aktive Task-Metriken zum Driver und informieren den Driver über den Status der Tasks. Executor speichern die Daten im Arbeitsspeicher während der Laufzeit des Programmes. Die wichtigsten Konfigurationen für einen Executor sind:

- spark.executor.cores: die Anzahl der Kerne für einen Executor.
- spark.executor.instances: die Anzahl der zu verwendenden Executor.
- spark.executor.memory: die Menge des Speichers pro Executor.

Resilient Distributed Dataset (RDD)

Ein Resilient Distributed Dataset (RDD) ist eine Sammlung von Elementen, die unveränderlich, parallelisierbar und fehlertolerant ist. RDDs können im Cluster verteilt und somit parallel bearbeitet werden. Die Bearbeitung findet durch sogenannte Transformationen und Aktionen statt. [Pen15]

Es gibt zwei Möglichkeiten, ein RDD zu erstellen [RLOW15]:

1. Daten im Treiberprogramm parallelisieren

Python 5.48: pySpark Daten Parallelizierung Bsp.

```
1   Data = [1, 2, 3, 4, 5]
2   distData = sc.parallelize(data,3)
```

2. Daten aus einer Datei laden:

Python 5.49: pySpark Daten von einer Datei laden

```
1   distFile = sc.textFile( "data.txt" ,3)
```

Innerhalb eines RDDs werden die Daten in sogenannte Partitions gespeichert. Die Anzahl der Partitions hat einen entscheidenden Einfluss auf die Parallelisie-

rung der Verarbeitung auf den Daten des RDDs. Spark erzeugt einen Task je Partition. Empfehlenswert sind laut der Spark Dokumentation zwei bis vier Partitions pro Rechenkern im Cluster. Spark bietet Funktionalitäten, mit denen die Anzahl der Partitions verändert werden kann. Ein Beispiel ist oben zu sehen. Beim Erzeugen eines RDD kann im parallelize-Befehl (oder textFile-Befehl) als zweiter Übergabeparameter die Anzahl der Partitions mitgegeben werden. [Apa16k]

DataFrames

Ein DataFrame ist eine Datenabstraktion wie ein RDD. Die Datenelemente eines DataFrames sind in Spalten strukturiert. Die Spalten werden mit einem Spaltennamen versehen. Ein DataFrame ist ähnlich wie eine relationale Datenbanktabelle strukturiert. Dabei können DataFrames aus verschiedenartig strukturierten Datenquellen konstruiert werden, was den entscheidenden Unterschied zu einer relationalen Datenbanktabelle darstellt. RDDs und DataFrames können von einem zum anderen Typ konvertiert werden. Ein DataFrame ermöglicht es, Structured Query Language (SQL) ähnliche Operationen auf dem Datenkonstrukt auszuführen. [Apa15b]

Python 5.50: DataFrame Beispiel

```
1   val df = Seq(("one", 1), ("one", 1), ("two", 1))
2   .toDF("word", "count")
3
4   df.show
5   +----+-----+
6   |word|count|
7   +----+-----+
8   | one|    1|
9   | one|    1|
10  | two|    1|
11  +----+-----+
12
13  val counted = df.groupBy('word').count
14
15  counted.show
16  +----+-----+
17  |word|count|
18  +----+-----+
19  | two|    1|
20  | one|    2|
21  +----+-----+
```

Das Beispiel zeigt, wie ein DataFrame mit Spark erstellt werden kann. Mit zwei Programmierzeilen kann eine Tabelle mit zwei Spalten *Word* und *Count* erstellt werden. Anschließend werden die Worte gruppiert und gezählt, um zu analysieren, wie häufig ein Wort vorkommt.

Programmiermodell Spark

Die wesentlichen Konzepte und Begriffe des Programmiermodells von Spark werden in diesem Abschnitt kurz erläutert. Zum Abschluss ist ein Word Count Beispiel mit Spark beschrieben. Für detaillierte Informationen über Apache Spark sei an dieser Stelle auf die Seminararbeit „Integration and analytical capabilities of Apache Spark“ im Anhang verwiesen.

Transformationen Transforamtionen sind RDD-Operationen, die ein neues RDD als Ergebnis zurückgeben (besipielsweise durch Filtern der Datenelemente). Transformationen werden nicht direkt ausgeführt, sondern nur, wenn eine Aktion ausgeführt wird. Dieses Konzept wird in Spark *Lazy-Evaluation* genannt [KKWZ15, S. 26-27]. Eine Tabelle mit den RDD-Transformationsfunktionen und deren Beschreibungen ist in der Seminararbeit „Integration and analytical capabilities of Apache Spark“ zu finden.

Aktionen Aktionen führen Berechnungen oder Aggregationen auf RDDs aus und liefern die Ergebnisse zurück. Die Aktionen werden verwendet, um etwas in den Datensätzen zu ändern [Pen15, S. 14-16]. Eine Tabelle, die einige RDD Aktionen und deren Beschreibungen zeigt, ist in der Seminararbeit „Integration and analytical capabilities of Apache Spark“ zu finden.

Lazy Evaluation *Lazy-Evaluation* ist ein Konzept in Spark. Es bedeutet, dass es ohne Aktionen keine Programmausführung gibt. Transformationen sind Anweisungen für ein Programm, die nicht vor der ersten Action ausgeführt werden. Spark verwendet die *Lazy-Evaluation*, um den Datenaustausch zu reduzieren, indem alle Datenoperationen gruppiert und auf einmal ausgeführt werden. [KKWZ15, S. 30]

Fault Tolerance Ein fehlertolerantes Programm ist ein Programm, das trotz eines Fehlers während der Programmlaufzeit das Programm erfolgreich beendet. Spark unterstützt Fehlertoleranz durch Speicherung der Metadaten über die Verarbeitungsschritte des RDD *lineage*. Fällt nun ein Teil des Programmes aus, können die betroffenen RDDs anhand der Metadaten reproduziert werden. [Bra14]

Persisting & Caching Persisting und Caching sind die Fähigkeiten, Daten (Ergebnisse aus den Berechnungen auf Daten eines RDD) im Hauptspeicher oder auf der Festplatte zwischenzuspeichern. Beim Persisting kann das Por-

gramm die Daten nur auf der Festplatte, nur im Hauptspeicher oder in einem hybriden Modus (Hauptspeicher und Festplatte) zwischenspeichern. Beim Caching werden die Daten im Hauptspeicher gehalten. Caching und Persisting können in Spark durch die Funktionen *cache()* und *persist()* festgelegt werden. [Apa16k]

Python 5.51: Spark Cache() Bsp.

```
1    rddFromTextFile.cache()
2    rddFromTextFile.persist("level")
```

Die folgende Tabelle zeigt auszugsweise Level für die *persist()* Funktion:

Storage Level	**Bedetung**
MEMORY_ONLY	Die RDD-Partitionen werden nur im Hauptspeicher gespeichert. Wenn im Hauptspeicher kein Platz mehr ist, werden einige Partitionen nicht gespeichert und bei erneutem Bedarf on-the-fly berechnet. MEMORY_ONLY ist die default level.
MEMORY_AND_DISK	Die RDD-Partitionen im Hauptspeicher speichern. Wenn im Hauptspeicher kein Platz mehr ist, werden die Partitionen auf die Festplatte ausgelagert.
DISK_ONLY	Die RDD-Partitionen nur auf der Festplatte zwischenspeichern.

Tabelle 5.6.: Spark Storage Leveln [Apa16k]

Word Count Beispiel

Der folgende Code-Block zeigt beispielhaft ein Spark Programm, das die Häufigkeit von Worten in einem Text zählt.

Python 5.52: pySpark WordCount Bsp.

```
1    text_file = sc.textFile("hdfs://...")
2    counts = text_file.flatMap(lambda line: line.split(" ")) \
3    .map(lambda word: (word, 1)) \
4    .reduceByKey(lambda a, b: a + b)
5    counts.saveAsTextFile("hdfs://...")
```

Zunächst wird in Zeile 1 exemplarisch dargestellt, wie eine Datei mit Textinhalt geladen wird. In Zeile 2 wird der geladene Text anhand eines Leerzeichens in einzelne Wortbestandteile des Textes gesplittet (hier mit der Funktion flatMap). Mit den Lambda Befehlen können Funktionen und ganze Code-Blöcke zu den Daten gebracht werden. In Zeile 3 wird jedes gesplittete Wort aus dem Text mit einer 1 (als Anzahl) versehen. In Zeile 4 werden alle gleichen Worte zusammengeführt über die reduceByKey Funktion und die Anzahlen zusammengerechnet. Sodass am Ende in Zeile 5 die Häufigkeiten für jedes Wort in einer Datei gespeichert werden.

Spark Jobs und Stages

Ein Spark-Job ist eine parallele Berechnung, der mehrere Tasks enthält und in Folge einer Aktion erstellt wird (zum Beispiel *saveAsTextFile()*, *collect()*). Jeder Job wird in Stages aufgeteilt, die wiederum Tasks enthalten.

Der Spark Driver liest das ganze Programm und teilt das Programm je nach Inhalt des Skriptes in Stages und Tasks. Die Tasks werden dann auf die Executor verteilt und parallel auf den verteilten Daten ausgeführt. [KKWZ15]

Ergebnisse Apache Spark

Apache Spark ist eine In-Memory Lösung im Hadoop Ecosystem, die ein großes Funktionsspektrum mit sich bringt. Durch die Kernfunktionalitäten und die weiteren Komponenten können mit Spark Daten transformiert und analysiert werden. Spark eignet sich für den Anwendungsfall XML-Dateien zu parsen. Wie im Word-Count Beispiel gezeigt wurde, können Funktionalitäten, die die Daten transformieren sollen, mit Hilfe von Lambda Expressions zu den Daten gebracht werden. Mit dieser Funktionalität können auch Funktionalitäten zu XML-Daten gebracht werden, die eben diese parsen. Dabei arbeitet Spark ähnlich wie Hadoop parallel und verteilt auf einem Cluster, mit dem großen Unterschied, dass Spark eine In-Memory Lösung ist.

5.2.5. Hadoop Streaming

Hadoop Streaming ist eine standardmäßige Funktionalität von Apache Hadoop. Mit Hadoop Streaming können Map/Reduce Jobs mit beliebigen Skripten (beispielsweise Shell- oder Python-Skripte) als Mapper und Reducer ausgeführt werden. [Apa16h] Der folgende Codeblock zeigt ein Beispiel für den grundlegenden Aufruf von Hadoop Streaming.

Bash 5.53: Hadoop Streaming - Standard Example

```
hadoop jar hadoop-streaming-2.7.2.jar
  -input myInputDirs
  -output myOutputDir
  -mapper /bin/cat
  -reducer /usr/bin/wc
```

Hadoop Streaming verarbeitet die Dateien standardmäßig Zeile für Zeile. Mapper und Reducer kommunizieren über die Standardeingabe (stdin) und Standardausgabe (stdout). Die Mapper erhalten die im HDFS abgelegten Chunks der Dateien, um diese Zeile für Zeile zu verarbeiten. Jeder Reducer schreibt die Daten, die an stdout übergeben werden, in eine Datei. Die in den Map/Reduce Jobs zu verarbeitenden Key-Value Paare werden standardmäßig per Tab-Delimiter in jeder Zeile getrennt. Wenn kein Tab gefunden werden kann, ist der Value null. [Apa16h]

Der Hadoop Streaming Befehl muss mit vier Pflicht-Parametern ausgeführt werden. Neben den Pflicht-Parametern existieren weitere optionale Parameter mit denen die Ausführung von Hadoop Streaming konfiguriert werden kann. Einen Auszug der möglichen Parameter mit der Beschreibung ist in Tabelle 5.7 zu sehen.

Parameter	**Pflicht /Optional**	**Beschreibung**
input	Pflicht	Pfadangabe zu den zu verarbeitenden Dateien
output	Pflicht	Pfadangabe für den generierten Output der Reducer
mapper	Pflicht	Auszuführendes Mapper-Skript
reducer	Pflicht	Auszuführendes Reducer-Skript
mapred.reduce.tasks	Optional	Anzahl der Reducer
mapred.map.tasks	Optional	Anzahl der Mapper
inputreader	Optional	Angabe einer inputreader Klasse, wie die Datensätze an die Mapper übergeben werden.
file	Optional	Dateien die auf die ausführenden Server mitgegeben werden sollen

Tabelle 5.7.: Auszug der Parameter für Hadoop Streaming [Apa16h]

Für das Ziel der Projektgruppe die XML-Dateien auf dem Hadoop Servercluster zu parsen, kann Hadoop Streaming eingesetzt werden. Die Parser-

Funktionalitäten können in der Map-Phase auf den XML-Dateien ausgeführt werden. Hierzu ist es erforderlich, dass die XML-Dateien vollständig - keine Chunks der XML-Dateien - an die Mapper übergeben werden. Um die XML-Dateien vollständig an die Mapper zu übergeben, wird der StreamXmlRecordReader als inputreader benutzt. Dem StreamXmlRecordReader wird ein Start- und End-Tag übergeben. Der Inhalt zwischen den beiden Tags wird an den Mapper übergeben, der Rest wird ignoriert.

Über die Nutzung des Key-Value Prinzips in der Map- und Reduce Phase können die geparsten Daten „gesteuert“ an die Reducer übergeben werden. Dabei kann im Reducer-Skript implementiert werden, in welcher aufbereiteten Form die Daten in die Datei geschrieben werden soll (beispielsweise im CSV-Format). Der Folgende Codeblock zeigt ein Hadoop Streaming Beispiel, indem aus den XML-Dateien Inhalte exemplarisch geparst und in Ausgabedateien geschrieben werden.

Bash 5.54: Hadoop Streaming - XML Example

```
1  hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-
       streaming.jar \
2    -D mapred.reduce.tasks=12 \
3    -D mapred.map.tasks=1 \
4      -inputreader "StreamXmlRecordReader,begin=<digifoto,end=</
           digifoto>" \
5    -file mapper3.py \
6    -file reducer2.py
7      -input /data/xml/inputs/*.xml \
8      -output /data/xml/outputs/01/ \
9      -mapper ./mapper3.py \
10     -reducer ./reducer2.py \
```

5.2.6. Ergebnis

Bei der Analyse der Data Transformation Tools hat sich herausgestellt, dass Apache Hive und Pig die Anforderungen nicht zufriedenstellend erfüllen. Apache Hive kann die XML-Dateien aufgrund der komplexen Verschachtelung und einer benötigten Einhaltung der Reihenfolge nicht anforderungsgemäß parsen. Bei Apache Pig tritt das gleiche Problem auf. Hier wird die API Piggybank verwendet, um die XML-Dateien zu parsen. Aber es ist nicht möglich, Tags mit Attributen zu filtern. Nur durch das Anpassen der API Piggybank wäre es möglich Attribute zu filtern. Diese Anpassung ist zu zeitintensiv und die Komplexität der XML-Dateien würde dann immer noch ein Problem darstellen. Aufgrund dieser Evaluation wird

auf Apache Hive und Pig für die Daten Transformation verzichtet.

Allerdings konnten Hadoop Streaming und Apache Spark positive Evaluationsergebnisse erzielen. Ein ganz wichtiger Punkt ist es, dass bei Hadoop Streaming und Apache Spark die Programmiersprachen Java, Scala und Python eingesetzt werden können. Mit den vorhanden APIs können komplexe XML-Dateien geparst und die Einhaltung der Reihenfolge beibehalten beziehungsweise reproduziert werden. Ein weiterer wichtiger Aspekt ist, dass Hadoop Streaming, sowie Spark parallel und verteilt auf einem Cluster arbeiten können. Die gestellten Anforderungen können somit erfüllt werden.

Wie bereits angesprochen, bieten sowohl Hadoop Streaming, als auch Spark die Möglichkeit mit verschiedenen APIs Programme zu entwickeln, die über eine der beiden Technologien verteilt auf den Daten ausgeführt werden können. Die Aufgabe der Projektgruppe ist es nun weiter ein solches Parser-Programm zu entwickeln, um die XML-Dateien parsen zu können. Die Entwicklung des Parsers wird in Kapitel 6.5 beschreiben.

5.3. Data Analytics

Die erste Aufgabe der Projektgruppe war es mit Hilfe des Hadoop Ecosystems die Informationen aus den XML-Dateien zu extrahieren. Mit Hilfe des Hadoop Ecosystems soll ein Überblick über die zur Verfügung stehenden Informationen erstellt werden und erstes Wissen oder erste Erkenntnisse aus den Informationen generiert werden. Für dieses analytische Vorhaben werden Analytic-Tools benötigt. Der Bereich Analytics lässt sich in folgende Bereiche aufteilen (Auszug aus der Seminararbeit „Analytische Kapazitäten des Apache Hadoop Ecosystem“):

Descriptive Analytics Die klassische Aufgabe der BI ist die descriptive (zu deutsch: deskriptive oder auch beschreibende) Analytics. Aufgabenbereich ist die systematische Auswertung von Vergangenheitsdaten und Bereitstellung eines ad-hoc Berichtswesens. Die Auswertungen können mit Online Analytical Processing (OLAP), statistischen Verfahren, Data Mining und Text Mining durchgeführt werden. Beispielsweise könnten die Absatz- und Produktzahlen (der Vergangenheit) pro Zeiteinheit pro Vertriebseinheit in einer Produkthierarchie dargestellt werden und Kunden mittels Data Mining Verfahren in Kundencluster eingeteilt werden. [Hum14]

Predictive Analytics Mit Hilfe von Data Mining Verfahren kann predictive (zu deutsch: prädikative oder auch vorhersagende) Analytics durchgeführt werden. Es wird versucht mit dem Vergleich von historischen und aktuellen Daten Prognosen zu erstellen. Dies können beispielsweise Prognosen zu Kauf-

wahrscheinlichkeiten, Kündigungswahrscheinlichkeiten oder Absatz- und Umsatzprognosen sein. [Hum14]

Prescriptive Analytics Prescriptive (zu deutsch: präskriptive) Analytics geht einen Schritt weiter. Es werden nicht nur Muster gesucht und Informationen oder Wissen generiert, sondern je nach Anwendungsfall bestimmte Handlungen vorgeschlagen. Ziel ist die Automatisierung von Entscheidungsprozessen. Um dies zu ermöglichen, sollten Geschäftsregeln und Modelle der möglichen Handlungsaktionen mit den Analysen verbunden werden. [Hum14]

Die Analytics Bereiche nehmen in ihrer Komplexität absteigend zu. In Abhängigkeit wird je höher die Komplexität auch der Nutzen der Analytics Bereiche größer. Aus den Erfahrungen des Projektpartners CEWE sollten die Analytics Bereiche sequentiell in ihrer absteigenden Reihenfolge durchlaufen werden, um den größtmöglichen Nutzen zu generieren. Zudem sollte die Projektgruppe mit Hilfe der analytischen Tools Datenverständnis über die zur Verfügung stehenden Informationen erlangen (Maximum-Werte, Minimum-Werte, Häufigkeiten, NULL-Werte etc.).

5.3.1. Anforderungen

Die Anforderungen sollen bei der Auswahl eines geeigneten Analytic Tools aus dem Hadoop Ecosystem helfen. Die Anforderungen sind in Tabelle 5.8 zu finden.

Kürzel	Beschreibung
TG3A-01	Die Analysen sollen benutzerfreundlich durchgeführt werden können (beispielsweise über ein UI).
TG3A-02	Deskriptive und explorative Datenanalyse soll möglich sein.
TG3A-03	Data-Mining, möglichst viele Algorithmen, sollen unterstützt werden.
TG3A-04	Die Software sollte Bestandteil des Hadoop Ecosystem sein, um die Hadoop Vorteile (Verteilung etc.) zu nutzen.

Tabelle 5.8.: Anforderungen an Analytic Tools

5.3.2. Apache Hive

Apache Hive ist bereits in Kapitel 5.2.2 beschrieben worden. Der Fokus lag bei der Beschreibung jedoch auf der Fähigkeit XML zu parsen mit Hilfe von Hive. Apache Hive bietet ein SQL Interface, um Daten aus dem HDFS abfragen zu können. Dabei

bietet HiveQL einen Funktionsumfang nahe dem SQL-92 Standard. ([Apa14a] und [Apa14b])

5.3.3. Apache Spark MLlib

Apache Spark ist grundlegend in Kapitel 5.2.4 beschrieben worden. Für die Auswahl von analytischen Tools aus dem Hadoop Ecosystem kommt die MLlib Komponente von Apache Spark in Frage. Es stehen die ganzen Funktionalitäten von Spark zur Verfügung, die in Kapitel 5.2.4 beschrieben worden sind. Zusätzlich wurde sich im Rahmen der Seminararbeit „Analytische Kapazitäten des Apache Hadoop Ecosystem“ mit der Spark MLlib beschäftigt. Zusammenfassend bietet die Spark MLlib eine Reihe von Machine Learning Algorithmen für die Aufgabenbereiche Clustering, Klassifikation, Prognose und Assoziationsanalyse. Beispielsweise sind Algorithmen wie FP-Growth, Logistische Regression, Naive Bayes, Entscheidungsbäume und K-Means implementiert. Zusätzlich bietet die MLlib auch statistische Funktionen, um die Daten zu beschreiben und diese explorativ zu untersuchen. [Apa15c] und [Apa15d]

5.3.4. Ergebnis

Nachdem die Anforderungen aufgestellt worden sind, mussten nicht sämtliche Tools aus dem Hadoop Ecosystem näher evaluiert werden. In der Seminararbeit „Analytische Kapazitäten des Apache Hadoop Ecosystem“ wurden bereits einige analytische Tools aus dem Hadoop Ecosystem im Hinblick auf ihre analytischen Fähigkeiten evaluiert. Sodass anhand der Anforderungen und der Verfügbarkeit im Hadoop Ecosystem eine schnelle Wahl auf Apache Hive und Apache Spark mit der MLlib gefallen ist. Aus diesem Grund wurden diese beiden Tools kurz vorgestellt und auch hier für weitere Details auf die Seminararbeit verwiesen. Apache Hive wird für die Datenerforschung und den Bereich deskriptive Analytics verwendet. Apache Spark bietet die geforderten Data Mining Funktionalitäten und zusätzliche Funktionalitäten für die Datenerforschung und deskriptive Analyse.

5.4. Data Visualization

Aufgabe der Data Visualization ist es, die Ergebnisse des Bereiches Data Analytics beispielsweise mit Hilfe von Diagrammen zu präsentieren. Durch intuitive und vereinfachende Darstellungen der Analysen können Zusammenhänge, Tendenzen oder Ausprägungen vom Nutzer schneller erfasst, interpretiert und letztlich in Erkenntnisse und Wissen umgewandelt werden.

Im folgenden Abschnitt 5.4.1 werden Anforderungen an die Visualisierung der Ergebnisse erstellt. Anschließend wird in Abschnitt 5.4.2 das Tool Zeppelin aus dem Hadoop Ecosystem vorgestellt. Am Ende dieses Abschnitts werden die Ergebnisse in Abschnitt 5.4.3 zusammengefasst.

5.4.1. Anforderungen

Im Rahmen der Projektgruppe sollen zwei verschiedene Visualisierungskomponenten konzeptioniert und erstellt werden:

Management Dashboard Das Management Dashboard zielt auf die Anwendergruppen Management, Geschäftsführung oder Vorstand ab. Es soll einen schnellen und einfach gehaltenen Überblick geben. Dabei soll die Oberfläche sehr übersichtlich gestaltet werden und einen rein lesenden Zugriff erlauben.

Technisches Cockpit Das TC bietet für Fachanwender und Fachverantwortliche aus den Fachbereichen die Möglichkeit umfangreiche Abfragen optisch darzustellen.

In Absprache mit CEWE hat die Projektgruppe Anforderungen an die Komponenten definiert. Die Anforderungen dienen sowohl zur Orientierung bei der Implementierung, als auch zur Bewertung der Zielerreichung des Teilbereiches der Projektgruppe. Hauptfokus der Projektgruppe liegt auf den Analysen und dem hierzu erforderlichen ELT-Prozessen. Dashboard und TC haben eine niedrigere Priorität. In den folgenden Tabellen 5.9, 5.10 und 5.11 werden die generellen Anforderungen an das Dashboard, an das TC, sowie weitere „nice-to-have“ Anforderungen aufgelistet.

Kürzel	Beschreibung
TG3D-01	Das Dashboard soll von fachfremden Nutzern bedienbar sein.
TG3D-02	Das Dashboard soll nicht durch unqualifizierte Handlungen zerstörbar sein.
TG3D-03	Das Dashboard soll Kartenausgaben unterstützen.
TG3D-04	Das Dashboard soll eine Auflistung der darstellbaren Diagramme besitzen.
TG3D-05	Das Dashboard soll die Visualisierungen zeitgesteuert ermitteln und ausgeben können (Scheduling).
TG3D-06	Das Dashboard soll temporäre Änderungsraten von Werten ausgeben können.
TG3D-07	Das Dashboard soll Daten aus unterschiedlichen Datenquellen nutzen und anzeigen können (Hive, Spark, Shell, etc.).
TG3D-08	Das Dashboard soll Unterseiten (Notebooks) haben können.

TG3D-09	Die Diagramme sollen durch eine Drill-Down Funktion detailliertere Informationen darstellen.
TG3D-10	Das Dashboard soll übersichtlich und intuitiv bedienbar sein.
TG3D-11	Die Diagramme sollen durch Variablen, wie beispielsweise ein Zeitintervall, skalierbar sein.

Tabelle 5.9.: Anforderungen an das Dashboard

Kürzel	Beschreibung
TG3TC-01	Das TC soll einfache Analysen, die nicht in den Bereich Data Mining fallen, ermöglichen.
TG3TC-02	Das TC soll Abfragen in HiveQL, Python, Spark und Shell ausführen können.
TG3TC-03	Das TC soll Ergebnisse von Abfragen visualisieren können.
TG3TC-04	Das TC soll alle nutzbaren Datenbanken anzeigen können.
TG3TC-05	Das TC soll alle Tabellen einer Datenbank anzeigen können.
TG3TC-06	Das TC soll alle Attribute einer Tabelle anzeigen können.
TG3TC-07	Das TC soll Abfragen durch Drag-and-Drop vereinfachen.
TG3TC-08	Das TC soll Vorlagen für Standardabfragen anbieten.

Tabelle 5.10.: Anforderungen an das technische Cockpit

Kürzel	Beschreibung
TG3N-01	Ein Monitoring-Fenster soll relevante Logs anzeigen.
TG3N-02	Es sollen Nachrichten des Alert-Systems angezeigt werden.
TG3N-03	Es soll Exportmöglichkeiten in andere Systeme/Web-Templates geben.
TG3N-04	Es soll gefilterte Views für Handelspartner geben.
TG3N-05	Es soll ein Rechtesystem für verschiedene Nutzer geben.

Tabelle 5.11.: „Nice-to-have“-Anforderungen des Dashboards

5.4.2. Apache Zeppelin

Das Tool Apache Zeppelin ist eine Komponente des Hadoop-Ecosystem, das für Visualisierungen und das Erstellen von Dashboards genutzt werden kann. In diesem Abschnitt wird Zeppelin vorgestellt. Dabei wird auf die Installation und Kon-

figuration eingegangen. Abschließend werden die Ergebnisse in einem Unterabschnitt zusammengefasst.

Das Projekt Apache Zeppelin wurde erstmals Ende 2014 vorgestellt [Rom15] und befindet sich seitdem im Inkubatorstatus und damit noch in der Entwicklungsphase. Es ist ein, unter der Apache Software License, Version 2.0 (ASL2) lizenziertes, interaktives Notizbuch mit einer modernen Weboberfläche [Fou16b]. Es ermöglicht einen komfortablen Zugriff auf unterschiedliche Datenbestände. Kollaborativ lassen sich verschiedene Sprachen wie Scala, SQL und viele mehr verwenden und sowohl deren Eingaben als auch deren Ergebnisse einfach dokumentieren [Fou16a]. Durch die Unterstützung vieler gleichzeitiger Benutzer und die Kombination mehrerer Interpreter ist es möglich, große Datensätze konzentriert durch unterschiedlichste Wissenschaftler analysieren und visualisieren zu lassen. Jeder Benutzer kann seine jeweilige Expertise in gemeinsame virtuelle Notizbücher einbringen und diese beliebig für andere freigeben.

Zeppelin integriert sich in das Hadoop Ecosystem, indem es durch seine Interpreter-APIs universelle Datenverarbeitungssysteme wie Apache Spark, Apache Flink, Apache Kylin und ähnliche unterstützt. Installieren und nutzen lässt es sich sowohl auf dem Server als auch auf dem Client unabhängig vom verwendeten Betriebssystem beziehungsweise dem verwendeten Browser, da es einerseits als Java-Webanwendung deployt wird und andererseits auf Webstandards setzt. Mittels verschiedener Diagrammtypen können analysierte Daten visualisiert werden. Die Form der Darstellung ist dabei sehr vielfältig. So existieren neben der Tabellenform zum Beispiel auch Torten-, Flächen-, Säulen-, Trend-, und Punktdiagramme. Außerdem lässt sich Zeppelin um weitere Diagrammformen und Interpreter ergänzen [Fou16c].

Installation und Konfiguration

Die Installation von Zeppelin auf einem Server kann manuell, oder über Ambari erfolgen.

Ambari-Installation Es gibt ein Ambari-Plugin für Apache Zeppelin, mit dem sich das Notizbuch nicht nur installieren, sondern auch konfigurieren und überwachen lässt. Dadurch wird diese Software genauso wie Hive oder andere Kerndienste darüber verwaltbar. Das Plugin ist Online verfügbar [2].

Manuelle Installation Der folgende Code-Block zeigt die notwendigen Schritte zur manuellen Installation von Zeppelin.

[2] `https://github.com/tzolov/zeppelin-ambari-plugin`

Bash 5.55: Schritte der manuellen Installation

```
# Verzeichnis erstellen und darin wechseln
mkdir /opt/zeppelin && cd /opt/zeppelin
# Zeppelin herunterladen
wget http://artfiles.org/apache.org/incubator/zeppelin
    /0.5.6-incubating/zeppelin-0.5.6-incubating-bin-all.
    tgz
# Archiv entpacken
tar xfvz zeppelin-0.5.6-incubating-bin-all.tgz
# Zeppelin starten
bin/zeppelin-daemon.sh start
# Weboberflaeche aufrufen
firefox http://localhost:8080/
```

5.4.3. Ergebnis

Nachfolgend werden die Ergebnisse der Datenvisualisierung dargestellt. Hierzu wird geprüft, ob die zuvor definierten Anforderungen an die Kernkomponenten Dashboard und TC mit Apache Zeppelin erfüllt werden konnten. Die Wahl für den Einsatz von Apache Zeppelin wurde bereits zu einem frühen Projektstatus getroffen. Ausschlaggebend waren hierbei die Ergebnisse der Seminararbeit „Visualisierungsmöglichkeiten mittels dem Apache Hadoop Ecosystem“ und ein kurzer Vergleich mit den Lösungen Kibana[3] und Apache Kylin[4], die hier nicht weiter betrachtet werden.

Für das Dashboard wurden zehn Anforderungen definiert. Durch den Einsatz von Apache Zeppelin und dessen Funktionsumfang konnten diese weitestgehend erfüllt werden. Die Zielvorgabe *TG3D-08*, die Existenz von Unterseiten, kann nicht direkt erfüllt werden, da zwar eine unlimitierte Anzahl an Notebooks erstellt werden kann, diese aber nicht in Beziehung zueinander gesetzt werden können. Anforderung *TG3D-09* beschreibt die Funktionalität, mittels Drill-Down, schrittweise detailliertere Informationen aus einem Diagramm entnehmen zu können. Dies kann ebenfalls nicht durch Zeppelin umgesetzt werden. Das TC, das nur Benutzern mit fachlichem Know-How zugänglich sein soll, um Abfragen und Analysen durchführen zu können, wurde mit neun Anforderungen beschrieben. Anforderung *TG3TC-07* definiert, dass Abfragen durch eine Drag- and Drop-Funktion vereinfacht durchgeführt werden sollen. Dies ist jedoch nur möglich, wenn nach der Ausführung einer Anfrage ein Diagramm erstellt wurde. Hierbei können die

[3] https://www.elastic.co/de/products/kibana
[4] http://kylin.apache.org

X- und Y-Werte des Diagramms angepasst werden. Ein Reservoir an Vorlagen (*TG3TC-08*) muss dem TC manuell beigefügt werden. Die Anforderung kann demnach erfüllt werden, wird jedoch nicht automatisch von Zeppelin gewährleistet.

Abschließend kann resümiert werden, dass das Tool Zeppelin für die Erreichung der Ziele im Bereich Data Visualisation geeignet ist.

5.5. Workflowmanagement

Der konzeptionierte ELT-Workflow besteht aus verschiedenen Aufgaben, die nacheinander ausgeführt werden sollen. Damit daraus kein manueller Prozess entsteht wird in diesem Abschnitt untersucht, ob es im Umfeld des Hadoop-Ecosystem Werkzeuge gibt, die einen Workflow automatisiert durchführen und überwachen können. In Abschnitt 5.5.1 werden diese und weitere Anforderungen an ein solches Tool erhoben. Im Anschluss werden in den Abschnitten 5.5.2 und 5.5.3 zwei Tools untersucht, die im Rahmen des Workflowmanagements die Anforderungen erfüllen könnten. Das Ergebnis der Betrachtung wird in Abschnitt 5.5.4 zusammengefasst.

5.5.1. Anforderungen

Im Folgenden werden die Anforderungen an Workflowmanagement Tools im Rahmen der Projektgruppe beschrieben. Die Tabelle 5.12 beschreibt die funktionalen Anforderungen. In Tabelle 5.13 sind die nicht-funktionalen Anforderungen aufgelistet.

Kürzel	Beschreibung
WFF-01	Einfache Abhängigkeit Management
WFF-02	Klarheit über den Status (bei Job fail/Success, Job Durchführung Dauer ...)
WFF-03	Scheduling
WFF-04	Logging
WFF-05	Einfache Zugriff auf Logs
WFF-06	Benachrichtigungen
WFF-07	Wiederholungen parametrisieren (bei Job fail...)

Tabelle 5.12.: Funktionale Anforderungen an ein Workflowmanagement-Tool

Kürzel	Beschreibung
WFNF-01	Skalierbarkeit
WFNF-02	Erweiterbarkeit
WFNF-03	Authentifikation und Autorisation

Tabelle 5.13.: Nicht-funktionale Anforderungen an ein Workflowmanagement-Tool

5.5.2. Airflow

Airflow ist ein Open Source Workflowmanagement Tool, das von Airbnb entwickelt wurde. Airflow ist eine Plattform, um Daten-Pipelines zu erstellen, zu überwachen und einzuplanen. Airflow wurde seit Mai 2016 in Apache Inkubator bereitgestellt. In diesem Abschnitt wird das Tool vorgestellt und der Einsatz in der Projektgruppe geprüft. [Apa16a]

Um eine Data Pipeline zu erstellen, kann ein Directed Acyclic Graph (DAG) definiert werden, der bestimmte konfigurierbare Tasks beinhalten kann. Ein konstruiertes Beispiel dient der Veranschaulichung: Ein einfacher DAG besteht aus drei Tasks A, B und C. Task A muss erfolgreich ausgeführt werden, bevor Task B laufen kann, aber Task C kann jederzeit ausgeführt werden. Im Beispiel kann es vorkommen, dass Task A nach 5 Minuten stoppt und Task B bis zu 5-mal neu gestartet werden kann, falls es fehlschlägt. Weiterhin wird der Workflow jeden Abend um 22.00 Uhr gestartet. [Apa16c]

Allgemein beschreibt ein DAG, wie ein Workflow durchgeführt werden soll. Seine Aufgabe ist es sicherzustellen, dass alles was getan werden soll zum richtigen Zeitpunkt in der richtigen Reihenfolge und mit dem richtigen Umgang mit unerwarteten Problemen geschieht. [Apa16c]

Die Definition von DAGs erfolgt in Python Dateien. Diese sind immer in einen *DAG_FOLDER* abgelegt. Airflow führt den Code in jeder Datei aus, um dynamisch ein DAG Objekt zu bauen. Es können beliebig viele DAGs mit einer beliebigen Anzahl an Tasks erstellt werden. Im Allgemeinen sollte jeder DAG einem einzigen logischen Arbeitsablauf entsprechen. [Apa16c]

Für den Aufbau von DAGs bietet Airflow weitere verschiedene Konzepte, wie beispielsweise Operatoren. Sie bestimmen was tatsächlich gemacht wird. Airflow liefert verschiedene Operatoren für viele gemeinsame Aufgaben [Apa16c]:

- *BashOperator*: führt ein Bash Aufruf aus.
- *PythonOperator*: ruft eine beliebige Python-Funktion auf.
- *HTTPOperator*: schickt ein HTTP Anfrage
- *SqlOperator*: führt ein SQL Anfrage aus

- *Sensor*: wartet auf eine bestimmte Zeit, Datei, Datenbank-Reihe, um eine vordefinierte Aktion durchzuführen.
- *HiveOperator*: dient als Schnittelle zwischen Airflow und Hive.

Zu einem anderen Konzept von Airflow gehören *Tasks*. Sie sind die Instanzierung eines Operators. Ein Task ruft ein Operator mit spezifischen Werte und Parametern auf. [Apa16c]

Um Tasks auszuführen und laufen zu lassen, bietet Airflow einen Mechanismus an, der *Executor* heißt. Airflow liefert drei Arten von Executorn [Apa16b]:

LocalExecutor Dieser Executor führt die Tasks lokal parallelisiert aus. Es nutzt die Python Multiprocessing-Bibliothek und Queues aus, um die Ausführung von Tasks zu parallelisieren.

SequentialExecutor Dieser Executor führt nur ein Task zu einem Zeitpunkt aus. Es ist auch der einzige Executor, der mit SQLite verwendet werden kann, da SQLite nicht mehrere Verbindungen unterstützt. Dieser Executor wird meinstens für das Testen und Debuggen benutzt.

CeleryExecutor Dieser Executor ist ein verteiltes System, um große Mengen an Nachrichten zu verarbeiten. Es bietet die Möglichkeit die Anzahl der Worker horizontal zu skalieren. Damit dieses funktioniert, muss eine Celery-Backend wie RabbitMQ oder Redis konfiguriert werden.

5.5.3. Azkaban

Azkaban ist eine Open Source Software, welche für die Planung und automatische Terminierung von Operationen auf einem Hadoop-Cluster genutzt werden kann. Entworfen wurde diese von dem Karrierenetzwerk *LinkedIn*. Automatische Operationen sollen durch eine Weboberfläche verwaltet werden können.

Generell besteht Azkaban aus drei Komponenten [Apa16g]:

Relationale Datenbank Mit Hilfe einer MySQL Datenbank werden Daten intern verwaltet. Hierzu gehören die folgenden Komponenten:

- Benutzerrollen und -Rechte
- Projekte
- Scheduler
- Historie vergangener Jobs
- Logs

Webserver Diese Komponente ist die Applikation für die Verwaltung der genannten Daten. Der Webserver bietet dem User eine grafische Nutzeroberfläche für die Steuerung aller Daten.

Executor Server Dieser Dienst ist für die Ausführung der Aufgaben verantwortlich. Basierend auf den Daten der MySQL-Datenbank werden Befehle automatisch ausgeführt.

Installation

Installiert wird Azkaban, indem vorerst das Paket mit allen benötigten Daten heruntergeladen wird. Anschließend muss manuell ein MySQL-Schema eingerichtet werden und die Zugangsdaten hierfür in den Konfigurationsdateien der Applikation eingetragen werden. Danach sind alle SQL-Dateien auf diesem Schema auszuführen, um Tabellen und Constraints anzulegen. In den Konfigurationsdateien können zudem weitere Einstellungen festgelegt werden wie beispielsweise der Port des Webservers.

Ergebnis

Das Ergebnis der Evaluation von Azkaban durch die Projektgruppe ist, dass diese Software in der Praxis einen höheren Implementierungsaufwand mit sich bringt, als durch die Dokumentation erwartet wurde. Auch nach mehrfacher Neuinstallation war der Webserver nicht erreichbar, obwohl alle Konfigurationen einwandfrei eingestellt wurden. Eine ausgeprägte Community, welche Lösungen für bekannte Probleme bereitstellt, gibt es leider nicht.

5.5.4. Ergebnis

In den vorigen Abschnitten wurden die Workflowmanagement-Tools Airflow und Azkaban vorgestellt. Während Airflow alle in Abschnitt 5.5.1 aufgestellten Anforderungen erfüllen konnte, erwies sich der praktische Einsatz von Azkaban als schwierig. Ohne ausgeprägte Community konnte das Tool nicht zum Einsatz gebracht werden. Somit kommt Airflow in der Projektgruppe zum Einsatz. Die Einführung von Airflow und die Konfiguration einzelner DAGs wird in Abschnitt 6.8 beschrieben.

6. Praktische Durchführung

Nach der Evaluation verfügbarer Software, welche als Lösungen für Prozessschritte in Frage kommen, wurde sich auf einige Tools festgelegt. Die so erhaltene Übersicht ergab zudem, dass an vielen Stellen Eigenentwicklungen benötigt wurden. In diesem Kapitel werden alle verwendeten Komponenten und Eigenentwicklungen aufgezeigt. Das Vorgehen wird hierbei in sechs Aufgabenabschnitte untergliedert: *Installation Hadoop & weiteren Komponenten, Datenimport, -Transformation, -Analyse, -Visualisierung* und die *Steuerung des Workflows*. Zu jeder Lösung werden hierbei die Konfigurationen, bzw. die kritischen Codeblöcke dargestellt und erläutert.

6.1. Installation Alpha-Version

Um Aussagekräftige Ergebnisse zu sammeln, wurde im Rahmen der Projektgruppe ein eigener Hadoop-Cluster aufgesetzt. Mit diesem wurden die zuvor evaluierten Tools und Eigenentwicklungen eingesetzt. In diesem Abschnitt wird beschrieben, wie der Hadoop-Cluster auf virtuellen Maschinen installiert und konfiguriert wurde. Die Abbildung 6.1 veranschaulicht, welche Dienste und Komponenten auf den Servern installiert wurden.

Innerhalb der gestrichelten Kästen werden Services, die auf den jeweiligen Servern installiert sind dargestellt. Des Weiteren sind zwei externe Datenbanken, welche auf dem Server-Master installiert sind, aufgeführt. Diese sind unter anderem für die Softwarekomponenten Flume- und Docker-Tests verwendet worden.

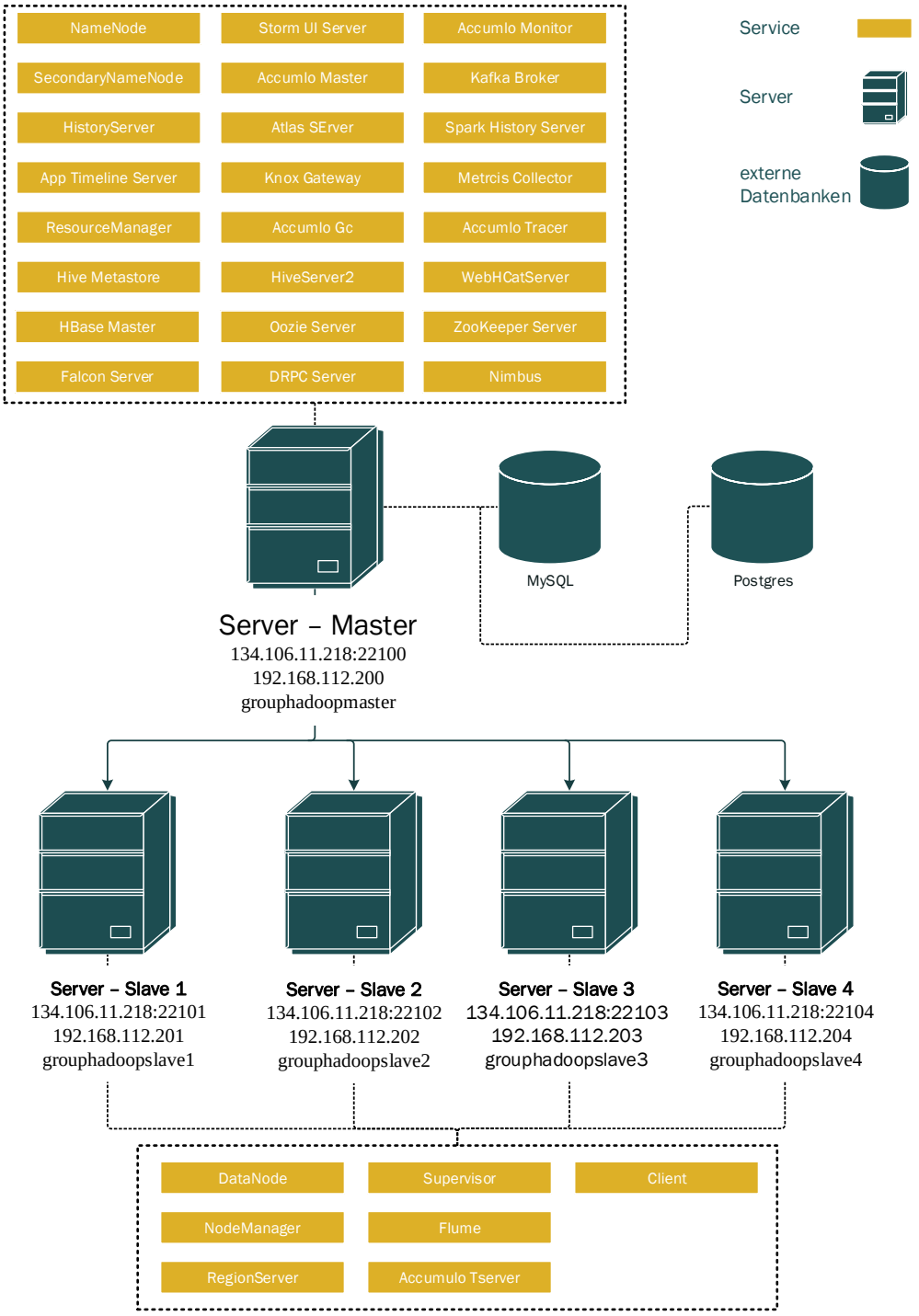


Abbildung 6.1.: Überblick Serverarchitektur Alpha-Cluster

6.1.1. Vorbereitungen zur Installation von Hadoop

Zunächst wurde Java auf dem Server *grouphadoopmaster* installiert. Für die Installation von Hadoop 2.6 muss Java in der Version 6 oder höher installiert werden.

Bash 6.1: Installation Java

```
sudo apt-get update
sudo apt-get install default-jdk
java -version
```

Hiernach sollte die aktuelle Version der Java-Installation ausgegeben werden.

Nach der Installation von Java, wurde auf allen fünf Servern ein User angelegt. Dieser User „hdusermaster“ wurde als User für alle Hadoop-Dienste verwendet.

Bash 6.2: Anlegen des Users hdusermaster

```
sudo addgroup hadoop
sudo adduser --ingroup hadoop hdusermaster
```

Für den Zugriff auf die einzelnen Servermaschinen wurde das SSH verwendet. Die Server des Hadoop-Clusters kommunizieren ebenfalls untereinander über das SSH-Protokoll. Aus diesem Grund musste auf jedem beteiligten Server SSH installiert werden. Für eine erfolgreiche Kommunikation muss auf jedem Server ein SSH-Key erzeugt bzw. hinterlegt wurde. Zunächst wurde auf dem Server *grouphadoopmaster* ein SSH-Key generiert.

Bash 6.3: SSH-Key generieren (ohne Passphrase)

```
ssh-keygen -t rsa -P ""
cat \$HOME/.ssh/id_rsa.pub >> \$HOME/.ssh/authorized_keys
```

Nach der Generierung, wurde dieser auf den vier Servermaschinen *grouphadoopslave1*, *grouphadoopslave2*, *grouphadoopslave3* und *grouphadoopslave4* in der Datei ‚authorized_keys‘ hinterlegt.

Bash 6.4: Hinterlegen des Public Key auf allen Slave-Servern

```
cat /home/hdusermaster/.ssh/id_rsa.pub | ssh 192.168.112.201 '
    cat >> .ssh/authorized_keys'
```

```
2   cat /home/hdusermaster/.ssh/id_rsa.pub | ssh 192.168.112.202 '
        cat >> .ssh/authorized_keys'
3   cat /home/hdusermaster/.ssh/id_rsa.pub | ssh 192.168.112.203 '
        cat >> .ssh/authorized_keys'
4   cat /home/hdusermaster/.ssh/id_rsa.pub | ssh 192.168.112.204 '
        cat >> .ssh/authorized_keys'
```

Auf den vier Slave-Servern (*grouphadoopslave1*, *grouphadoopslave2*, *grouphadoopslave3*, *grouphadoopslave4*) wurde ebenfalls ein SSH-Key erzeugt.

Bash 6.5: SSH-Key generieren (ohne Passphrase)

```
1   ssh-keygen -t rsa -P ""
2   cat \$HOME/.ssh/id_rsa.pub >> \$HOME/.ssh/authorized_keys
```

Anschließend wurde der jeweilige Public Key auf dem Master-Server *grouphadoopmaster* hinterlegt.

Bash 6.6: Hinterlegen des Public-Key auf dem Master-Server

```
1   cat /home/hdusermaster/.ssh/id_rsa.pub | ssh 192.168.112.200 '
        cat >> .ssh/authorized_keys'
```

Auf dem Master-Server sowie auf den Slave-Servern mussten außerdem die Berechtigungen auf den Unterordner ‚ssh' im Homeverzeichnis des Users „hdusermaster", sowie der Datei ‚authorized_keys' verändert werden.

Bash 6.7: Berechtigungen vom Ordner ‚ssh' verändern

```
1   chmod 700 ~/.ssh
2   chmod 600 ~/.ssh/authorized_keys
```

Um nicht bei jeder SSH-Verbindung jeweils die IP-Adresse verwenden zu müssen, wurde auf jedem Server in der ‚/etc/hosts'-Datei der jeweilige Hostname eingetragen.

Bash 6.8: Erweitern der */etc/hosts*-Datei auf jedem Server

```
1   192.168.112.200 grouphadoopmaster.local grouphadoopmaster
2   192.168.112.201 grouphadoopslave1.local grouphadoopslave1
3   192.168.112.202 grouphadoopslave2.local grouphadoopslave2
```

```
4   192.168.112.203 grouphadoopslave3.local grouphadoopslave3
5   192.168.112.204 grouphadoopslave4.local grouphadoopslave4
```

Ein Funktionstest vom Master-Server aus wurde abschließend ebenfalls ausgeführt. Dadurch wurde sichergestellt, dass eine SSH-Verbindung über den Hostnamen auf die jeweiligen Server-Maschinen möglich war.

Bash 6.9: SSH-Verbindung - Funktionstests

```
1   ssh grouphadoopslave1
2   exit
3   ssh grouphadoopslave2
4   exit
5   ssh grouphadoopslave3
6   exit
7   ssh grouphadoopslave4
8   exit
```

Nachdem jeder Server über SSH erreichbar war, war es für Installation von Hadoop Voraussetzung, dass auf jedem Server das Network Time Protocol (NTP) installiert war.

Bash 6.10: Installation NTP

```
1   sudo apt-get install ntp
2   ntpdate -s ntp.ubuntu.com
3   sudo service ntp start
```

Bevor nun mit der Installation von Hadoop begonnen werden konnte, musste noch sichergestellt werden, dass IPv6 auf allen Server-Maschinen deaktiviert war. Hierzu musste die Datei */etc/sysctl.conf* editiert werden.

Bash 6.11: IPv6 deaktivieren

```
1   sudo nano /etc/sysctl.conf
```

Hinzufügen von folgenden Zeilen:

Bash 6.12: IPv6 deaktivieren

```
1   net.ipv6.conf.all.disable\_ipv6 = 1
2   net.ipv6.conf.default.disable\_ipv6 = 1
3   net.ipv6.conf.lo.disable\_ipv6 = 1
4   sudo reboot
```

Um abschließend sicherzustellen, dass IPv6 deaktiviert wurde, wurde ein Befehl ausgeführt, der bei einer erfolgreichen Deaktivierung von IPv6 den Returncode ‚1' zurück gab.

Bash 6.13: Test, ob IPv6 deaktiviert

```
1   cat /proc/sys/net/ipv6/conf/all/disable\_ipv6
```

Alle Vorbereitungen für die Installation von Hadoop waren somit abgeschlossen, so dass mit der eigentlichen Installation von Hadoop begonnen werden konnte.

6.1.2. Installation von Hadoop

Die nachfolgenden Schritte zeigen die Installation von Hadoop 2.6. Diese wurden auf dem Server *grouphadoopmaster* ausgeführt. Die komplette Installation wurde mit Hilfe von ‚Ambari' vorgenommen. Im ersten Schritt wurde das ‚Ambari'-Repository in der Version 2.1.2 auf dem Server *grouphadoopmaster* in der *sources.list.d* Datei hingefügt, heruntergeladen und anschließend mit Hilfe von dem Befehl ‚apt-get' installiert. Die nachfolgenden Installationsschritte wurden alle unter dem User „hdusermaster" ausgeführt.

Bash 6.14: Download von Ambari 2.1.2

```
1   cd /etc/apt/sources.list.d
2   sudo wget http://public-repo-1.hortonworks.com/ambari/ubuntu12
        /2.x/updates/2.1.2/ambari.list
3   sudo apt-key adv --recv-keys --keyserver keyserver.ubuntu.com
        B9733A7A07513CAD
```

Nachdem ‚Ambari' heruntergeladen war, wurde die Installation gestartet. Dabei musste der für den Start/Stop von Ambari notwendige User „hduser-

master“, sowie die Oracle-JDK hinterlegt werden. Die erweiterten Datenbank-Konfigurationen waren hierbei nicht notwendig.

Bash 6.15: Installation von Ambari 2.1.2

```
sudo apt-get update
sudo apt-get install ambari-server
// 'hdusermaster' eingeben
Oracle JDK auswaehlen (1 = 1.7 auswaehlen)
Advanced Database Configuration (n waehlen)
sudo ambari-server start
```

Ambari war somit installiert und konnte über eine Weboberfläche konfiguriert werden. Die Weboberfläche war mit dem Browser auf dem Port 8080 erreichbar.

Auf der Ambari-Weboberfläche war es zunächst beim ersten Start erforderlich, dem User „admin“ ein Passwort zu hinterlegen. Als nächsten Schritt wurde auf ‚Launch install wizard‘ geklickt, welche durch eine Schritt-für-Schritt Installation führte. Hier musste zunächst ein Clustername vergeben werden. Hier wurde ‚pgdash_alpha‘ eingegeben. Im nächsten Schritt war es notwendig, den Stack auszuwählen. Hier wurde „HDP 2.3“ ausgewählt. Im darauffolgenden Installationsschritt ‚Install Option‘ wurde unter dem Menüpunkt ‚Advanced‘ lediglich „Ubuntu12“ ausgewählt, da auf den Servermaschinen „Ubuntu“ in der Version 12 vorinstalliert war. Im nächsten Menüpunkt „Confirm Hosts“ war es erforderlich, alle Fully Qualified Domain Names (FQDN) der fünf Server-Maschinen einzugeben.

Config 6.16: Eingabe der FQDN

```
grouphadoopmaster.local
grouphadoopslave1.local
grouphadoopslave2.local
grouphadoopslave3.local
grouphadoopslave4.local
```

Des Weiteren musste der Private SSH-Key des Users ‚hdusermaster‘ in dem Menüpunkt hinterlegt werden. Im darauffolgenden Installations-Menüpunkt mussten die einzelnen Komponenten ausgewählt, die auf dem Master-Server installiert werden sollten. Hier wurden keine Änderungen vorgenommen, sondern die von Ambari vorgeschlagenen Installationsziele ohne Änderungen übernommen. Im nachfolgenden Screenshot sind diese ersichtlich.

Im nächsten Installations-Menüpunkt mussten die Komponenten ausgewählt

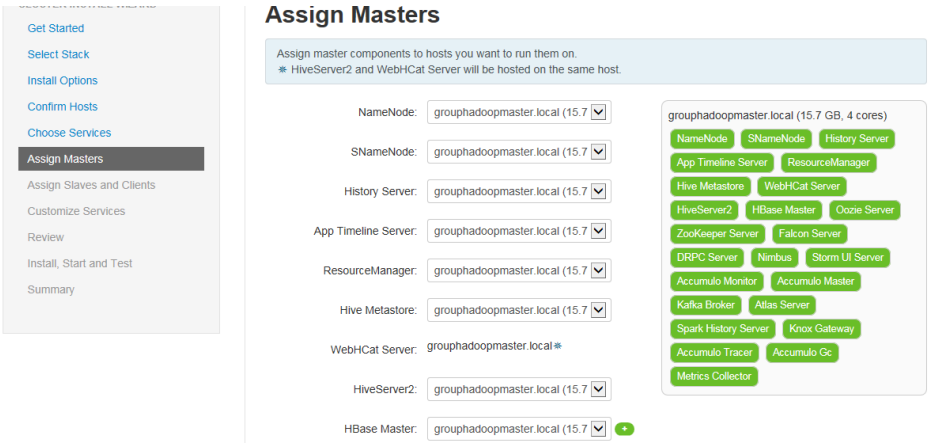


Abbildung 6.2.: Auswahl der Komponenten - Assign Master

werden, die auf den einzelnen Slave-Servern installiert werden sollten. Auch hier wurden keine Änderungen vorgenommen, sondern die von Ambari vorgeschlagenen Installationsziele übernommen. In den nachfolgenden zwei Screenshots sind diese ersichtlich.

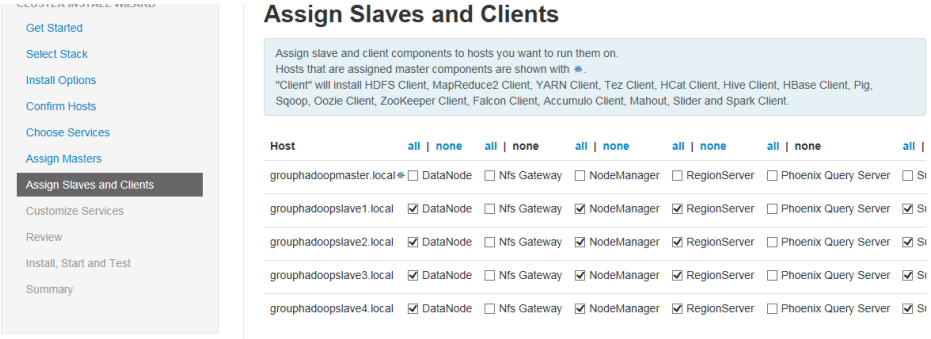


Abbildung 6.3.: Auswahl der Komponenten - Assign Slaves and Clients 1

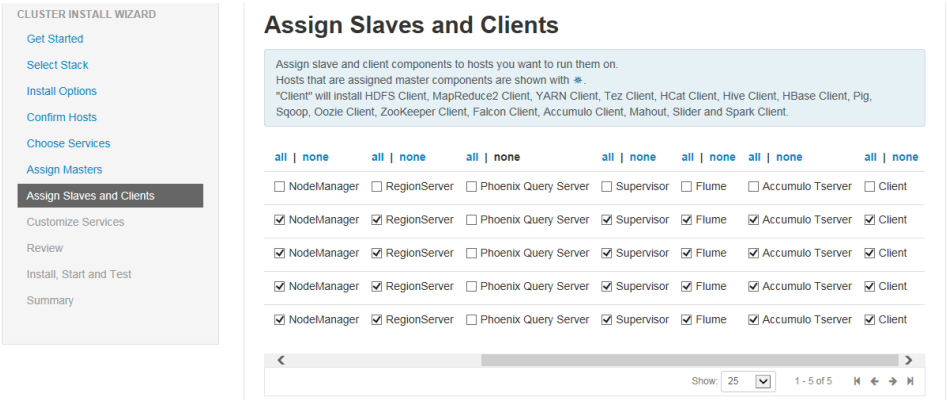


Abbildung 6.4.: Auswahl der Komponenten - Assign Slaves and Clients 2

Nach der Vergabe aller Passwörter und durch anschließendes klicken auf ‚Next', wurde in Form einer Zusammenfassung bzw. Übersicht aufgezeigt welche Komponenten genau installiert werden. Diese ist nachfolgend ersichtlich:

Admin Name : admin

Cluster Name : pgdash_alpha

Total Hosts : 5 (5 new)

Repositories :

- debian7 (HDP-2.3):
  http://public-repo-1.hortonworks.com/HDP/debian7/2.x/updates/2.3.4.0
- debian7 (HDP-UTILS-1.1.0.20):
  http://public-repo-1.hortonworks.com/HDP-UTILS-1.1.0.20/repos/debian6
- redhat6 (HDP-2.3):
  http://public-repo-1.hortonworks.com/HDP/centos6/2.x/updates/2.3.4.0
- redhat6 (HDP-UTILS-1.1.0.20):
  http://public-repo-1.hortonworks.com/HDP-UTILS-1.1.0.20/repos/centos6
- redhat7 (HDP-2.3):
  http://public-repo-1.hortonworks.com/HDP/centos7/2.x/updates/2.3.4.0
- redhat7 (HDP-UTILS-1.1.0.20):
  http://public-repo-1.hortonworks.com/HDP-UTILS-1.1.0.20/repos/centos7
- suse11 (HDP-2.3):
  http://public-repo-1.hortonworks.com/HDP/suse11sp3/2.x/updates/2.3.4.0
- suse11 (HDP-UTILS-1.1.0.20):
  http://public-repo-1.hortonworks.com/HDP-UTILS-1.1.0.20/repos/suse11sp3
- ubuntu12 (HDP-2.3):
  http://public-repo-1.hortonworks.com/HDP/ubuntu12/2.x/updates/2.3.4.0
- ubuntu12 (HDP-UTILS-1.1.0.20):
  http://public-repo-1.hortonworks.com/HDP-UTILS-1.1.0.20/repos/ubuntu12
- ubuntu14 (HDP-2.3):
  http://public-repo-1.hortonworks.com/HDP/ubuntu14/2.x/updates/2.3.4.0
- ubuntu14 (HDP-UTILS-1.1.0.20):
  http://public-repo-1.hortonworks.com/HDP-UTILS-1.1.0.20/repos/ubuntu12

Services:

- ***HDFS***
  - DataNode : 4 hosts
  - NameNode : grouphadoopmaster.local
  - Nfs Gateway : 0 host
  - SNameNode : grouphadoopmaster.local
- ***YARN + MapReduce2***
  - App Timeline Server : grouphadoopmaster.local
  - NodeManager : 4 hosts
  - ResourceManager : grouphadoopmaster.local
- ***Tez***
  - Clients : 4 hosts
- ***Hive***
  - Metastore : grouphadoopmaster.local
  - HiveServer2 : grouphadoopmaster.local
  - WebHCat Server : grouphadoopmaster.local
  - Database : MySQL (New MySQL Database)
- ***HBase***
  - Master : grouphadoopmaster.local
  - RegionServer : 4 hosts
  - Phoenix Query Server : 0 host
- ***Pig***
  - Clients : 4 hosts
- ***Sqoop***
  - Clients : 4 hosts
- ***Oozie***
  - Server : grouphadoopmaster.local
  - Database : MySQL (New Derby Database)
- ***ZooKeeper***
  - Server : grouphadoopmaster.local
- ***Falcon***
  - Server : grouphadoopmaster.local
- ***Storm***
  - DRPC Server : grouphadoopmaster.local
  - Nimbus : grouphadoopmaster.local
  - UI Server : grouphadoopmaster.local
  - Supervisor : 4 hosts
- ***Flume***
  - Flume : 4 hosts
- ***Accumulo***
  - Gc : grouphadoopmaster.local
  - Master : grouphadoopmaster.local
  - Monitor : grouphadoopmaster.local
  - Tracer : grouphadoopmaster.local
  - Tserver : 4 hosts

Abbildung 6.5.: Übersicht der zu installierenden Komponenten / Dienste

Der folgende Screenshot zeigt den laufenden Installationsprozess in Ambari. Mit einer Übersicht bzw. Installation Summary wurde die Installation von Ha-

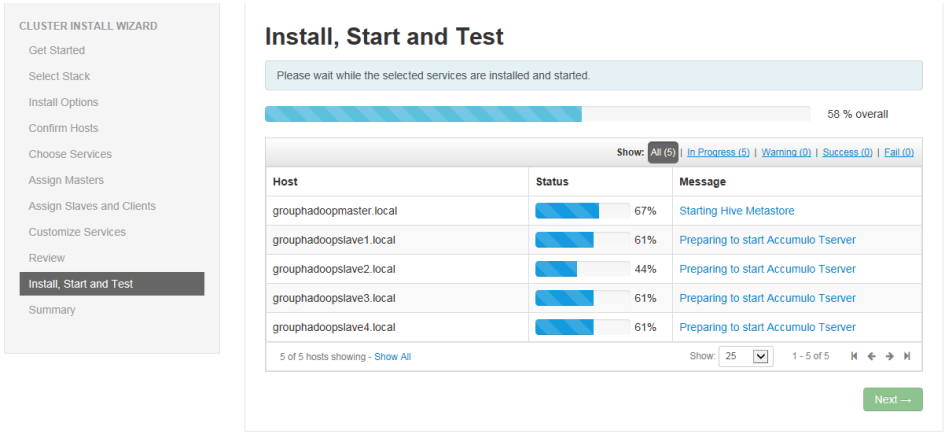


Abbildung 6.6.: Installation in Progess

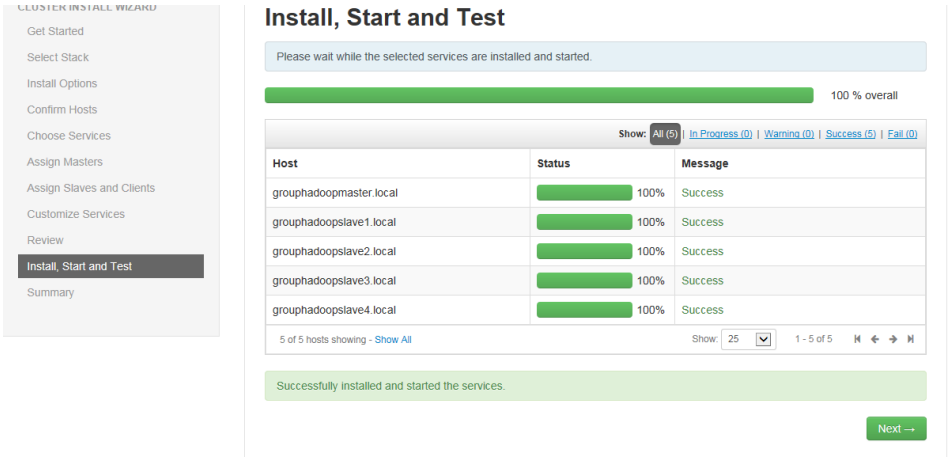


Abbildung 6.7.: Installation Success

doop 2.6 über Ambari erfolgreich abgeschlossen.

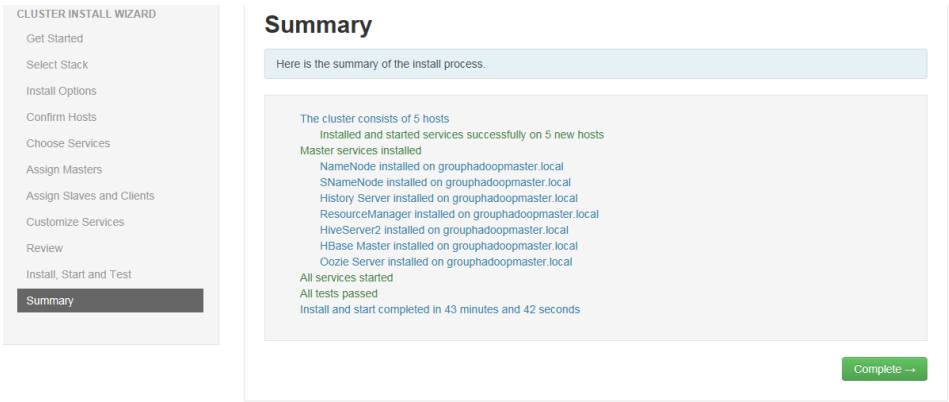


Abbildung 6.8.: Installation Summary

6.1.3. Konfiguration Ambari

Nachdem die Installation von Hadoop 2.6 erfolgreich abgeschlossen worden konnte, wurden auf dem Server *grouphadoopmaster* innerhalb von Ambari noch einige Einstellungen vorgenommen werden. So wurden innerhalb von Ambari noch zwei verschiedene ‚Views' angelegt: eine HDFS- und Hive-View. Eine View in Ambari ermöglichst es, innerhalb von Ambari auf verschiedene Dateien bzw. Komponenten zuzugreifen. So können bspw. über die ‚HDFS-View' die Dateien aus dem HDFS angezeigt werden.

Bevor die Views jedoch angelegt werden konnten, war es erforderlich, dass der verfügbare Arbeitsspeicher für den Ambari-Servers erhöht wird. Dies hat den Hintergrund, dass jede View eine eigene Speicherzuordnung erhält und somit, bei einer mehrfachen Verwendung der Views es zu einem Arbeitsspeicher-Problem kommen könnte. Für die Erhöhung des Arbeitsspeichers waren folgende Schritte notwendig:

Auf dem Server 'grouphadoopmaster' die Datei 'ambari-env.sh' öffnen:

Bash 6.17: Erhöhung Arbeitsspeicherzuordnung ‚Ambari'

```
1   sudo nano /var/lib/ambari-server/ambari-env.sh
```

Zur Variable ‚AMBARI_JVM_ARGS' springen und den Default-Wert von ‚-Xmx2048m' durch den Folgenden verändern:

Bash 6.18: Erhöhung Arbeitsspeicherzuordnung ‚Ambari'

```
1   -Xmx4096m -XX:PermSize=128m -XX:MaxPermSize=128m
```

Neustart des ‚Ambari'-Servers um die Änderungen abzuschließen.

Bash 6.19: Erhöhung Arbeitsspeicherzuordnung ‚Ambari'

```
1   sudo ambari-server restart
```

Nachdem die Arbeitsspeicherzuordnung von ‚Ambari' erhöht wurde, konnten die beiden View über die ‚Ambari'-Weboberfläche angelegt werden. Zur Anlage der HDFS-View waren folgende Schritte notwendig:

Zunächst mussten „Services → HDFS → Configs → Advanced → Custom core-site" zwei neue Propertys hinzugefügt werden:

Bash 6.20: Anlegen der „HDFS-View“ in ‚Ambari‘

```
1   hadoop.proxyuser.hdusermaster.groups = *
2   hadoop.proxyuser.hdusermaster.hosts = *
```

Anschließend konnte unter dem Menüpunkt „Admin → Manage Ambari → Views“ eine neue View erstellen. Hierfür wurde als Namen „HDFS View“ verwendet. Nachdem die Einstellungen gespeichert wurden, musste die HDFS-Komponenten neu gestartet werden. Anschließend waren die HDFS-Files-View sichtbar und nutzbar.

Zur Erstellung der Hive-View waren folgende Schritte notwendig.

In der Ambari-Weboberfläche unter „Services → HDFS → Configs → Advanced → Custom core-site“ drei weitere neue Propertys hinzufügen:

Bash 6.21: Anlegen der „Hive-View“ in ‚Ambari‘

```
1   hadoop.proxyuser.hive.groups = *
2   hadoop.proxyuser.root.groups= *
3   hadoop.proxyuser.root.hosts= *
```

Anschließend musste im HDFS ein neue Ordner sowie entsprechende Berechtigungen vergeben werden:

Bash 6.22: Anlegen der „Hive-View“ in ‚Ambari‘

```
1   su - hdfs
2   hadoop fs -mkdir /user/admin
3   hadoop fs -chown admin:hadoop /user/admin
4   hadoop fs -chmod 777 /user/admin
```

Nachdem die Ordner und Berechtigungen im HDFS angelegt bzw. vergeben waren, mussten in der Hive-Config noch Einstellungen vorgenommen werden. Hierfür wurde in der ‚Ambari‘-Weboberfläche unter „Hive → Config → Advanced“ der entsprechende Menüpunkt aufgerufen. Dabei wurden folgende Werte editiert:

Bash 6.23: Anlegen der „Hive-View“ in ‚Ambari‘

```
1   HiveServer2 Heap Size = 512MB
2   Changed Metastore Heap Size = 1024MB
3   Changed Client Head Size = 512MB
```

Abschließend konnte unter dem Menüpunkt „Admin → Manage Ambari → Views → Hive → Create Instance“ eine Hive-View erstellt werden. Dabei wurde als Name „Hive-View“ verwendet, sowie der entsprechend notwendige „Tez View Instance Name“ = TEZ_CLUSTER_INSTANCE vergeben. Nachdem die Einstellungen gespeichert wurden, mussten die HDFS- und Hive-Komponenten neugestartet werden. Anschließend war die Hive-View nutzbar.

Nachdem die Grundinstallation von Hadoop 2.6, sowie die notwendigsten Anpassungen vorgenommen wurden, wurden die weiteren notwendigen Komponenten und Tools installiert.

6.1.4. Installation Zeppelin Notebook

Nachfolgend wird Schritt-für-Schritt die Installation von Zeppelin Notebook beschrieben. Im Rahmen der Laufzeit der Projektgruppe wurden u.a. mehrere unterschiedliche Version von „Zeppelin“ installiert. Von den Installationsschritten her gab es jedoch wenige bis keine Unterschiede je nach Version. Die folgende Installationsanleitung bezieht sich auf die Zeppelin Version 0.7.0-SNAPSHOT. Diese wurde am 21. Juli 2016 gebaut.

Für die Installation von Zeppelin wurde zunächst ein Ordner unter */opt/* angelegt.

Bash 6.24: Installation Zeppelin Notebook 0.7.0-SNAPSHOT

```
1   sudo mkdir /opt/zeppelin-incubating2107
```

Anschließend wurde innerhalb des neu angelegten Ordners das Git-Repository von Zeppelin geklont.

Bash 6.25: Installation Zeppelin Notebook 0.7.0-SNAPSHOT

```
1   cd /opt/zeppelin-incubating2107/
2   sudo git clone https://github.com/apache/zeppelin.git
```

Nachdem das *Clonen* ausgeführt wurde, wurde via „Maven“ das Building ausgeführt. Hierbei mussten die entsprechenden Parameter bzw. Versionsinformationen der Kompenten mit angegeben werden.

Bash 6.26: Installation Zeppelin Notebook 0.7.0-SNAPSHOT

```
cd /opt/zeppelin-incubating2107/zeppelin/
sudo mvn clean package -Pspark-1.5 -Dspark.version=1.5.2 -
    Dhadoop.version=2.7.1 -Phadoop-2.6 -Pyarn -DskipTests
```

Nachdem das Building erfolgreich durchgeführt wurde, musste noch ein Log- und ein Pid-Ordner angelegt werden.

Bash 6.27: Installation Zeppelin Notebook 0.7.0-SNAPSHOT

```
sudo mkdir /opt/zeppelin-incubating2107/zeppelin/log
sudo mkdir /opt/zeppelin-incubating2107/zeppelin/pid
```

Bevor Zeppelin nun gestartet werden konnte, mussten noch zwei Config-Dateien angelegt werden bzw. kopiert werden.

Bash 6.28: Installation Zeppelin Notebook 0.7.0-SNAPSHOT

```
sudo cp /opt/zeppelin-incubating2107/zeppelin/conf/zeppelin-env.
    sh.template /opt/zeppelin-incubating2107/zeppelin/conf/
    zeppelin-env.sh
sudo cp /etc/hive/conf/hive-site.xml /opt/zeppelin-
    incubating2107/zeppelin/conf/
```

In der Config-Datei ‚zeppelin-env.sh' mussten verschiedene Parameter hinzugefügt werden:

Bash 6.29: Installation Zeppelin Notebook 0.7.0-SNAPSHOT

```
sudo nano /opt/zeppelin-incubating2107/zeppelin/conf/zeppelin-
    env.sh
```

Hinzufügen der folgenden Parameter:

Bash 6.30: Installation Zeppelin Notebook 0.7.0-SNAPSHOT

```
export SPARK\_HOME=/usr/hdp/2.3.4.0-3485/spark/
export HADOOP\_HOME=/usr/hdp/2.3.4.0-3485/hadoop/conf/
export ZEPPELIN\_LOG\_DIR=/opt/zeppelin-incubating2107/zeppelin/
    log
```

```
4   export ZEPPELIN\_PID\_DIR=/opt/zeppelin-incubating2107/zeppelin/
        pid
5   export ZEPPELIN\_PORT=9998
```

Zum Start von Zeppelin konnte eine Startdatei verwendet werden.

Bash 6.31: Installation Zeppelin Notebook 0.7.0-SNAPSHOT

```
1   cd /opt/zeppelin-incubating2107/zeppelin
2   sudo bin/zeppelin-daemon.sh start
```

Zeppelin gestartet. Aufrufbar unter „http://localhost:9998“

Nachdem Zeppelin erfolgreich gestartet werden konnte, mussten in den Interpreter-Einstellungen noch zwei Settings editiert werden. Zuerst wurden die Spark Interpreter Settings editiert.

Bash 6.32: Spark Interpreter Settings editieren

```
1   // Properties
2   master  yarn-client
```

Anschließend wurden die Java Database Connectivity (JDBC) Interpreter Settings editiert.

Bash 6.33: JDBC Interpreter Settings editieren

```
1   // Properties editieren / hinzufuegen
2   hive.driver = org.apache.hive.jdbc.HiveDriver
3   hive.url = jdbc:hive2://localhost:10000
4   hive.user = hive
5   hive.password = pgdashHiveDB2016!
6   // Dependencies hinzufuegen
7   org.apache.hive:hive-jdbc:0.14.0
8   org.apache.hadoop:hadoop-common:2.6.0
```

Zeppelin war nun vollständig installiert und unter „http://localhost:9998“ erreichbar.

6.1.5. Installation Airflow

Im Folgenden wird die Installation von Airflow beschrieben. Die Installation wurde unter dem User „hdusermaster“ durchgeführt.

Airflow benötigte eine HOME-Umgebungsvariable. Diese wurde in das */home/hdusermaster*-Verzeichnis gesetzt.

Bash 6.34: Installation Airflow

```
1   export AIRFLOW_HOME=\~\/airflow
```

Bevor die Installation von Airflow gestartet werden konnte, mussten noch zahlreiche Tools und Packages installiert werden.

Bash 6.35: Installation Airflow

```
1   sudo easy_install -U setuptools numpy
2   sudo apt-get install python-numpy
3   sudo apt-get install libpq-dev python-dev libxml2-dev libxslt1-
        dev libldap2-dev libsasl2-dev libffi-dev
4   sudo apt-get install libssl-dev
5   sudo apt-get install libpcap-dev libpq-dev
6   sudo apt-get install libevent-dev
7   sudo apt-get install python-dev build-essential libssl-dev
        libffi-dev libxml2-dev libxslt1-dev zlib1g-dev python-pip
8   sudo apt-get install -y libxml2-dev libxslt1-dev zlib1g-dev
        python3-pip
9   sudo apt-get install libsasl2-modules-gssapi-mit
10  sudo apt-get install python-lxml
11  sudo apt-get install python-numpy libicu-dev
12  sudo apt-get install libssl-dev
13  sudo apt-get install libffi-dev
14  sudo apt-get install libjpeg-dev
15  sudo apt-get install libvirt-dev
16  sudo apt-get install libsqlite3-dev
17  sudo apt-get install libcurl4-openssl-dev
18  sudo apt-get install libxml2-dev libxslt1-dev python-dev
```

Die Installation von Airflow erfolgte mittels ‚pip install‘. Im gleichen Schritt wurden weitere, so genannte „Executor“, für Hive, HDFS, Devel_Hadoop, Celery, JDBC, MySQL und RabbitMQ installiert.

Bash 6.36: Installation Airflow

```
sudo pip install update
sudo pip install "airflow [devel\_hadoop, celery, jdbc, hdfs,
    hive, mysql, rabbitmq]"
```

Nachdem die Installation abgeschlossen war, wurde Airflow gestartet. Dadurch wurde in ‚AIRFLOW_HOME' eine Config-Datei (‚airflow.cfg' angelegt.

Bash 6.37: Installation Airflow

```
sudo airflow webserver
```

Die angelegte Config-Datei wurden nun an verschiedenen Stellen editiert. So wurde einerseits der Connectionstring der Datenbank auf die lokale MySQL-Datenbank auf den Server ‚grouphadoopmaster' geändert, der Executor auf den „CeleryExecutor" geändert, sowie die Standard-URL inkl. des Web-Server-Ports geändert.

Bash 6.38: Installation Airflow

```
sudo nano ~/airflow/airflow.cfg
sql_alchemy_conn = mysql://airflow:airflow@localhost/airflow
executor = LocalExecutor
base_url = http://localhost:9090
web_server_port = 9090
```

Um die Änderungen nun zu übernehmen, musste Airflow neu gestartet werden, sowie die Datenbank reinitialisiert werden.

Bash 6.39: Installation Airflow

```
sudo airflow webserver restart
sudo airflow initdb
```

Airflow war nun lauffähig und konnte mit Hilfe eines Browsers unter dem Port 9090 erreicht werden.

6.1.6. Installation Sqoop

Sqoop wurde zu Testzwecken auf dem Server *grouphadoopmaster* installiert. Im Folgenden wird Schritt-für-Schritt die Installation von Sqoop in der Version 1.4.6 be-

schrieben.

Zunächst wurden die Sqoop-Installationsdateien heruntergeladen.

Bash 6.40: Installation Sqoop 1.4.6

```
1   sudo wget www-eu.apache.org/dist/sqoop/1.4.6/sqoop-1.4.6.
        bin__hadoop-2.0.4-alpha.tar.gz
```

Anschließend wurden die Installationsdateien entpackt und in den Ordner */usr/lib/sqoop* verschoben.

Bash 6.41: Installation Sqoop 1.4.6

```
1   sudo tar -zxvf sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz
2   sudo mv sqoop-1.4.6.bin__hadoop-2.0.4-alpha /usr/lib/sqoop
```

Des Weiteren musste die „.bashrc"-Datei des Users ‚hdusermaster' erweitert um folgende Zeilen erweitert werden und anschließend ausgeführt werden.

Bash 6.42: Installation Sqoop 1.4.6

```
1   export SQOOP_HOME=/usr/lib/sqoop
2   export PATH=\$PATH:\$SQOOP\_HOME/bin
3   .bashrc ausfuehren: source ~/.bashrc
```

Die Konfiguration von Sqoop erfolgt in der Datei „sqoop-env.sh". Diese wurde auf Basis eines vorhandenen Templates erzeugt.

Bash 6.43: Installation Sqoop 1.4.6

```
1   cd \$SQOOP_HOME/conf
2   sudo mv sqoop-env-template.sh sqoop-env.sh
```

Die Datei „sqoop-env.sh" musste um folgende Parameter erweitert werden.

Bash 6.44: Installation Sqoop 1.4.6

```
1   sudo nano sqoop-env.sh
2   export HADOOP\_COMMON\_HOME=/etc/hadoop
3   export HADOOP\_MAPRED\_HOME=/etc/hadoop
```

Für die Verbindung mit den Datenbanken mussten zusätzlich die jeweils benötigten JAR-Dateien im ‚/lib'-Verzeichnis von Sqoop abgelegt werden. Im Rahmen der Projektgruppe wurde hier lediglich der MySQL-Connector benötigt. Diese wurde zunächst heruntergeladen, entpackt und ins ‚/lib'-Verzeichnis von Sqoop kopiert.

Bash 6.45: Installation Sqoop 1.4.6

```
1  sudo wget http://ftp.ntu.edu.tw/MySQL/Downloads/Connector-J/
       mysql-connector-java-5.1.30.tar.gz
2  sudo tar -zxf mysql-connector-java-5.1.30.tar.gz
3  sudo mv mysql-connector-java-5.1.30/mysql-connector-java-5.1.30-
       bin.jar /usr/lib/sqoop/lib
```

Mit dem Befehl ‚sudo sqoop-version' wurde überprüft, ob die Sqoop-Installation erfolgreich war. Bei einer erfolgreichen Installation wurde dabei die entsprechende Version von Sqoop ausgebeben.

Bash 6.46: Installation Sqoop 1.4.6

```
1  sudo sqoop-version
```

Output: ‚Sqoop 1.4.6.2.3.4.0-3485'

Sqoop war somit lauffähig und konnte für den Import von Daten aus relationalen Datenbankensystemen verwendet werden.

6.2. Installation Sonstiger Komponenten

Neben der Installation der Alpha-Version wurden im Rahmen der Projektgruppe noch weitere Tools installiert. Hierzu wurden zwei Server von der Universität zur Verfügung. Die Server waren dabei virtuelle Maschinen die mit einem Linuxbetriebssystem („12.04.5 LTS - GNU/Linux 3.13.0-95-generic x86_64") vorinstalliert waren.

6.2.1. Installation FTP-Server

Für Testzwecke, u.a. für die Entwicklung von AXT, sowie für weitere Test wurde ein FTP-Server installiert. Im Folgenden wird die Installation des FTP-Servers „Very Secure FTPD" Schritt-für-Schritt beschrieben.

Bevor die Installation ausgeführt wurde, war es zunächst notwendig, dass über den „apt-get“-Befehl ein Update zum Neueinlesen der Paketlisten notwendig. Anschließend wurde über „apt-get“ der „Very Secure FTPD“-Server installiert.

Bash 6.47: Installation Very Secure FTPD

```
1   sudo apt-get update
2   sudo apt-get install vsftpd
```

Nach erfolgreicher Installation war es notwendig die Datei ‚vsftpd.conf‘ zu editieren. Hierbei wurden u.a. das Anonymous-Login deaktiviert, Schreibrechte definiert, sowie eine Userlist-Datei hinterlegt.

Bash 6.48: Installation Very Secure FTPD

```
1   sudo nano /etc/vsftpd.conf
2   // Folgende Parameter hinzufuegen:
3   anonymous_enable=NO
4   local_enable=YES
5   write_enable=YES
6   userlist_deny=NO
7   userlist_enable=YES
8   userlist_file=/etc/vsftpd.user_list
```

Die Installation des FTP-Servers war nun abgeschlossen. Der Server konnte über die Befehle ‚sudo service vsftpd start | stop | restart‘ jeweils gestartet, beendet oder neugestartet werden.

FTP-User anlegen

Um Personen zu berechtigen auf den FTP-Server zuzugreifen, mussten entsprechende User angelegt werden. Hierzu waren folgende Schritte notwendig:

Zunächst musste ein lokaler User auf dem Server angelegt werden. Hierzu wurde der Befehl „useradd“ verwendet. Der Parameter ‚-m‘ gab dabei an, dass ein HOME-Verzeichnis angelegt werden soll. Der Parameter ‚-s‘ verwies auf den Pfad „/usr/sbin/nologin“ definierte dabei, dass für den User kein Shell-Zugriff möglich war.

Bash 6.49: FTP-User anlegen

```
1   sudo useradd -m -s /usr/sbin/nologin USERNAME
2   // Eingabe eines Usernamen notwendig
3   sudo passwd USERNAME
4   // Eingabe eines Passworts notwendig
```

Nachdem der User angelegt wurde, musste dieser noch in der Datei ‚vsftpd.user_list' hinterlegt werden, so dass der Zugriff auf den FTP-Server möglich war. Des Weiteren musste der Server neugestartet werden.

Bash 6.50: FTP-User anlegen

```
1   sudo nano /etc/vsftpd.user_list
2   // USERNAME hinzufuegen
3   sudo service vsftpd restart
```

Der angelegte User hatte nun Zugriff auf den FTP-Server.

6.2.2. Installation Git

Im Rahmen der Projektgruppe wurde zur verteilten Versionsverwaltung von Dateien das Tool „Git" eingesetzt. Die Installation von Git wird nun im Folgenden näher beschrieben.

Bevor die Installation ausgeführt wurde, war es zunächst notwendig, dass über den „apt-get"-Befehl ein Update zum Neueinlesen der Paketlisten notwendig. Anschließend wurde über „apt-get" das Package „git-core" installiert.

Bash 6.51: Installation Git

```
1   sudo apt-get update
2   sudo apt-get install git-core
```

Zur einfacheren Verwaltung u.a. hinsichtlich der Berechtigungsvergabe wurde eine Usergruppe ‚gitgroup' angelegt. Des Weitern

Bash 6.52: Installation Git

```
1   sudo groupadd gitgroup
2   sudo git config core.sharedRepository true
```

Git-User anlegen

Um Git nutzen zu können mussten entsprechende User angelegt werden. Hierzu waren folgende Schritte notwendig:

Bash 6.53: Git User anlegen

```
1   sudo useradd -m USERNAME
2   // Eingabe eines Usernamen notwendig
3   sudo passwd USERNAME
4   // Eingabe eines Passworts notwendig
5   sudo adduser USERNAME gitgroup
```

6.2.3. Installation GitLab

Zur einfacheren Verwaltung der vielen Git-Repositorys wurde die Software „GitLab“ installiert. GitLab ist eine Webanwendung und wurde ebenfalls auf einem Server innerhalb der Universität installiert.

Bevor die Installation ausgeführt wurde, war es zunächst notwendig, dass über den „apt-get“-Befehl ein Update zum Neueinlesen der Paketlisten notwendig. Anschließend war es notwendig, einen OpenSSH-Serser sowie sowie Zertifikate zu installieren.

Bash 6.54: Installation GitLab

```
1   sudo apt-get update
2   sudo apt-get install curl openssh-server ca-certificates postfix
3   // Internet waehrend der Installation auswaehlen
```

Anschließend wurden die GitLab Serverpakete heruntergeladen sowie die Installation von GitLab ausgeführt.

Bash 6.55: Installation GitLab

```
1   sudo apt-get install gitlab-ce
2   restart gitlab-ctl
3   sudo gitlab-ctl reconfigure
```

Nachdem die Installation von GitLab ausgeführt wurde, musste ein SSL-Zertifikat erstellt werden.

Bash 6.56: Installation GitLab

```
sudo mkdir /etc/nginx/ssl
sudo openssl req -x509 -nodes -days 365 -newkey rsa:4096 -keyout
    gitlab.key -out gitlab.crt
// Eingabe von entsprechenden Credentials
```

Anschließend musste der Pfad zu diesem Zertifikat, sowie die URL in der Datei „gitlab.rb“ editiert werden.

Bash 6.57: Installation GitLab

```
sudo nano /etc/gitlab/gitlab.rb
// Aendern der externen URL
nginx['ssl_certificate'] = "/etc/nginx/ssl/gitlab.crt"
nginx['ssl_certificate_key'] = "/etc/nginx/ssl/gitlab.key"
external_url 'https://134.106.13.174:4443'
```

In der Datei ‚gitlab.conf‘ wurden die für den https-Zugriff auf die URL notwendigen Einstellungen vorgenommen.

Bash 6.58: Installation GitLab

```
sudo nano /etc/apache2/sites-available/gitlab.conf
// Hinzufuegen des folgenden Inhalts:
<VirtualHost *:443>
  ServerName 134.106.13.174
  SSLEngine on
  SSLCertificateFile /etc/nginx/ssl/gitlab.crt
  SSLCertificateKeyFile /etc/nginx/ssl/gitlab.key
  <Proxy *>
    Require all granted
  </Proxy>
  SSLProxyEngine on
  ProxyRequests Off
  ProxyPass / https://134.106.13.174:4443/
  ProxyPassReverse / https://134.106.13.174:4443/
  Header edit Location ^http://134.106.13.174 https
      ://134.106.13.174/
  RequestHeader set X-Forwarded-Proto "https"
</VirtualHost>
```

Nachdem die notwendigen Anpassungen für den https-Zugriff angewandt

wurden, mussten einige Services gestartet bzw. neugestartet werden.

Bash 6.59: Installation GitLab

```
1   sudo a2enmod headers
2   sudo service apache2 restart
3   sudo a2enmod ssl
4   sudo service apache2 restart
5   sudo gitlab-ctl reconfigure
```

Hiermit war die Installation von GitLab abgeschlossen und unter der Adresse „https://134.106.13.174:4443“ erreichbar.

6.2.4. Installation MySQL-Server

Für verschiedene Tests, vor allem für die Tests von Sqoop, wurde auch eine MySQL-Datenbank installiert. Die für die Installation notwendigen Schritte werden im Folgenden beschrieben.

Bevor die Installation ausgeführt wurde, war es zunächst notwendig, dass über den „apt-get“-Befehl ein Update zum Neueinlesen der Paketlisten notwendig. Anschließend wurde über „apt-get“ der „MySQL“-Server installiert.

Bash 6.60: Installation MySQL-Server

```
1   sudo apt-get update
2   sudo apt-get install mysql-server
```

Nachdem die Installation abgeschlossen wurde, musste die Datei ‚my.cnf‘ verändert werden. Hierbei wurde einerseits konfiguriert, dass auch ein lokaler Zugriff auf die Datenbank möglich war, so die maximale Dateigröße verändert.

Bash 6.61: Installation MySQL-Server

```
1   sudo nano /etc/mysql/my.cnf
2   // Veraendern der 'bind-adress'
3   bind-address = 0.0.0.0
4   // Veraendern der 'max filesize'
5   max filesize = 10M
```

Abschließend wurde noch die maximal Upload-Dateigröße verändert,sowie der Apache-Server neugestartet.

Bash 6.62: Installation MySQL-Server

```
1   sudo nano /etc/php5/apache2/php.ini
2   // Veraeandern der 'upload_max_filesize'
3   upload_max_filesize = 10M
4   sudo service apache2 restart
```

Die Installation des MySQL-Servers war somit abgeschlossen und konnte entsprechend für verwendet werden.

6.2.5. Sphinx

Sphinx wurde im Rahmen der Projektgruppe für die Erstellung von (umfangreichen) Dokumentationen. Sphinx entstammt ursprünglich aus dem Python-Umfeld und wird dort auch häufig eingesetzt. Im Folgenden wird die Installation und Nutzung von Sphinx beschrieben.

Die Installation von Sphinx wurde über den „apt-get“-Befehl ausgeführt.

Bash 6.63: Installation Sphinx

```
1   sudo apt-get install python-sphinx
```

Zur Erstellung einer Dokumentation musste zuerst deren Grundgerüst angelegt werden. Dazu wurde der Sphinx-Assistent verwendet.

Bash 6.64: Initialisierung Sphinx

```
1   sudo sphinx-quickstart
```

Um nun letztendlich Dokumentationen zu erstellen wurde der Befehl ‚sphinx-build‘ verwendet.

Bash 6.65: Nutzung Sphinx

```
1   sudo sphinx-build -b BUILDER QUELLE ZIEL DATEI(EN)
```

Sphinx war nun nutzbar und konnte im Rahmen der Projektgruppe für die Dokumentation der Python-Quellcodes verwendet werden.

6.3. Installation Beta-Version

In diesem Abschnitt wird die Installation der Beta-Version beschrieben. Dabei wird, ähnlich wie im Abschnitt 6.1, vorgegangen, und Schritt-für-Schritt die einzelnen Installationen erläutert. Ein großer Unterschied bei der Installation bzw. den Installation auf der Beta-Version des Projektpartners im Gegensatz zur Alpha-Version war, dass Hadoop als solches schon installiert war. Dabei wurde durch den Projektpartner ein Hadoop-Cluster von *Cloudera* eingesetzt. Die Grafik 6.9 veranschaulicht den groben Aufbau der Servermaschinen innerhalb des CEWE-Netzwerkes.

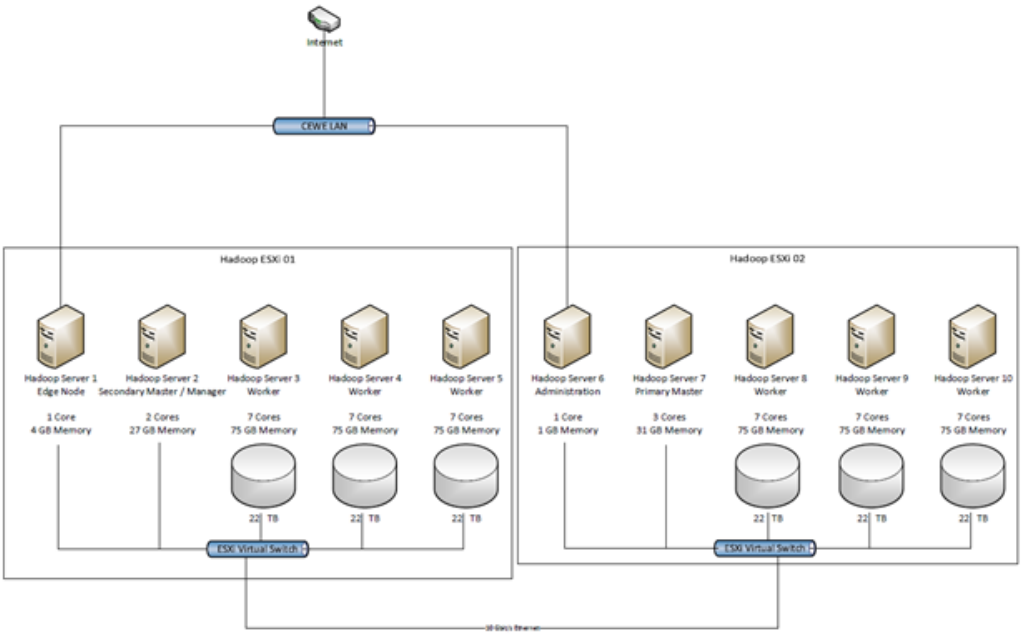


Abbildung 6.9.: Überblick Serverarchitektur Beta-Cluster

Die Grafik zeigt, wie viele Maschinen innerhalb des CEWE-Netzwerks betrieben sind und wie diese innerhalb des Netzwerkes angebunden sind. Sämtliche Installation und Konfigurationen wurden auf dem Server „Hadoop Server 1 - Edge Node“ unter dem Root-Benutzer „dfm1“ ausgeführt.

6.3.1. Installation Docker

Docker wurde im Rahmen der Projektgruppe verwendet, um verschiedene Anwendungen (wie z.B. Zeppelin oder Airflow) mithilfe von Betriebssystemvirtualisierung in Containern zu isolieren. Dies vereinfachte dabei einerseits die Bereitstellung von Anwendungen, da sich Container, die alle nötigen Pakete enthalten,

leicht als Dateien transportieren und installieren lassen. Andererseits gewährleisten Container die Trennung der auf einem Rechner genutzten Ressourcen, sodass ein Container keinen Zugriff auf Ressourcen anderer Container hat.

Docker benötigt als Minimal-Spezifikation für den Betrieb auf einem Linuxsystem eine 64-bit Betriebssystemversion und ein Kernel der Version 3.10. Beide Voraussetzungen waren auf dem Beta-Cluster bei CEWE gegeben, so dass die Installation von Docker keine Probleme darstellte.

1. Im ersten Schritt war es notwendig, dass für den ‚apt-get'-Befehl https-Transport-Pakete, sowie Zertifikate installiert waren.

Bash 6.66: Installation Docker

```
1     sudo apt-get update
2     sudo apt-get install apt-transport-https ca-certificates
```

2. Als nächstes wurde ein ‚apt-key' dem Keyserver hinzugefügt.

Bash 6.67: Installation Docker

```
1   sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.
        net:80 --recv-keys 58118
        E89F3A912897C070ADBF76221572C52609D
```

3. Es folgte die eigentliche Installation von Docker und den weiterhin benötigten Paketen.

Bash 6.68: Installation Docker

```
1   sudo apt-get update
2   sudo apt-get purge lxc-docker
3   sudo apt-cache policy docker-engine
4   sudo apt-get update
5   sudo apt-get install docker-engine
```

4. Docker war nun installiert, und musste lediglich noch gestartet werden. Im gleichen Zuge wurde ein Docker-Test-File heruntergeladen, um die Funktionalität von Docker zu testen.

Bash 6.69: Installation Docker

```
1   sudo service docker start
2   sudo docker run hello-world
```

Docker war nun auf dem Server „Hadoop Server 1 - Edge Node“ installiert und konnte für die Docker-Container für „Zeppelin“ und „Airflow“ verwendet werden.

Hilfreiche Docker-Befehle

Im folgenden werden einige wichtige Docker-Befehle aufgeführt.

Bash 6.70: Starten des Docker Services

```
1   sudo service docker start
```

Bash 6.71: Stoppen des Docker Services

```
1   sudo service docker stop
```

Bash 6.72: Löschen von nicht mehr benötigen Docker-Images

```
1   sudo docker rm -v \$(sudo docker ps -a -q -f status=exited)
2   sudo docker rmi \$(sudo docker images -f "dangling=true" -q)
3   sudo docker run -v /var/run/docker.sock:/var/run/docker.sock -v
        /var/lib/docker:/var/lib/docker --rm martin/docker-cleanup-
        volumes
```

Bash 6.73: Auflisten aller vorhandenen Docker-Images

```
1   sudo docker images
```

Bash 6.74: Löschen eines Docker-Images

```
1   sudo docker rm -f ID\_DES\_IMAGES
```

Bash 6.75: Auflisten aller laufenden Docker-Container

```
1   sudo docker ps
```

Bash 6.76: Wechsel in einen laufenden Docker-Container

```
1   sudo docker exec -it ID\_DES\_CONTAINERS /bin/bash
```

6.3.2. Installation Maven 3.X

Bevor die einzelnen Tools / Komponenten auf der Beta ausgerollt werden konnten, war es notwendig Maven in der Version 3.X auf dem Beta-Cluster zu installieren. Die folgenden Installationsschritte zeigen die Installtion von Maven 3.3.3.

1. Zunächst mussten die benötigten Maven-Installationspakete heruntergeladen und entpackt werden.

Bash 6.77: Installation Maven 3.3.3

```
1   sudo wget https://archive.apache.org/dist/maven/maven
        -3/3.3.3/binaries/apache-maven-3.3.3-bin.tar.gz
2   sudo tar -zxf apache-maven-3.3.3-bin.tar.gz
3   sudo cp -R apache-maven-3.3.3 /usr/local
4   sudo ln -s /usr/local/apache-maven-3.3.3/bin/mvn /usr/bin/
        mvn
```

2. Nachdem Maven heruntergeladen und entpackt war, musste der „.profile“-Datei noch eine Umgebungsvariable hinzugefügt werden.

Bash 6.78: Installation Maven 3.3.3

```
1   echo "export M2\_HOME=/usr/local/apache-maven-3.3.3" >> ~/.
        profile
2   source ~/.profile
```

3. Durch Ausführen des Befehls „mvn -v“ wurde die Version von Maven ausgegeben.

Bash 6.79: Installation Maven 3.3.3

```
1    sudo mvn -v
```

Output: „Maven 3.3.3“

6.3.3. Installation Zeppelin innerhalb eines Docker-Containers

Im Folgenden wird Installation von Zeppelin innerhalb eines Docker-Containers beschrieben. Das verwendete Docker-File wurde dabei auf Basis eines bestehenden Docker-Files[1] erstellt. Hierbei wurden im File einige Erweiterungen und Veränderungen vorgenommen, welche im Quellcodes des Dockerfiles detailliert beschrieben sind. Der Quellcode des Dockerfile ist dem Anhang beigefügt. Das File wurde dabei innerhalb eines GitLab-Repositorys gespeichert und konnte somit sehr schnell auf dem Beta-Cluster von CEWE „gepullt“ werden.

Zunächst wurden ein Unterordner ‚zeppelin_notebook_dir‘ im Verzeichnis ‚/stor/OSF_Tracking‘ erstellt. Dieser war notwendig, damit die Zeppelin-Notebook-Dateien außerhalb des Docker-Containers gespeichert werden. Anschließend wurde innerhalb dieses Verzeichnisses („/stor/OSF_Tracking/git/“) der Inhalt des GitLab-Repositorys „zeppelin_docker_beta“ „gepullt“.

Bash 6.80: Installationsschritte Zeppelin-Docker

```
1    sudo mkdir /stor/OSF_Tracking/zeppelin_notebook_dir
2    cd /stor/OSF\_Tracking/git/
3    sudo git -c http.sslVerify=false clone https
         ://134.106.13.174:4443/pgdash/zeppelin.git
```

Im nächsten Schritt wurde innerhalb des neu erstellten Verzeichnisses zum eigentlichen Dockerfile navigiert und das Dockerimage über den „docker build“-Befehl gebuildet. Hierfür wurde als Imagename „zeppelin_beta_070_010916“ verwendet.

Bash 6.81: Installationsschritte Zeppelin-Docker

```
1    cd /stor/OSF_Tracking/git/zeppelin/Dockerfile_beta/
2    sudo docker build --no-cache -t zeppelin_beta_070_010916 .
```

Nachdem das Building erfolgreich durchgelaufen war, konnte das Dockerimage gestartet werden. Hierfür wurde der Befehl „docker run“ verwendet. In-

[1] https://github.com/tssp/docker-zeppelin

nerhalb dieses Befehls wurden zeitgleich noch Parameter mitgegeben. Dabei war der Parameter „-p“ für zwei Ports (20097 und 20098), der Parameter „-d“ zum Starten als Daemon, sowie „-v“ für das Zeppelin-Notebook-Verzeichnis (/stor/OSF_Tracking/zeppelin_notebook_dir) mitgegeben.

Bash 6.82: Installationsschritte Zeppelin-Docker

```
sudo docker run -d -p 20097:20097 -p 20098:20098 -v /stor/
    OSF_Tracking/zeppelin_notebook_dir:/stor/OSF_Tracking/
    zeppelin_notebook_dir zeppelin_beta_070_010916
```

Zeppelin konnte nun unter „http://localhost:20097“ aufgerufen und gestartet werden. Dabei mussten in den Interpreter-Einstellungen von Zeppelin noch Settings editiert werden.

Bash 6.83: JDBC Interpreter Settings editieren

```
// Properties editieren / hinzufuegen
hive.driver = org.apache.hive.jdbc.HiveDriver
hive.url = jdbc:hive2://hadoop-server1:10000
hive.user = hive
hive.password = hivepassword
// Dependencies hinzufuegen
org.apache.hive:hive-jdbc:0.14.0
org.apache.hadoop:hadoop-common:2.6.0
```

Zeppelin war nun vollständig installiert und unter „http://localhost:20097“ erreichbar.

6.3.4. Installation Airflow innerhalb eines Docker-Containers

Im Folgenden wird Installation von Airflow innerhalb eines Docker-Containers beschrieben. Die verwendeten Docker-Files wurde dabei auf Basis von bestehenden Docker-Files (https://github.com/puckel/docker-airflow) erstellt. Hierbei wurden einige Erweiterungen und Veränderungen vorgenommen, welche im Quellcodes der Dockerfiles detailliert beschrieben sind. Der Quellcode der Dockerfiles ist dem Anhang beigefügt. Das File wurde dabei innerhalb eines GitLab-Repositorys gespeichert und konnte somit sehr schnell auf dem Beta-Cluster von CEWE „gepullt“ werden.

Zunächst wurden ein Unterordner ‚airflow_dir‘ im Verzeichnis „/stor/OSF_Tracking“ erstellt. Dieser war notwendig, damit die Airflow-DAG-

Dateien außerhalb des Docker-Containers gespeichert werden. Anschließend wurde innerhalb dieses Verzeichnisses („/stor/OSF_Tracking/git/“) der Inhalt des GitLab-Repositorys „airflow_beta_docker“ „gepullt“.

Bash 6.84: Installationsschritte Airflow-Docker

```
1   sudo mkdir /stor/OSF_Tracking/airflow_dir
2   sudo git -c http.sslVerify=false clone https
        ://134.106.13.174:4443/pgdash/airflow.git
```

Im nächsten Schritt wurde zum eigentlichen Dockerfile navigiert und das Dockerimage über den „docker build“-Befehl gebuildet. Hierfür wurde als Imagename „puckel/docker-airflow“ verwendet.

Bash 6.85: Installationsschritte Airflow-Docker

```
1   cd /stor/OSF_Tracking/git/airflow/docker
2   sudo docker build --rm -t puckel/docker-airflow .
```

Bevor das Dockerimage nach dem Building gestartet werden konnte, musste noch eine wichtige Anpassung am „/etc/sudoers“-File vorgenommen werden. Hierfür wurde das Programm „visudo“ verwendet. Es war notwendig, dass der Benutzer „dfm1“ alle ‚sudo‘-Befehle ohne Eingabe eines Passworts ausführen konnte. Hierfür musste ein Eintrag im „/etc/sudoers“-File hinzugefügt werden.

Bash 6.86: Installationsschritte Airflow-Docker

```
1   sudo visudo
```

Hinzufügen von:

Bash 6.87: Installationsschritte Airflow-Docker

```
1   dfm1 ALL=(ALL) NOPASSWD: ALL dfm2 ALL=(ALL) NOPASSWD: ALL
```

Zum Starten des Dockerimages wurde der Befehl ‚docker-compose up‘ verwendet. Innerhalb dieses Befehls wurden zeitgleich noch Parameter mitgegeben. Der Parameter „-f“ gab dabei an, dass das File ‚docker-compose-CeleryExecutor.yml‘ verwendet wird und der Parameter „-d“ zum Starten als Daemon.

Bash 6.88: Installationsschritte Airflow-Docker

```
cd /stor/OSF_Tracking/git/airflow/docker
sudo docker-compose -f docker-compose-CeleryExecutor.yml up -d
```

Airflow war nun vollständig installiert und unter „http://localhost:9595“ erreichbar.

6.3.5. Ausrollen der einzelnen GitLab-Repositorys

Für den „Rollout“ der evaluierten Tools und Eigenentwicklungen wurden auf dem Beta-Cluster bei CEWE die einzelnen GitLab-Repositorys „gepullt“. Im Folgenden werden dabei dafür notwendigen Schritten aufgezeigt.

Für AXT wurde zunächst das GitLab-Repository gepullt. Anschließend wurde mittels Maven der Build ausgeführt.

Bash 6.89: Rollout AXT

```
cd /stor/OSF_Tracking/git/
sudo git -c http.sslVerify=false clone https
    ://134.106.13.174:4443/pgdash/axt.git/
cd axt/axt
sudo mvn clean install -c -D skipTests=true
```

Output: „Build Success“

Die für AXT notwendigen Konfigurationen werden im Abschnitt Airflow näher beschrieben.

Für das Deployment von HIMP wurde nach den selben Schritten wie von AXT vorgegangen. Zunächst wurde das GitLab-Repository gepullt und anschließend das Building ausgeführt.

Bash 6.90: Rollout HIMP

```
cd /stor/OSF_Tracking/git/
sudo git -c http.sslVerify=false clone https
    ://134.106.13.174:4443/pgdash/himp.git/
sudo mvn clean install -c -D skipTests=true
```

Output: „Build Success“

Die notwendigen Konfigurationen von HIMP sind ebenfalls im Abschnitt Airflow ersichtlich.

Für SaPPy und den Launcher waren lediglich ein „pullen“ des jeweiligen GitLab-Repositorys notwendig.

Bash 6.91: Rollout SaPPy und Launcher

```
1  cd /stor/OSF_Tracking/git/
2  sudo git -c http.sslVerify=false clone https
       ://134.106.13.174:4443/pgdash/sappy.git/
3  cd /stor/OSF_Tracking/git/
4  sudo git -c http.sslVerify=false clone https
       ://134.106.13.174:4443/pgdash/launcher.git/
```

6.4. Data Import

In diesem Kapitel wird die praktische Umsetzung des Data Imports im Rahmen der Projektgruppe beschrieben. Die Evaluation von Tools für den Data Import (siehe Kapitel 5) hat ergeben, dass für die Anbindung von relationalen Datenbanken *Apache Sqoop* geeignet ist und genutzt werden soll. Für den Import der Log Dateien konnte im Rahmen der Evaluation kein Tool am Markt identifiziert werden, das den Anforderungen der Projektgruppe entspricht. Die Empfehlung der Projektgruppe ist es, eine Eigenentwicklung für diese Aufgabe einzusetzen.

In dem folgenden Abschnitt wird die Möglichkeit des Imports von Tabellen aus relationalen Datenbank durch Apache Sqoop beschrieben. Der zweite Abschnitt dokumentiert die Eigenentwicklung des Tools Agent for XML Transportation (AXT), welches für den Import der Log Dateien entwickelt wurde. Dabei wird die Architektur und des Ablauf des Programms grob umrissen. Weiterhin werden die Konfigurationsmöglichkeiten beschrieben. Zusätzlich sind essentielle Stellen aus dem Quellcode dargestellt und erläutert.

6.4.1. Import relationaler Daten

Für die Verwaltung der Schnittstelle zwischen relationalen Datenbanken und dem Hadoop-Ecosystem wurde nach der Evaluation das Tool *Sqoop* ausgewählt, da es die geforderten Anforderungen erfüllt. Anwendung findet es bei der Übertragung von MySQL-Tabellen und deren Meta-Daten in Hive.

In der Alpha-Version wurde zu Testzwecken ein MySQL-Server installiert und mit extern bezogenen Daten des Projektpartners gefüllt. Hierbei lagen die Daten in Form von Excel-Tabellen vor, die in CSV-Dateien konvertiert und in zuvor definierte MySQL-Tabellen importiert wurden. Anschließend konnten die ausgewähl-

ten Tabellen mit Sqoop in Hive übertragen werden. Hierzu verwendete Befehle mit spezifischen Argumenten hatten folgende Struktur:

Bash 6.92: Sqoop: Schema importieren

```
1   sqoop create-hive-table --connect jdbc:mysql://134.106.11.218/
        ext_data --username root  --password <PWD> --table
        AD_BESUCHE  --hive-table ext_data.AD_BESUCHE
```

Dieses Statement überträgt die Meta-Daten der Tabelle aus MySQL und erstellt eine Hive-Tabelle mit identischer Struktur.

Bash 6.93: Sqoop: Daten importieren

```
1   sqoop import --connect jdbc:mysql://134.106.11.218/ext_data --
        username root  --password <PWD> --table AD_BESUCHE --hive-
        import --hive-table ext_data.AD_BESUCHE
```

Das zweite Statement überträgt nun alle Tabellen-Werte in die Hive-Tabelle.

Nutzung von Sqoop in der Beta-Phase via Airflow-Skript

In der Beta-Phase des Projekts sollten die durch den Projektpartner zur Verfügung gestellten Daten über einen gesicherten Zwischenserver zur Verfügung gestellt werden. Der hierbei eingesetzte MySQL-Server sollte in regelmäßigen Intervallen auf Aktualisierungen geprüft werden und aktuelle Daten via Sqoop in die existierenden Hive-Tabellen übertragen werden.

Aufgrund der Prioritäten und Fokussierung auf die Migration des Hadoop-Ecosystems bei CEWE wurde entschlossen, Sqoop in der Beta-Phase nicht zu implementieren.

Ausblick: Nutzung von Sqoop bei CEWE

In der Praxis kann Sqoop mit wenigen Befehlen gesteuert werden. Das Tool kann hierbei manuell über die *Linux-Shell*, aber auch via *Airflow* eingesetzt werden. Initial müssen alle zu importierenden Tabellen ausgewählt werden und entsprechenden Hive-Datenbanken erstellt werden. Anschließend können die Metadaten der Quelltabellen eingelesen und hieraus entsprechende Hive-Tabellen mit dem Aufrufparameter *„–create-hive-table“* erzeugt werden. Bei bereits existierenden Tabellen wird der Auftrag abgebrochen, ohne Änderungen vorzunehmen. Mit dem Ar-

gument „*–hive-import*“ können das Schema und die Tabelleninhalte der relationalen Tabellen zu Hive übertragen werden. Bei bereits existierenden Tabellen können die Daten mit „*–hive-override*“ aktualisiert werden.

Für die automatisierte Verwaltung der Hive-Zieltabellen kann Airflow genutzt werden. Dabei muss zuvor definiert werden, welche Datenbanken und Tabellen importiert bzw. aktualisiert werden sollen.

6.4.2. Import von XML-Dateien: AXT

Da nach umfangreicher Evaluation kein bestehendes Tool gefunden werden konnte, welches den Import von Dateien den Anforderungen entsprechend realisiert (siehe Kapitel 5.1) wurde eine Eigenentwicklung angestrebt. Ergebnis dieser Eigenentwicklung ist das java-basierte Programm AXT.

Im Rahmen des ELT-Prozesses ist AXT der Einstiegspunkt. Neben der Übertragung der Daten soll AXT auch in der Lage sein, Dateien mit Meta-Informationen wie dem Dateinamen oder der Dateigröße anreichern zu können. Für eine einfachere Datenverarbeitung in den nachgelagerten Schritten soll das Programm die vergleichsweise kleinen Log-Dateien (10kB bis 10 MB) in größere Dateien zusammenfassen können. Dieses wird als *mergen* oder auch *konkatenieren* bezeichnet. Die nachgelagerten Prozesse und Programme greifen auf die von AXT übertragenen und vorbereiteten Dateien zu, um die Informationen aus den Dateien zu extrahieren. Würde die Logik für diese Extraktion angepasst, sollten bereits verarbeitet Dateien die im HDFS liegen erneut in den ELT-Prozess gegeben werden können. Dies wird als *repeat workflow* bezeichnet. Hierfür kann AXT so konfiguriert werden, dass Daten nicht per FTP vom Server bezogen werden, sondern direkt aus dem HDFS bezogen werden um diese erneut anzureichern oder zu konkatenieren.

Technische Anforderungen

Aus der einleitenden Einordnung von AXT in den ELT-Prozess der Projektgruppe ergeben sich diverse technische Anforderungen die zu erfüllen sind. Eine essentielle Anforderung die sich ableiten lässt, ist die Sicherstellung der Korrektheit von Log Dateien. Da diese als XML verarbeitet werden, durch Einfügen von Meta-Informationen aber manipuliert werden können, ist die korrekte Manipulation eine wichtige Anforderungen um zu gewährleisten, dass den nachgelagerten Prozessen gültige XML Dateien bereitgestellt werden. Eine besondere technische Anforderung ist der Umgang mit Eigens entwickeltes Dateiformat für komprimierte Daten (ZML) Dateien. Diese sind gepackte (komprimierte) XML Dateien in einer 1:1 Beziehung, also eine ZML Datei ist immer genau eine XML Datei.

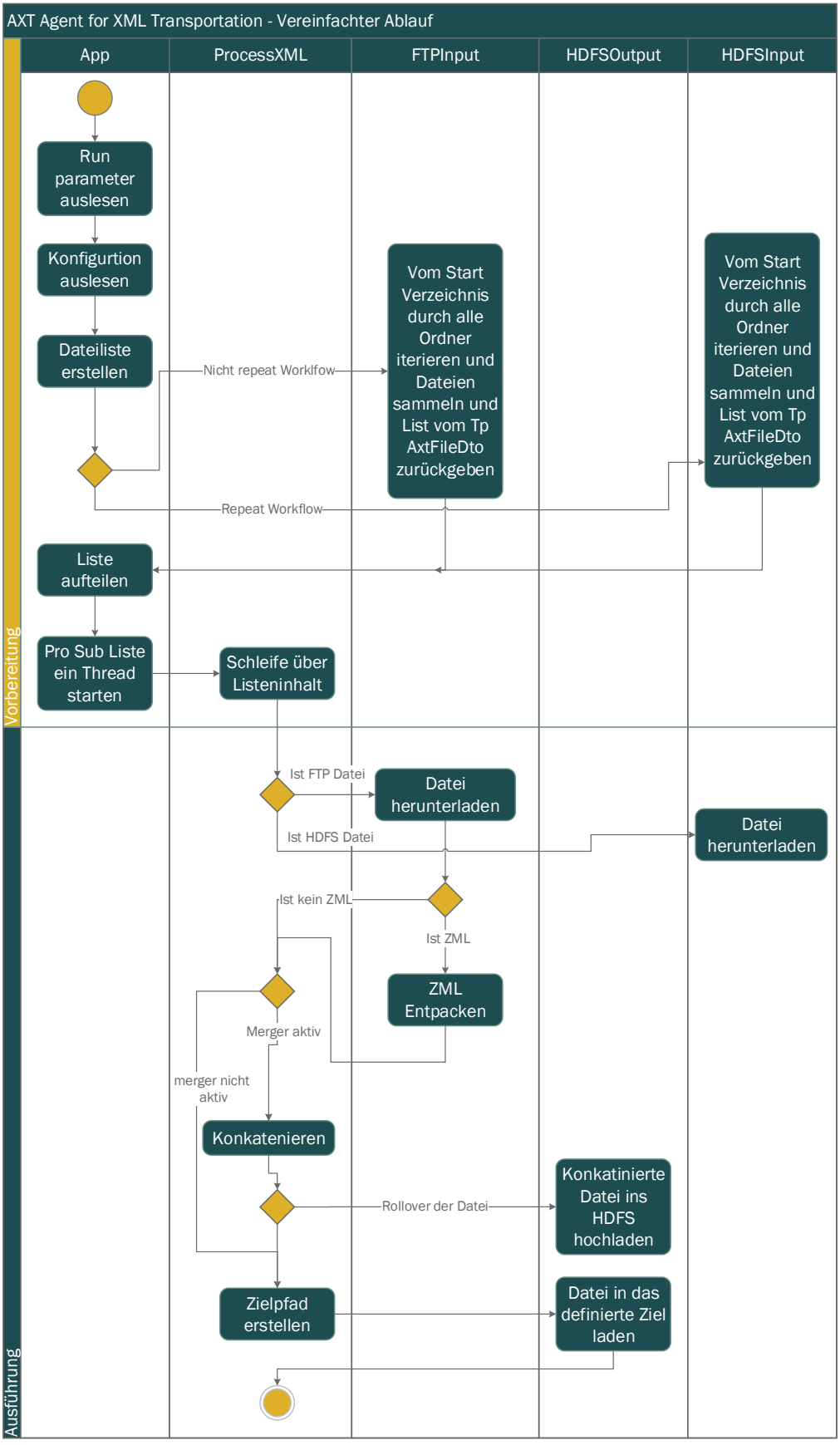


Abbildung 6.10.: Vereinfachtes Aktivitätsdiagramm von „AXT“

Für die korrekte Dekomprimierung müssen die ersten vier Bytes entfernt werden. Es folgt somit, dass eine Dekomprimierung auf Byte-Ebene erfolgen muss. Durch die Einbindung von AXT in ein Workflowmanagementsystem ist die korrekte Terminierung, auch im Fehlerfall, eine wichtige technische Anforderung.

Implementierung

Wie in der Einleitung dieses Abschnittest bereits erwähnt, handelt es sich bei AXT um ein java-basiertes Programm. Um die Verwaltung benötigter Bibliotheken und Abhängigkeiten zu vereinfachen, wurde das Projekt als *maven*-Projekt angelegt. Für die Verbindung zum Sever per FTP wird der FTPClient von *apache.commons.io* genutzt. Mit Hilfe des *hadoop-client* (Hadoop API für Java) wird die Verbindung zum HDFS aufgebaut.

In der Abbildung 6.10 ist ein abstrahierter typischer Programmablauf von AXT zu sehen. Die Begriffe der Abbildung 6.10 werden für das weitere Verständnis folgend erläutert.

Run parameter auslesen: Beim Starten von AXT müssen diverse *run parameter* mitgeben werden. Diese spezifizieren unter anderem den Pfad zur der Konfigurationsdatei. Eine genaue Auflistung der Parameter ist nachfolgend zu finden.

Konfiguration auslesen: Es werden die als *run parameter* mitgegebenen Einstellungen ausgelesen. Zusätzlich werden die Einstellungen aus der Konfigurationsdatei ausgelesen. Eine Übersicht über die Konfigurationen in der Konfigurationsdatei sind im Abschnitt 6.4.2 einzusehen. Sämtliche Einstellungen werden in dem Objekt *AxtConfiguration* gespeichert welches dem Programm überall zur Verfügung steht.

Ermitteln von zu übertragenden Dateien: Nachdem die Einstellungen gelesen worden sind, wird mit deren Hilfe eine Verbindung per FTP zum Server aufgebaut auf dem die Log Dateien zu finden sind. Durch eine Einstellung in der Konfigurationsdatei wird in das Startverzeichnis navigiert. Ergebnis dieses Vorgangs ist eine Liste mit Objekten vom Typ *AxtFileDto*.

Aufteilen der Liste auf Threads: Wenn der *run parameter* für die Anzahl der Threads gesetzt wurde, wird die zuvor erstellte Liste in Sub-Listen aufgeteilt. Es werden so viele Sub-Listen erstellt, wie Threads spezifiziert wurde. Jede Sub-Liste wird an einen Thread übergeben welcher gestartet wird. Die Klasse `ProcessXML.java` beinhaltet die Ausführungslogik des Threads. Die folgenden Unterpunkte wird von jedem Thread ausgeführt. Die Threads arbeite unabhängig und parallel zueinander und beeinflussen sich nicht gegenseitig. Die folgenden Unterpunkte stellen den exemplarischen Ablauf für

einen Thread dar, der wie bereits erwähnt nahezu beliebig vervielfacht werden kann durch die Angabe des Parameters für die Anzahl der Threads.

Herunterladen der aktuellen Datei: Durch eine Schleife wird die übergebene List innerhalb des Threads abgearbeitet. Die folgenden Schritte werden innerhalb dieser Schleife wiederholt. Es wird aus dem `AxtFileDto` die Information bezogen, ob es sich um eine Datei von einem FTP-Server handelt. Ist dies der Fall, wird eine Verbindung zu dem Server aufgebaut (mit Hilfe der in der Konfigurationsdatei spezifizierten Einstellungen). Anschließend wird die Datei mit Hilfe des im `AxtFileDto` gespeicherten absoluten Pfad auf die lokale Festplatte heruntergeladen. In der Konfigurationsdatei kann ebenfalls angegeben werden, ob die heruntergeladene Datei vom FTP-Server gelöscht werden soll. Ist dies mit *true* definiert, so wird die Datei vom Server gelöscht.

Entpacken der Datei Handelt es sich bei der zu ladenden Datei um eine ZML-Datei (zu erkennen an der Dateiendung *.zml*) Muss diese entpackt werden. Das Entpacke wird mit Hilfe der *ZLIB* vorgenommen. Dabei werden Buffer von 1024 Bytes jeweils dekodiert. Zuvor müssen jedoch die ersten 4 Bytes einmalig abgeschnitten werden da es sonst keine gültige Datei ist. Die heruntergeladene, entpackte Datei wird wie erwähnt auf der lokalen Festplatte zwischen gespeichert. Der Speicherort wird ebenfalls aus dem `AxtConfiguration` Objekt entnommen und kann in der Konfigurationsdatei angegeben werden.

Mergen bzw. konkatenieren der Datei: Wurde über den run parameter spezifiziert dass AXT die Dateien mergen bzw. konkatenieren soll, wird nach dem herunterladen und entpacken der Merge angesprochen. Dieser konkateniert die zuvor heruntergeladene Datei zu einer größeren Datei. Dabei spielen zwei Faktoren eine Rolle. Der erste Faktor ist die Anzahl an bereits konkatenierten Dateien. Der zweite Faktor beschreibt die maximale Größe der zuerstellenden Datei. Beide dieser Faktoren werden in der Konfigurationsdatei spezifiziert und werden verwendet um bei erreichen eines der Faktoren eine neue Datei zu erstellen.

Upload der konkatenierten Datei: Ist einer der oben beschriebenen Faktoren erreicht, wird die Datei in das HDFS geladen. Das Ziel im HDFS ist in der Konfigurationsdatei gesetzt ebenso wie ein *true false* Flag welches anzeigt ob die konkatenierte Datei nach dem upload in HDFS von der lokalen Festplatte gelöscht werden soll.

Upload der lokal gespeicherten, entpackten Datei: Die Datei wird nun in das HDFS geladen. Dabei wird zuvor geprüft, dass der Dateiname kei-

ne Zeichen enthält die im HDFS nicht erlaubt sind, wie z.B. „:“. Nach dem Upload zum HDFS wird geprüft ob die lokale Datei gelöscht werden soll. Diese Einstellung ist ebenfalls in der Konfigurationsdatei zu tätigen.

Nachdem nun der Ablauf von AXT beschrieben wurde, werden nun zentrale Code Passagen dargestellt um die Kernfunktionalität auf Entwicklungsebene aufzuzeigen.

Es wird zunächst der Quellcode zum herunterladen einer Datei vom FTP-Server dargestellt.

Java 6.94: AXT - FTPInput.java: Herunterladen über FTP

```
1   // use this method since it handles status code 150 issue
2   // properly
3   resultRetrieve = this.ftpClient.retrieveFile(fileName,
4   byteArrayOutputStream);
5   if (resultRetrieve) {
6     // inflate zml while wirting to local FS
7
8     byte[] byteArray = byteArrayOutputStream.toByteArray();
9
10    // remove first 4 bytes cause they are broken
11    byteArray = Arrays.copyOfRange(byteArray,
12    AXTUtils.BYTES_TO_TRIM_FOR_ZML, byteArray.length);
13
14
15    byte[] decompressedByteArray = new byte[0];
16    try {
17      decompressedByteArray = this.decompress(byteArray,
18      BUFFER_SIZE_FOR_DECOMPRESSING);
19    } catch (DataFormatException e) {
20    e.printStackTrace();
21  }
```

Wie zu sehen ist, wird mit der `retrieveFile` Methode gearbeitet mit einem Kommentar dass diese den Statuscode 150 korrekt verarbeitet. Durch fortlaufende Entwicklertests während der Implementierung wurde festgestellt, dass andere Methoden zum Herunterladen der Dateien Probleme verursachen können. In diesem konkreten Fall war ein Problem, dass der FTP-Server einen Statuscode 150 sendet welcher nicht korrekt verarbeitet werden konnte und der `ftpClient` in einer Endlosschleife gefangen war. Rückgabe werde der `retrieveFile` Methode ist ein Boolean der den Wert `true` oder `false` annehmen kann. Ist er `true`, so wurde das

Herunterladen erfolgreich Abgeschlossen. Dieser Ausschnitt zeigt den Standardfall, dass eine ZML-Datei heruntergeladen wurde. Diese muss entpackt werden. Dazu werden, wie zuvor bereits erwähnt, die ersten 4 Byte abgeschnitten und die `decompress` Methode aufgerufen, welche nun auf Byteebene die die ZML-Datei entpackt. Dieser Vorgang ist im nachfolgenden Quellcode implementiert.

Java 6.95: AXT - FTPInput.java: Entpacken von ZML

```
1     /**
2     * Method which decompresses the given byte array.
3     * @param byteArrayToDecompress the byte array to be
          decompressed
4     * @param bufferSize the buffer size in which the array will be
          decompressed
5     * @return the decompressed array
6     * @throws DataFormatException In case the bytes to decompress
          are not
7     * in ZLIB format
8     * @throws IOException In case the byte could not be written to
          the
9     * outputstream
10    */
11    private byte[] decompress(final byte[] byteArrayToDecompress,
          final int
12    bufferSize) throws DataFormatException, IOException {
13
14      Inflater inflater = new Inflater();
15      inflater.setInput(byteArrayToDecompress);
16
17      LOG.info("Original length of byte array to be decompressed:
            "
18      + byteArrayToDecompress.length);
19
20      ByteArrayOutputStream byteArrayOutputStream  =
21      new ByteArrayOutputStream(byteArrayToDecompress.length);
22      byte[] bufferArray = new byte[bufferSize];
23      int i = 0;
24      while (!inflater.finished()) {
25        LOG.info("loop run: " + i);
26        int count = inflater.inflate(bufferArray);
27        byteArrayOutputStream.write(bufferArray, 0, count);
28        i++;
29      }
30
31      byteArrayOutputStream.close();
32
```

```
33        return byteArrayOutputStream.toByteArray();
34
35      }
```

Nachdem nun ein Auszug des Quellcodes für das Herunterladen angeführt wurde, wird im Folgenden ein Ausschnitt dargestellt, der das Hochladen in das HDFS realisiert.

Java 6.96: AXT - HDFSOutput.java: Upload zum HDFS

```
1       /**
2       * Creates a file with the passed filename and the given
            content in HDFS.
3       * @param fileName The name of the new file in the HDFS. Note:
            If this
4       *                 does NOT contain the path, the file will be
            created w
5       *                 ithin the current directoy
6       * @param  pathToFileOnDisk path to the file on disk which
            should be
7       *                          written to HDFS
8       * @return <code>true</code> if the file was
9       * successfully create, <code>false</code> otherwise
10      */
11      public final boolean writeFile(final String fileName,
12      final String pathToFileOnDisk) {
13        try {
14          this.userGroupInformation.doAs(new HDFSWriteAction(this.
                fileSystem,
15          fileName, pathToFileOnDisk));
16          return true;
17        } catch (Exception e) {
18        LOG.error(e.getMessage());
19      }
20      return false;
21    }
```

Wie in dem Quellcodeauszug zu sehen ist, definiert die Klasse `HDFSOuput` eine `writeFile` Methode, die einen Dateinamen und einen Pfad zu einer im lokalen Dateisystem existierenden Datei entgegen nimmt. Der Dateiname definiert dabei, wie die hochzuladene Datei im HDFS heißen soll. Zu sehen ist, das eine neue `HDFSWriteAction` ausgeführt wird über den Aufruf von `userGroupInformation.doAs()`. Um dies nutzen zu können, ist im folgenden der Konstruktor dieser Klasse aufgeführt in der das FileSystem initialisiert wird.

Java 6.97: AXT - HDFSOutput.java: Initialisieren des FileSystem Objekts für HDFS

```
1     /**
2     * Constructor for an HDFSOutput.
3     * @param defaultFs the starting point in the file system.
4     * @throws Exception thrown in case it was not possible to
          create a
5     * files ystem object.
6     */
7     public HDFSOutput(final String defaultFs)
8     throws Exception {
9
10      this.defaultFS = defaultFs;
11      this.configuration = new Configuration();
12      this.userGroupInformation = UserGroupInformation.
            createRemoteUser(
13      "hdfs");
14
15      // set this beaucse maven-assembly might break things, check
16      // http://stackoverflow.com/questions/17265002/
17      // hadoop-no-filesystem-for-scheme-file
18      configuration.set("fs.hdfs.impl",
19      "org.apache.hadoop.hdfs.DistributedFileSystem");
20      configuration.set("fs.defaultFS", defaultFS);
21      configuration.set("hadoop.job.ugi", "hdfs");
22
23      fileSystem = FileSystem.get(configuration);
24
25    }
```

Es gilt zu beachten, dass für die `userGroupInformation` derzeit ein fester User *hdfs* genutzt wird, da dies ein Standarduser ist der in jedem HDFS System vorhanden sein sollte. Zu dem `configuration` Objekt ist zu sagen, dass die Konfigurationen zunächst Standardwerte sind. Hervorzuheben ist das setzen der Konfiguration `fs.defaultFS`. Dies wird mit einem Parameter gesetzt, welcher vom Nutzer über die Konfigurationsdatei gesetzt werden kann. Er gibt das Startverzeichnis an. Darin enthalten ist auch die die Serveradresse und der Port.

Die `writeFile` Methode nutzt eine neuen `HDFSWriteAction`. In dieser Klasse ist der eigentliche upload gekapselt. Dies ist nachfolgend als Quellcodeausschnitt zu sehen.

Java 6.98: AXT - HDFSWriteAction.java: HDFS Upload

```
1     /**
2     * Runs the actual write operation.
3     * @return Default return null.
4     * @throws Exception If the write operation could not be
          executed.
5     */
6     public final Void run() throws Exception {
7       LOG.info("Starting new write action in HDFS");
8       this.fileSystem.copyFromLocalFile(new Path(this.
            pathToFileOnDisk),
9       new Path(this.pathOfFileToBeCreate));
10      LOG.info("Uploaded from disk: " + this.pathToFileOnDisk
11      + " into " + this.pathOfFileToBeCreate);
12      return null;
13    }
```

In dieser Methode wird über das `FileSystem` Objekt, welches die API Implementierung für die Methode des HDFS darstellt, die Methode `copyFromLocalFile` aufgerufen. Dieser Methodew ird der Pfad zur lokalen Datei übergeben die hochgeladen werden soll, sowie der Pfad mit der Zielstruktur für das HDFS

AXT Konfigurationsdatei

Im vorangegangen Abschnitt wurde bereits auf die Konfigurationsdatei hingewiesen. Nachfolgende ist diese exemplarisch dargestellt.

Config 6.99: Konfigurationsdatei von AXT

```
1   ########## FTP connection configuration ##########
2   #server IP
3   #axt.ftp.server.ip=192.168.112.199
4   axt.ftp.server.ip=127.0.0.1
5   #username
6   axt.ftp.server.username=ftpuser
7   #password
8   axt.ftp.server.password=dash_elefant
9   #the working directory, $$$$ will be replaced by date to be able
        to load specifically stuff
10  axt.ftp.server.workingDir=/var/xml_data/$$$$
11  #set true if the server should delete files after download
12  axt.ftp.server.deleteFiles=false
13
14
```

```
15  ########## HDFS configuration (when running axt on local machine
        replace grouphadoopmaster.local with localhost ##########
16  axt.hdfs.destination.rawData=hdfs://localhost/xml_archive/
17  #specify the hdfs directory where the merged file should be
        uploaded
18  axt.hdfs.destination.mergedData=hdfs://localhost/ready2Parse/
19  #specify the source which should be used when running axt as
        repeat workflow. most likely this is the same directory as
        for raw data
20  axt.hdfs.source.repeatWorkflow=hdfs://localhost/xml_archive/$$$$
21
22
23  ########## temporary (download) folder configuration ##########
24  #where to save the downloaded file (raw data)
25  axt.temp.rawData.download.dir=d:/opt/pgdash/xmlDownloaded
26  #set to true if downloaded file should removed after usage or
        false if not
27  axt.temp.rawData.deleteAfterDownload=false
28  #specify where the merged file should be saved on local system
29  axt.temp.merger.download.dir=d:/opt/pgdash/xmlMerged
30  #specify if the merged file on local system should be deleted
        after usage
31  axt.temp.merger.deleteAfterDownload=false
32
33
34  ########## merger configuration - the destination path on hdfs
        is specified in hdfs config block ##########
35  #specify the required file size of merged file in MB. note that
        additional tags i.e. root, filename also need some space
36  axt.merger.mergedFileSize=30
37  #specify the max number of files merged into one file
38  axt.merger.maxNumberOfFiles=1000
39  #true if singe xml should contain filename / filesize, false
        otherwise
40  axt.util.insert.filenameAndSizeToRawData=true
```

Die Aufteilung innerhalb der Konfigurationsdatei zeigt deutlich vier Abschnitte die konfiguriert werden können.

1. *FTP Konfigurationen:* Hier werden Einstellungen rund um FTP vorgenommen. Der erste spezifizierte Parameter, ist die IP Adresse des FTP -Servers. Die zwei folgenden Parameter sind der Username und das Password, also die Login-Daten, die genutzt werden um sich beim FTP-Server anzumelden. Als dritter Paraemter ist das *workingDirectory* angegeben. Die ist der Einstiegspunkt von welchem aus AXT beginnt die Ordner zu durchsuchen. Es

ist auf den Platzhalter, *$$$$*, hinzuweisen. Dies rührt durch die Ordnerstruktur von CEWE her. Die Log Dateien werden auf dem FTP-Server in einer Ordnerstruktur abgelegt die aus einem Datum gebildet wird. Dieser Platzhalter wird also mit einem Datum ersetzt, welches bei starten von AXT übergeben wird. Als letzte Einstellung kann definiert werden, ob Dateien vom FTP-Server nach erfolgreicher Verarbeitung von AXT gelöscht werden sollen.

2. *HDFS Konfigurationen:* In diesem Block werden Einstellungen bezüglich der Verbindung zum HDFS vorgenommen. Die erste Einstellung, *rawData* gibt an, wo im HDFS die einzelnen XML Dateien gespeichert werden sollen. Auch hier gilt es, den Plazhalter *$$$$* zu beachten. Wie auch für FTP gibt es auch für die Ordnerstruktur im HDFS eine Vorgabe die aus einem Datum generiert wird. Genau so gibt es auch einen run parameter, über welchen dieses Datum mitgegeben wird. Zu beachten ist außerdem, die Notwendigkeit des prefix *hdfs://* welcher benötigt wird um eindeutig anzuzeigen dass es sich um einen Pfad für HDFS handelt. Dies gilt auch für den folgenden Paramter *mergedData* welcher spezifiziert wo die konkatenierten Dateien gespeichert werden sollen. Der letzte Parameter gibt an, von welchem Pfad aus, Dateien für den repeat Workflow bezogen werden sollen. Hier wird ebenfalls mit dem Platzhalter gearbeitet, um zu bestimmten Tagen zu navigieren zu können.
3. *Konfigurationen für den lokalen Zwischenspeicher:* Wie im Abschnitt der Implementierung (6.4.2) bereits erwähnt, werden die vom FTP-Server geladenen Datei auf der lokalen Festplatte gespeichert. Der Ort, wo diese Dateien gespeichert werden sollen, wird im ersten Parameter spezifiziert. Der nachfolgende Parameter gibt an, ob diese Dateien auf der lokalen Festplatte nach erfolgreicher Weiterverarbeitung (Upload ins HDFS) gelöscht werden sollen. Als Dritter Parameter, wird spezifiziert, wo die gemergten bzw. konkatenierten Dateien abgelegt werden sollen. Auch hier gibt der folgenden und somit vierter Parameter in diesem Block an, ob die konkatenierten
4. *Konfigurationen für der Merger:* In dem letzten Block der Konfigurationsdatei werden Einstellungen für den Merger bzw Konkatenierer vorgenommen. Der erste Parameter definiert die maximale Größe der Datei, in der die kleineren XML-Dateien konkateniert werden. Wird diese Größe erreicht, wird ein *rollover* angestoßen. Rollover bezeichnet den Vorgang, bei dem eine Datei als beendet angesehen wird, und eine neue Datei erstellt wird in der dann wieder andere Dateien zusammengefasst werden. Mit Hilfe des zweiten Parameters, wird spezifiziert wie viele einzelnen Dateien, maximal, in einer

Datei zusammengefasst werden sollen, bis ein *rollover* passieren soll. Beide Einstellungen bestimmen also einen Rollover. Es gibt zwischen diesen Kriterien keine Wertigkeit, d.h. egal welches Kriterium zuerst erfüllt wird, es wird sofort ein Rollover angestoßen. Der dritte Parameter gibt an, ob Meta-Informationen in die einzeln entpackten Dateien integriert werden sollen.

AXT Ausführung - Parameter

In den vorangestellten Abschnitten wurde bereits auf die *run parameter* verwiesen die beim Start von AXT übergeben werden. In diesem Abschnitt wird nun der Start von AXT vorgestellt sowie die einzelnen Parameter erläutert.

Um AXT zu starten, muss zunächst an den Ort navigiert werden, an dem die *axt-0.0.1-SNAPSHOT-jar-with-dependencies.jar* liegt. Der Aufruf dieser Java Archive (JAR) Datei ist im folgenden dargestellt.

Bash 6.100: Starten von AXT

```
1   java -jar axt-0.0.1-SNAPSHOT-jar-with-dependencies.jar "path/to/
        config.properties" 2016-09-12 2016-12-09 false true 2
```

Nach der Angabe der JAR die ausgeführt werden soll, folgen die *run paraemter*. Für AXT müssen wie zu sehen sind Sechs Parameter spezifiziert werden, wobei der letzte optional ist. Die Funktion der einzelnen Parameter sind wie folgt:

1. Parameter - cofig file: Der erste Parameter ist ein Pfad, der angibt wo die Konfigurationsdatei zu finden ist. Diese wird benötigt, um weitere Einstellungen zu laden wie z.B. die Verbindungseinstellungen zum FTP-Server. Eine vollständige Auflistung der Einstellungsmöglichkeiten ist im vorangegangene Abschnitt einzusehen.

2. Parameter - FTP Date: Der zweite Parameter spezifiziert ein Datum, welches genutzt wird, um auf dem FTP-Server den gewünschten Tag herunterzuladen und in das HDFS zu laden. Mit Hilfe dieses Datums wird der Platzhalter aus der Konfigurationsdatei für FTP ersetzt. Zu Beachten gilt, das Datum im Format *YYYY-MM-DD*, also Jahr-Monat-Tag, anzugeben.

3. Parameter - HDFS Date: Der dritte Parameter ist ebenfalls ein Datum im Format *YYYY-MM-DD* welches dazu dazu genutzt wird, eine Datums-basierte Ordnerstruktur im HDFS anzulegen. Mit Hilfe dieses Parameters wird der Platzhalter in der HDFS Konfiguration aus der Konfigurationsdatei ersetzt.

4. Parameter - Repeat Workflow Der vierte Parameter ist ein boolscher Wert (*true* bzw. *false*) der spezifiziert ob AXT einen repeat workflow durchführen soll.

Ist dies der Falll, wird mit Hilfe des dritten Parameters die einzelnen XML-Dateie aus dem HDFS geladen und weiter verarbeitet und nicht vom FTP.

5. **Parameter - Merger:** Der fünfte Parameter, ebenfalls ein boolscher Wert, spezifiziert ob der Merger bzw. Konkatenierer aktiv ist. Ist *true* gegeben, so wird AXT die einzelnen Dateien in große Dateien zusammenfassen. Nach welchen Regeln diese zusammengefasste Datei entsteht kann in der Konfigurationsdatei bestimmt werden und ist im vorhergehenden Abschnitt beschrieben. Wird als Parameterwert aber *false* übergeben, so werden keine zusammengefassten Dateien erstellt, die Konfiguration aus der Konfigurationsdatei spielt nun keine Rolle mehr.
6. **Parameter - Anzahl der Threads** Der Sechste und somit letzte Parameter gibt an, wie viele Threads von AXT erstellt werden sollen, um die Dateien vom FTP-Server zu verarbeiten. Die genaue Funktionsweise, wie die Dateien auf Threads aufgeteilt werden, kann im Abschnitt der Implementierung nachgeschlagen werden. Außerdem stellt dieser Parameter eine Besonderheit da, weil er *optional* ist, d.h. er muss nicht gesetzt werden. Wird dieser Parameter nicht gesetzt, so wird AXT nur einen Thread starten um die Dateien zu verarbeiten.

Fazit

Die java-basierte Eigenentwicklung AXT schließt die Lücke, die zwischen FTP-Server und HDFS klafft. Es konnte erfolgreich ein Tool entwickelt werden, welches als Eingang Dateien von einem FTP-Server beziehen kann und als Ausgang diese bezogenen Dateien in ein HDFS ablegt. Durch die Kapselung der Logik in Threads und die Konfigurationsmöglichkeit über run parameter können auch große Mengen an Dateien effizient und in kurzer Zeit übetragen werden. Indem sämtliche Konfigurationseinstellungen in eine separate Konfigurationsdatei ausgelagert wurde, besteht die Möglichkeit schnelle Änderungen vorzunehmen ohne Quellcode anpassen zu müssen. Durch den Einsatz von java mit dem Objektorientierten Ansatz ist eine gute Wartbarkeit gewährleistet und bietet eine gute Grundlage für eine Weiterenticklung.

6.5. Data Transformation

Nachdem die unstrukturierten XML-Rohdaten in das HDFS geladen wurden (Abschnitt 6.4) findet eine Transformation der Daten statt.

Während der explorativen Datenanalyse, die auch einen Workshop mit dem Unternehmenspartner umfasste, wurden die interessantesten Informationen, die in

den XML-Dateien enthalten sind extrahiert und auf dieser Basis eine Zielstruktur entwickelt. Diese Zielstruktur wird durch den Datenbanklayer „Hive“ verwaltet und in Unterabschnitt 6.5.1 beschrieben. Die Transformation der Informationen aus den XML-Dateien in das relationale Schema wird durch einen Parser durchgeführt. In Unterabschnitt 6.5.2 wird die Architektur des Parsers beschrieben. Die Implementierung eines Parsers wird in Unterabschnitt 6.5.3 erläutert. Hier werden auch Probleme, die während der Implementierung auftraten diskutiert und als Konsequenz daraus in Unterabschnitt 6.5.4 mögliche Alternativen aufgezeigt. Die finale Implementierung des Parsers wird schließlich ausführlich in Unterabschnitt 6.5.5 beschrieben. Für die Ausführung des Parsers wurden die beiden Alternativen Hadoop Streaming (Abschnitt 6.5.6) und Spark (Abschnitt 6.5.7) untersucht.

6.5.1. Datenbanklayer in Hive

Zur Persistierung der geparsten Daten aus den XML Dateien wird eine Datenbank implementiert. Dieser Layer dient zur strukturierten Speicherung der Informationen aus den XML Dateien. Die Datenbank wird in Hive verwaltet. Sie bildet die Ausgangsbasis für einfache Analysen und für Data Mining Vorhaben. Die einfachen Analysen ermitteln beispielsweise Kennzahlen für das Dashboard, während die Daten für Data Mining Methoden zunächst aufbereitet werden müssen. In diesem Abschnitt wird der Aufbau des relationalen Layers beschrieben.

Alle Datenbanktabellen sind partitioniert. Eine Partition umfasst jeweils eine Tagesverarbeitung. Die Partitionierung ist erforderlich, um Zeitscheiben aus dem Hive Schema zu löschen, da delete und update Operationen, wie in üblichen relationalen Datenbanksystemen, nicht möglich sind. Partitionen hingegen können gelöscht werden.

In der Abbildung 6.11 sind alle Datenbanktabellen mit ihren Primär- und Fremdschlüsseln dargestellt. Außerdem sind die Beziehungen der Tabellen untereinander zu sehen. Bei den Schlüsseln und Relationen handelt es sich jedoch lediglich um semantische Beziehungen. In Hive selber sind keine Constraints vorgesehen.

Im Folgenden werden die einzelnen Tabellen und die wichtigsten Attribute beschrieben. Die weiteren Attribute sind dem Datenbankschema in Anhang A.2 zu entnehmen.

Die Struktur der XML Dateien lässt sich in verschiedene Bereiche unterteilen, die auch die Grundlage für die Struktur der Datenbankrelationen bilden. Am Anfang jeder XML Datei werden allgemeine Informationen über die Session, den genutzten DFM und den Handelspartner gespeichert. Die Tags `<tracker>`, `<terminal>` und `<ticket>` enthalten diese Informationen. Informationen über die

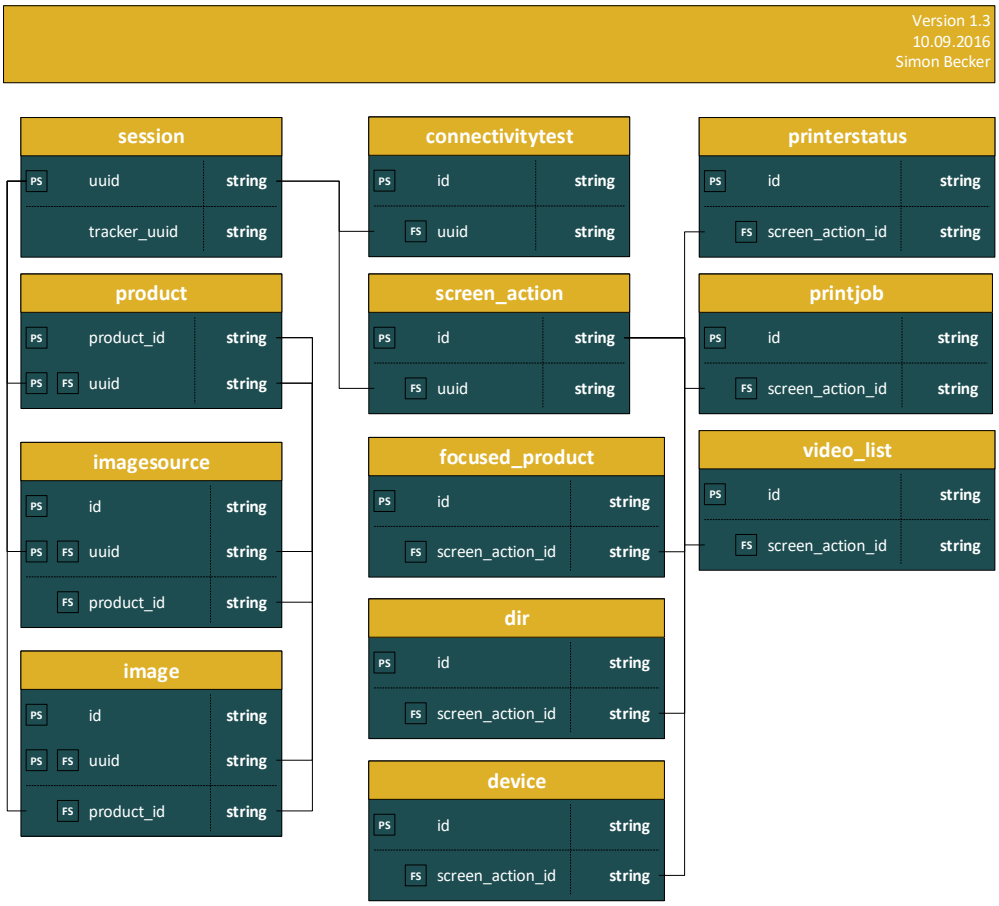


Abbildung 6.11.: Übersicht über die Beziehungen der Relationen

den Auftrag finden sich im Tag `<summary>`. Diese geparsten Informationen werden in der Relation session gesammelt. In der Tabelle 6.1 sind alle Attribute aufgeführt.

session

Attribut	Datentyp	Beschreibung
uuid	string	Eindeutige ID mit der jede Session und damit jede XML Datei eindeutig definiert wird.
fileSize	int	Größe der XML Datei
timestamp	timestamp	Datum und Uhrzeit der XML Datei

Tabelle 6.1.: Attribute der Tabelle session

Im XML werden auch Informationen über das Verhalten des Nutzers am DFM protokolliert. Die Informationen werden im Tag `<path>` gekapselt. Grundsätzlich wird zwischen verschiedenen Aktionen unterschieden. So gibt es die Tags `<enter>`,

<leave>, <medium> und <mifare>. Die Attribute der Tags beinhalten grundlegende Informationen zu den Klicks des Kunden. Zusätzlich enthält jedes Tag eigene Attribute. In der Tabelle screen_action werden die Tags gespeichert. Für das Speichern der Clickstream Daten ist es insbesondere wichtig, dass die Reihenfolge der Tags wiederhergestellt werden kann. Dazu für jede Session ein Zähler initialisiert und hochgezählt. In der Tabelle 6.2 sind die Attribute der Tabelle dargestellt und die wichtigsten beschrieben.

screen_action

Attribut	Datentyp	Beschreibung
id	string	Generierter eindeutiger Schlüssel je Session.
uuid	string	uuid der Session zur Zuordnung des Clickstreams zur Session.
sequence	int	Zähler je Session, der die Reihenfolge der Ereignisse wiedergibt.
action_type	string	Attribut zur Beschreibung der Ereignisart. (enter, leave, ...)

Tabelle 6.2.: Attribute der Tabelle screen_action

Die verschiedenen Ereignisarten können eine unterschiedliche Anzahl verschachtelter Untertags enthalten. Diese werden in separate Tabellen geparst. Im Tag <medium> können die Untertags <dir> und <device> vorkommen. Über die <dir> -Tags werden die Verzeichnisstrukturen des genutzten Mediums extrahiert und in die Tabelle dir gespeichert. In der Tabelle device finden sich weitere Informationen zum eingesetzten Medium. Die Tabellen printjob und printerstatus enthalten Informationen über den eingesetzten Drucker, bzw. den aktuellen Druckauftrag. Die Informationen treten in den XML Dateien als Untertags der Clickstream Informationen auf. Das Tag <focusedProduct> tritt ebenfalls als Untertag in den Tags <enter> bzw. <leave> auf. Es enthält Informationen über ein aktuell betrachtetes Produkt in Relation zum übergeordneten Tag. Im <summary> Tag werden die Informationen der Session zusammengefasst. Einen großen Teil des Tags bilden Informationen über die gekauften Produkte und die verwendeten Bilder. Die Bilder werden den entsprechenden Produkten für welche sie verwendet wurden zugeordnet. Im Datenbankschema werden die Informationen über die Tabellen image und product abgebildet. Die Tabelle image enthält einige Informationen über das Bild an sich, insbesondere auch die Exif Daten. In der Entität product finden sich Informationen zu den gekauften Produkten, wie zum Beispiel die Artikelnummer, der Preis und die Bezeichnung des Produkts.

6.5.2. XML Parser Architektur

Der XML-Parser bildet eine zentrale Komponente des ELT-Prozesses. Als Input bekommt der Parser die XML-Dateien angeliefert. Anschließend werden die Informationen extrahiert und in die geforderte strukturierte Form gebracht. In diesem Abschnitt wird die Architektur des Parser-Moduls anhand der Grafik 6.12 vorgestellt.

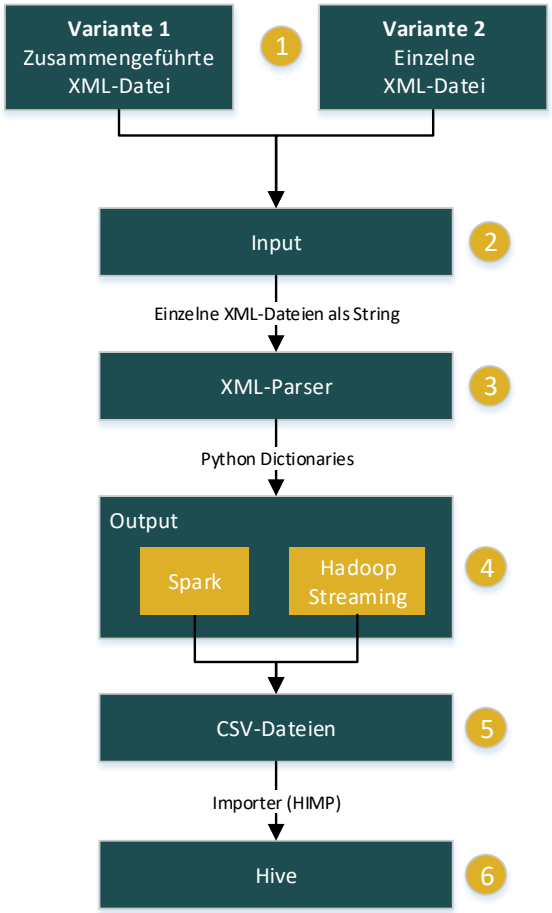


Abbildung 6.12.: Architektur des XML-Parsers

Die Abbildung zeigt die einzelnen Verarbeitungsschritte des Parsers. Die Nummerierung neben den Kästen zeigt die Reihenfolge an, in der die Schritte vollzogen werden. Das Parser-Modul kann zwei verschiedene Arten von Input verarbeiten. In Schritt 1 sind die beiden Varianten aufgeführt. Der Input kann entweder aus einer einzelnen XML-Datei bestehen, oder aus einer Datei, in der mehrere XML-Dateien zusammengeführt wurden. Die Input Klasse in Schritt 2 splittet die Dateien bei Bedarf und leitet den Inhalt des XML als Zeichenkette weiter. Den Kern

des Moduls bildet der XML-Parser (Schritt 3). Dieser nimmt den einzelnen XML-String, parst diesen komplett und gibt Key-Value Paare zurück. Dabei entspricht der Key dem Tabellennamen in Hive und der Value stellt die einzelnen Datensätze dar. Je nach Ausführungsvariante (Hadoop Streaming oder Apache Spark) werden die Key-Value Paare verarbeitet (Schritt 4) und in CSV-Dateien exportiert. Die generierten CSV-Dateien liegen im HDFS verteilt und sind bereit zum Importieren in die Hive-Tabellen (Schritt 5). Für den Import in das Datenbankschema sorgt das in Java geschriebene Programm HIMP.

6.5.3. Minidom Parser-Modul

Der *Pyxmlparser*, auch Minidom Parser genannt, ist eine Python-basierte Eigenentwicklung. Bei dem Minidom Parser ist die API `xml.minidom` verwendet worden. Der *Pyxmlparser* übersetzt hierarchische XML-Dateien in ein relationales Format.

Aufbau

Bei der Entwicklung des Parser-Moduls wurde sich am Aufbau der XML-Datei orientiert. Somit unterteilt sich der *Pyxmlparser*, wie auch die XML-Datei, in die Bereiche *Tracker, Terminal, Ticket, Path* und *Summary*. Zu jedem Bereich wird mindestens eine Python-Liste erzeugt. Die Listen werden zusammengefasst in einem Python-Dictionary hinterlegt und nach einem Aufruf zurückgegeben. Dies wird folgend durch die Mainmethode *def parser()* dargestellt.

Python 6.101: Pyxmlparser: Initialisierung der Dictionaries

```
1   def parser(doc, path_to_csv='data/CSVs/'):
2     ## output Dictionary (to be returned or written in CSV)
3     _dict = {'errorlog': list(),'session': list(),'screen_action':
          [],'printjob': [],'printerstatus': [],'focused_product':
         [],'device': [],'dir': [],'product': [],'raw_image': [],'
         edit_image': [],'video_list': [], 'connectivitytest': []}
4     ERROR = _dict
5     xmldoc = None
6     ## output lists
7     screen_action_list = []
8     screen_action_enter_list = []
9     screen_action_leave_list = []
10    screen_action_medium_list = []
11    screen_action_activate_list = []
12    screen_action_mifare_list = []
13    screen_action_event_list = []
```

Zu Beginn der Verarbeitung wird differenziert, ob das Modul von Apache Spark oder Hadoop Streaming aufgerufen wird. Es werden beide Varianten in Bezug auf die Performance analysiert. Bei Apache Spark wird ein Key-Value-Paar mit dem Dateinamen und Inhalt der XML-Datei an das Modul übergeben. Während bei Hadoop Streaming nur der Dateiinhalt an das Modul übergeben wird. Deshalb muss zu Beginn des Parser-Moduls zwischen den beiden Ansätzen unterschieden werden, damit anschließend auf einer einheitlichen Datenbasis weitergearbeitet werden kann.

Python 6.102: Pyxmlparser: Differenzierung Spark und Hadoop Streaming

```
1   if type(doc) is list:
2       ## beginning for spark
3       xmlFileName = doc[0]
4       session_timestamp = xmlFileName[4:23].replace('T',' ').
            replace('_',':')
5   try:
6       xmlString = codecs.encode(doc[1], "ascii", 'ignore')
7       xmldoc = minidom.parseString(xmlString)
8   except Exception as e:
9       if 'errorlog' not in ERROR:
10          ERROR['errorlog'] = list()
11          ERROR["errorlog"].append([xmlFileName, str(e), timestr])
12          # if xmldoc
13          xmldoc.unlink()
14          return ERROR
15      else:
16          # beginning for hadoop streaming
17          xmlFileName = ''
18          session_timestamp = ''
19          try:
20              xmlString = codecs.encode(doc, "ascii", 'ignore')
21              xmldoc = minidom.parseString(xmlString)
22              if xmldoc.getElementsByTagName("pgdash-merger-append
                    -file-name"):
23                  filename = xmldoc.getElementsByTagName("pgdash-
                        merger-append-file-name")[0]
24                  xmlFileName = filename.firstChild.data.encode('
                        utf-8')
25                  session_timestamp = xmlFileName[4:23].replace('T
                        ',' ').replace('_',':')
26          except Exception as e:
27              if 'errorlog' not in ERROR:
28                  ERROR['errorlog'] = list()
29                  ERROR["errorlog"].append([xmlFileName, str(e),
```

```
                        timestr])
30                    # if xmldoc
31                    xmldoc.unlink()'''
32              return ERROR
```

Die einzelnen Bereiche *Tracker, Terminal, Ticket, Path* und *Summary* haben eine eigene Methode, welche teilweise wieder Submethoden enthalten. In den Submethoden werden die Attribute geparst. Dazu wird das zu parsende XML-Tag in einem Document-Element gespeichert. Das Document-Element wird in einer for-Schleife iteriert und die zu parsenden Attribute werden in einer Variable gespeichert. Hierbei sind die zu parsenden Attribute hart im Quellcode benannt. Neue Attribute müssten im Quellcode hinzugefügt werden. Anschließend werden die gespeicherten Variablen einer Liste hinzugefügt.

Python 6.103: Pyxmlparser: Submethode am Beispiel Connectivitytest

```
1   def connectivitytestParser():
2     ## Init VAR for connectivitytest
3     connectivitytest_uuid, connectivitytest_timestamp,
          xtcitest_url, xtcitest_durationInMSec,
          xtcitest_connectivity, httptest_url,
          httptest_durationInMSec, httptest_connectivity = '', '', '
          ', '', '', '', '', ''
4
5     connectivitytest = xmldoc.getElementsByTagName("
          connectivitytest")
6     for x in connectivitytest:
7         connectivitytest_uuid = str(uuid.uuid4())
8         connectivitytest_timestamp = x.getAttribute("timestamp")
9         connectivitytest_timestamp = connectivitytest_timestamp.
              replace("T"," ")
10
11    xtcitest = xmldoc.getElementsByTagName("xtcitest")
12    for x in xtcitest:
13        xtcitest_connectivity = x.getAttribute("connectivity")
14        xtcitest_durationInMSec = x.getAttribute("durationInMSec")
15        xtcitest_url = x.getAttribute("url")
16
17    httptest = xmldoc.getElementsByTagName("httptest")
18    for x in httptest:
19        httptest_connectivity = x.getAttribute("connectivity")
20        httptest_durationInMSec = x.getAttribute("durationInMSec")
21        httptest_url = x.getAttribute("url")
22
23    ## add to output list
```

```
24      connectivitytest_list.append(connectivitytest_uuid)
25      connectivitytest_list.append(session_array[0])
26      connectivitytest_list.append(connectivitytest_timestamp)
27      connectivitytest_list.append(xtcitest_url)
28      connectivitytest_list.append(xtcitest_durationInMSec)
29      connectivitytest_list.append(xtcitest_connectivity)
30      connectivitytest_list.append(httptest_url)
31      connectivitytest_list.append(httptest_durationInMSec)
32      connectivitytest_list.append(httptest_connectivity)
33
34
35      x = connectivitytest_list[:]
36      connectivitytest_list.append(x)
37      del connectivitytest_list[:]
38
39      if 'connectivitytest' not in _dict:
40          _dict['connectivitytest'] = list()
41          _dict["connectivitytest"].append(x)
```

In den Bereichsmethoden werden die Listen mit den geparsten Attributen aus den Submethoden in der Variable *res* zurückgegeben. Die Variable *res* wird dann dem Dictionary hinzugefügt.

Python 6.104: Pyxmlparser: Bereichsmethode am Bespiel Path

```
1   ## Event-Tag
2   if item.nodeName == "event":
3   res = get_event_attributes(item, screenAction_id,
        sequence_counter,session_array[0])
4
5   if 'screen_action' not in _dict:
6     _dict['screen_action'] = list()
7   _dict["screen_action"].append(res)
```

In der Mainmethode werden alle Bereichsmethoden aufgerufen, die das Dictionary befüllen und zum Schluss wird das Dictionary zurückgegeben.

Python 6.105: Pyxmlparser: Bereichsmethode

```
1   ## main function for xmlparser
2   try:
3       ## fill lists
4       trackerParser(session_timestamp, xmlFileName)
5       terminalParser()
```

```
6         ticketParser()
7         pathParser()
8         summaryParser()
9         connectivitytestParser()
10
11        if 'session' not in _dict:
12            _dict['session'] = list()
13        _dict["session"].append(session_array)
14
15        return _dict
16
17        except Exception as e:
18        ## fill errorlist
19        if 'errorlog' not in ERROR:
20            ERROR['errorlog'] = list()
21        ERROR["errorlog"].append([xmlFileName, str(e), timestr])
22
23        return ERROR
```

Probleme

Im Laufe der Entwicklungen hat sich gezeigt, dass die API *xml.minidom* einige Probleme aufweist. Es ist vermehrt zu Schwierigkeiten aufgrund eines erhöhten RAM-Bedarfs gekommen, welches sich durch einen out-of-memory-Fehler ausdrückte. Eine weitere Recherche hat ergeben, dass dieses Problem durch die verwendete API *xml.minidom* ausgelöst wird, da diese das Vielfache der XML-Datei an RAM benötigt. Um das Problem zu lösen, wurde mittels eines Benchmarkings, welches in Kapitel 6.5.4 beschrieben ist, nach Alternativen gesucht worden.

Fazit

Mit der *xml.minidom* API, die vom *Pyxmlparser* benutzt wird, muss sich am Objektbaum der XML-Datei entlang gearbeitet werden, um die richtigen XML-Tags für den richtige Dictionary-Key zu parsen. Um dieses Vorgehen zu realisieren mussten viele For-schleifen und If-Abfragen angewendet werden. Das führte sehr oft zu Komplikationen und die Übersicht im Code war nicht mehr gewährleistet. Schlussendlich konnten die Attribute mit dem Parser geparst werden, welche für die Analyse erforderlich waren. Aber wie in diesem Kapitel beschrieben, führte die API *xml.minidom* zu einem out-of-memory Fehler.

6.5.4. Benchmark: Python XML-Parsing Frameworks

Der Parser, welcher basierend auf dem Python-Framework `minidom` entworfen wurde, brachte bei der Ausführung mit großen Datenmengen einen zu hohen Bedarf des Hauptspeichers mit sich.[2] Eine regelmäßige Anwendung dieses Programmes zum Transformieren der Daten erweist sich daher als problematisch. Eine Neuentwicklung, basierend auf einem Framework, welches den RAM der verarbeitenden Maschine schonte, wurde somit benötigt.

Eine Online-Recherche der Projektgruppe ergab, dass für die Evaluierung der unterschiedlichen XML-Frameworks in Python kein offizielles *Benchmarking* veröffentlicht wurde. Um ein fehlerfreies und absehbares Ergebnis zu entwickeln, wurde in diesem Zuge ein Benchmarking durchgeführt. Evaluiert wurden hierbei die folgenden sieben Frameworks.

1. lxml.etree
2. lxml.objectify
3. xml.dom.minidom
4. xml.dom.pulldom
5. xml.etree.ElementTree
6. xml.etree.cElementTree
7. xml.sax

Um eine Vergleichbarkeit herzustellen wurden diese Bibliotheken, unabhängig von ihrer Vorgehensweise, auf XML-Dateien verschiedener Größen angewendet. Konkret handelte es sich um 15 Dateien zwischen einem Kilobyte und 64 Megabyte Zudem wurde der Test in zwei Abschnitte untergliedert: *Baseline* und *Multitag*. Baseline bedeutet in diesem Fall, dass gemessen wurde, welche Menge an Hauptspeicher ein Framework ausschließlich für das Laden einer Datei benötigte. Beim sogenannten Multitag wurde hingegen gemessen, welche Menge an RAM das jeweilige Framework beim lesen von Daten am Anfang, in der Mitte und am Ende der XML-Datei benötigte. Zusätzlich wurde als relevanter Faktor die Performanz der Frameworks herangezogen. Für die *Baseline*- und *Multitag*-Tests wurde hierfür die Zeit zur Verarbeitung gemessen.

Diese Vorgehensweise verschaffte der Projektgruppe eine nützliche Übersicht über die RAM-Auswirkungen und Performanz der Bibliotheken mit skalierenden Datenmengen und aufwändigen Operationen. Die genauen Ergebnisse der Speicher- und Zeitbedarfe unter den verschiedenen Bedingungen können im Anhang betrachtet werden.[3]

[2] Siehe Kapitel 6.5.3: Minidom Parser-Modul.
[3] Siehe Anhang A.3: Benchmark der XML-Bibliotheken in Python.

Herausstechend und relevant war insbesondere das Framework `xml.sax`. Dieses durchsucht eine XML-Datei in einzelnen Schritten und erzeugt bei konkreten Inhalten verarbeitbare Events. Es werden dabei keine Strukturen aufgebaut und der Hauptspeicher wird dabei nur für die Events genutzt. Der Bedarf an Hauptspeicher-Ressourcen ist daher unabhängig von der Dateigröße beinahe nicht existent.[4] Als Vergleich stieg bei dem Framework `xml.dom.minidom` der Speicherbedarf exponentiell.[5] Die Abbildung 6.13 verdeutlicht dies.

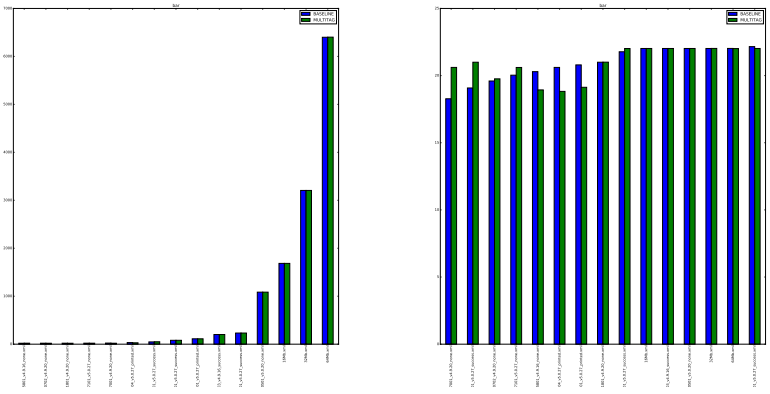


Abbildung 6.13.: Vergleich der Benchmark-Resultate von `dom.minidom` (links) und `xml.sax` (rechts)

Auf der x-Achse werden die unterschiedlich großen Dateien dargestellt, während die y-Achse den Hauptspeicherbedarf darstellt. Die blauen Balken stellen die Ergebnisse für *Baseline* dar, während die grünen *Multiline* aufzeigen. Auf den ersten Blick erscheint der Bedarf von `xml.sax` konstant gleich hoch, wie der maximale Bedarf von `dom.minidom`. Dies liegt jedoch an der unterschiedlichen Wahl der Skala der y-Achse. Im linken Diagramm zeigt die vertikale Achse einen maximalen Speicherbedarf von 7.000 Megabye, während das rechte Diagramm eine Maximalhöhe von 25 Megabyte aufweist.

Zeitlich befanden sich alle Frameworks auf ähnlichem Niveau. Lediglich die Bibliotheken `xml.dom.pulldom` und `xml.etree.ElementTree` weisen einen deutlich erhöhten Zeitaufwand auf.[6]

Auf Basis der Ergebnisse dieses Benchmarking entschied sich die Projektgruppe für eine Nutzung des Frameworks `xml.sax` bei der Entwicklung des neuen Parsers.

[4] Siehe Anhang A.3: RAM-Benchmark des Frameworks xml.sax.
[5] Siehe Anhang A.3: RAM-Benchmark des Frameworks xml.dom.minidom.
[6] Siehe Anhang A.3: Zeit-Baseline-Benchmark: Alle Frameworks mit verschiedenen Dateigrößen.

6.5.5. Parsen in relationales Datenformat: SaPPy

Der *Sax Parser on Python* (SaPPy) ist eine Python-basierte Eigenentwicklung, dessen Funktion das Übersetzen von XML-Daten in ein relationales Format ist. SaPPy basiert auf dem durch Python zur Verfügung gestellten Framework *xml.sax*[7]. Dieses durchläuft XML-Daten *line-by-line* (Zeile für Zeile). Bei jeder gefundenen Information wird hierbei ein Event generiert, welches durch eine konkrete Implementierung verwendet werden kann. Durch dieses Verfahren wird sichergestellt, dass ein niedriger Bedarf an Ressourcen durch den Hauptspeicher benötigt wird.

Die freie Konfigurierbarkeit ist eine weitere besondere Eigenschaft von SaPPy. In einer beiliegenden Konfigurationsdatei können im JSON-Format sowohl die Zielstruktur (Tabellen und Spalten), als auch Speicheranweisungen zu sämtlichen benötigten Informationen der XML-Daten festgelegt werden.

Für die Umsetzung wurde vorerst eine Software-Architektur entworfen, durch welche alle Anforderungen abgedeckt werden können. Diese wird in der Abbildung 6.14 dargestellt.

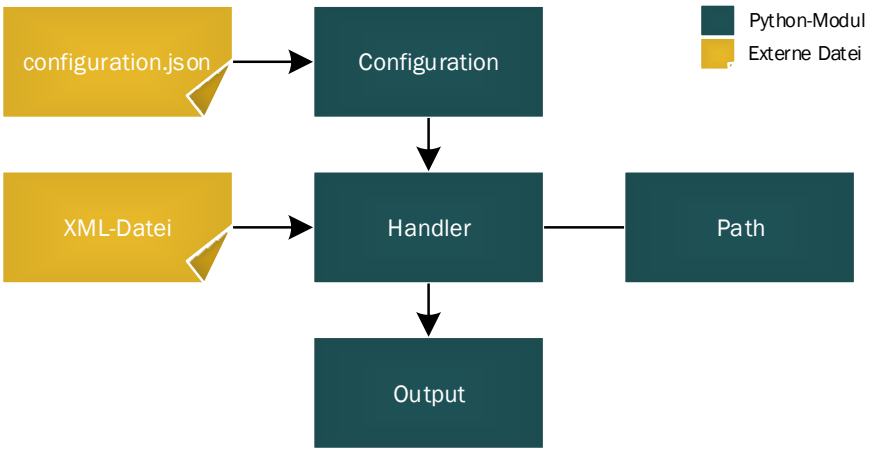


Abbildung 6.14.: Architektur des XML-Parsers „SaPPy“

Die Begriffe der Abbildung werden für das weitere Verständnis folgend erläutert.

configuration.json: Diese Konfigurationsdatei enthält Informationen über die Zieltabellen, die gesuchten Daten innerhalb der XML-Dateien und Anweisungen über die jeweiligen Zuordnungen der Quell- und Zielstruktur.

Configuration: Dieses Modul bildet die Schnittstelle zwischen der Konfigurationsdatei und dem Handler. Mit Hilfe verschiedener Funktionen können In-

[7]Siehe Webseite: `https://docs.python.org/2/library/xml.sax.html`

formationen über Anweisungen der Konfigurationsdatei im Programm abgerufen werden.

XML-Datei Dies ist die zu untersuchende Datei im Quellformat. Beim Programmaufruf wird der Pfad zu dieser Datei übergeben.

Handler: Der Handler ist eine konkrete Implementation des Sax-Frameworks. Bei einem Durchlauf werden für verschiedene XML-Informationen Events generiert, welche mit Hilfe der Anweisungen der Konfiguration verarbeitet und gegebenenfalls in der Output-Struktur hinterlegt werden.

Path: Durch das Sax-Framework werden keine Informationen über die hierarchische Struktur der XML-Daten zur Verfügung gestellt. Die Position eines Tags ist jedoch ebenfalls eine entscheidende Information und wird benötigt. Zu diesem Zweck wird das Path-Modul benötigt, welches den aktuellen Pfad ständig aktualisiert.

Output: Der Output ist ein Modul aus verschiedenen Klassen, welche eine Verwaltung der Daten in einer Datenbank-ähnlichen Form zur Verfügung stellt. Es können Objekte erzeugt werden, welche Tabellen entsprechen und Zeilen-Objekte enthalten. Es können zudem Primärschlüssel-Spalten in der Tabellen definiert werden, für welche pro Zeile automatisch eindeutige Schlüssel generiert werden.

Nach der Festlegung auf die Architektur der Software, fand die Konzeption des Workflows von SaPPy statt. Die einzelnen Schritte des fertigen Endergebnisses werden im Folgenden aufgezählt.

1. *Konfigurationsdatei laden:* Die im JSON-Format hinterlegte Konfigurationsdatei wird durch das Programm geladen und im Hauptspeicher gehalten. Das *Configuration-Modul* bietet als Schnittstelle Funktionen für jeweils benötigte Informationen.

2. *Output Struktur anlegen:* Mit Hilfe der Informationen der Konfigurationsdatei werden Tabellen und Spalten angelegt. Dabei werden den Zieltabellen Spalten für Primärschlüssel zugewiesen, welche SaPPy automatisch mit eigens generierten UUIDs füllt.

3. *Handler erzeugen:* Eine Instanz der eigenen Implementierung eines *Handlers* wird erzeugt. Dieser verarbeitet Events, welche durch in den XML-Daten gefundene Informationen generiert werden. Die Verarbeitung wird hierbei durch den Entwickler festgelegt und nachfolgend detailliert erläutert.

4. *Parsen:* Für jedes gefundene Element in den XML-Daten wird mit Hilfe des *Configuration-Moduls* überprüft, ob und wie eine Information verwendet werden soll. Wird erkannt, dass es sich um eine zu speichernde Informati-

on handelt, so wird diese in der entsprechenden Stelle der *Output-Struktur* hinterlegt. Wenn jedoch ein endendes XML-Tag gefunden wird, dessen Bezeichnung einer Tabelle des *Output-Moduls* entspricht, so wird in dieser Tabelle eine neue Zeile begonnen.

5. *Output ausgeben:* Die erzeugten Output-Inhalte werden für die weitere Verarbeitung in das benötigte Zielformat konvertiert.

Erkennbar wird, dass aufgrund der Architektur und der beabsichtigten dynamischen Abwicklung und Anpassbarkeit die Konfigurationsdatei eine entscheidende Komponente ist. Diese wurde in zwei Abschnitte unterteilt: Ein Block für die Festlegung der *Zielstruktur* und ein Block für die *Anweisungen* für unterschiedliche Tags.

Folgend wird beispielhaft die Konfiguration einer Zieltabelle dargestellt.

JSON 6.106: Konfiguration einer Zieltabelle in „SaPPy“

```
"tableDef_1": [{
  "tableName": "person"
}, {
  "columns": ["uuid",
  "firstname",
  "lastname"
  ]
}, {
  "PRIMARY_KEY": "uuid"
}]
```

In diesem Beispiel wird eine Tabelle mit dem Namen „Person“ angelegt, bei welcher neben verschiedenen Feldern ein Primärschlüssel für das Feld *uuid* angelegt wird. Entscheidend ist für die Erkennung einer Tabellenkonfiguration für SaPPy hierbei, dass der Prefix *„tablDef_“* enthalten ist.

Die Konfiguration für den Block der Anweisungen ist komplexer. Da in den gegebenen XML-Dateien Werte in unterschiedlichen Formaten hinterlegt wurden, mussten diese Formate aufgenommen und separat unterschieden werden, damit diese entsprechend gespeichert werden. Hierbei wird bei den Anweisungen in vier Fälle unterschieden:

1. *Attribut:* Dieser Fall bedeutet, dass eine benötigte Information in einem Attribut eines XML-Tags zu finden ist. Beispiel:

   ```
   <person firstname=“Max“ lastname=“Mustermann“ />
   ```

Zu konfigurieren ist hierbei, um welches Tag und um welches Attribut es sich handelt. Dementsprechend können eine Zieltabelle und -Spalte deklariert werden.

2. *Attribut in Abhängigkeit eines anderen Attributes:* In diesem Fall ist eine Information vorhanden, allerdings ist nur mit Hilfe eines weiteren Attributes erkennbar, um welche Information es sich handelt. Beispiel:

   ```
   <personinfo key="fname" val="Max" />
   <personinfo key="lname" val="Mustermann" />
   ```

   In diesem Fall kann es bedeuten, dass verschiedene Attributinhalte verschiedene Zieltabellen- oder Spalten zur Folge haben.

3. *Inhalt:* Hierbei befindet sich eine Information zwischen einem Start- und Endtag. Beispiel:

   ```
   <fname>Max</fname>
   <lname>Mustermann</lname>
   ```

   Für die Speicherung ist hierbei lediglich das Tag entscheidend.

4. *Inhalt in Abhängigkeit eines Attributes:* Dieser Fall bedeutet, dass eine Information sich zwischen einem Start- und einem Endtag befindet. Jedoch ist erst mit Hilfe eines Attributes erkennbar, um welche Information es sich handelt und in welcher Tabelle und Spalte diese hinterlegt werden sollte. Beispiel:

   ```
   <personinfo key="fname">Max</personinfo>
   <personinfo key="lname">Mustermann</personinfo>
   ```

   In diesem Fall kann es bedeuten, dass verschiedene Attributinhalte verschiedene Zieltabellen- oder Spalten zur Folge haben.

Damit diese vier Fälle korrekt verarbeitet werden können, wurden hierfür Möglichkeiten der jeweiligen Konfiguration implementiert. Wie diese korrekt genutzt werden können wird folgend anhand von Beispielen dargestellt.

Soll in der XML-Datei nach einem Attribut gesucht werden, so kann dies folgendermaßen konfiguriert werden:

JSON 6.107: Konfiguration eines Attributes in „SaPPy“

```
"person": [{
  "firstname": [{
    "valueFrom": "ATTRIBUTE"
  }, {
    "destinationTable": "person"
  }, {
```

```
7       "destinationColumn": "firstname"
8       }]
9     }]
```

In diesem Beispiel wird im Tag `<person>` nach dem Inhalt des Attributs „`firstname`“ gesucht. Wird hier etwas gefunden, so wird der Inhalt in der Spalte „`firstname`“ der Zieltabelle „`person`“ hinterlegt.

Das nächste Beispiel zeigt den Umgang mit Attributen in Abhängigkeit anderer Attribute.

JSON 6.108: Konfiguration eines Attributes in Abhängigkeit eines anderen Attributes in „SaPPy“

```
1     "personinfo": [{
2       "val": [{
3         "valueFrom": "ATTRIBUTE_DEPENDING_ON_ATTRIBUTE"
4       }, {
5         "identificationAttributeName": "key"
6       }, {
7         "cases": [{
8           "fname": [{
9             "destinationTable": "person"
10          }, {
11            "destinationColumn": "firstname"
12          }]
13        }, {
14          "lname": [{
15            "destinationTable": "person"
16          }, {
17            "destinationColumn": "lastname"
18          }]
19        }]
20      }]
21    }]
```

Erkennbar ist, dass hier ein weiterer Zweig geschrieben wurde mit dem Identifier „`cases`“. Beinhaltet das identifizierende Attribut „`key`“ den Wert „`fname`“, so wird der Wert des Attributes „`val`“ in die Spalte „`firstname`“ geschrieben.

Weiterhin gibt es die Möglichkeit, den Inhalt zwischen einem Start- und einem Endtag zu speichern. Dies wird folgendermaßen konfiguriert:

JSON 6.109: Konfiguration von Inhalten in „SaPPy“

```
"fname": [{
  "firstnameContent": [{
    "valueFrom": "INFORMATION_FROM_CONTENT"
  }, {
    "destinationTable": "person"
  }, {
    "destinationColumn": "firstname"
  }]
}]
```

Auf diese Art und Weise wird jedes mal Inhalt in der Spalte „`firstname`“ der Zieltabelle „`person`“ hinterlegt, wenn Inhalt zwischen einem startenden und einem endenen `<fname>`-Tag gefunden wird.

Der letzte Fall ist die Speicherung von Inhalt basierend auf einem Attribut. Dieses wird wie folgt deklariert:

JSON 6.110: Konfiguration von Inhalten in Abhängigkeit von Attributen in „SaPPy“

```
"personinfo" : [{
  "key" : [{
    "valueFrom" : "CONTENT_DEPENDING_ON_ATTRIBUTE"
  }, {
    "cases": [{
      "fname": [{
        "destinationTable": "person"
      }, {
        "destinationColumn": "firstname"
      }]
    }, {
      "lname": [{
        "destinationTable": "person"
      }, {
        "destinationColumn": "lastname"
      }]
    }]
  }]
}]
```

Das identifizierende Attribut ist hierbei „`key`“ des `<personinfo>`-Tags. Der Inhalt wird basierend auf dem Inhalt des Attributs in einer anderen Spalte und Tabelle hinterlegt.

Eine weitere Funktion ist die Möglichkeit der Generierung von Primär- und Fremdschlüsseln. Einige Zeileneinträge sind nur mit Bezug auf eine andere Tabelle sinnvoll. Zu diesem Zweck wurde für die Konfiguration eine Option berücksichtigt. Bei jedem Tag kann ein Zweig mit dem Identifier „`foreignKeys`“ erstellt werden.

JSON 6.111: Konfiguration von Fremdschlüsseln in „SaPPy“

```
"foreignKeys": [{
  "FK_1": [{
    "parent": "abteilung"
  }, {
    "destinationTable": "person"
  }, {
    "destinationColumn": "abteilung"
  }]
}]
```

Da bei jeder neuen Zeile automatisch ein Primärschlüssel generiert wird, ist SaPPy in der Lage, auf diesen zurückzugreifen und in einer anderen Tabelle zu hinterlegen. In diesem Beispiel wird bei einem gefundenen Tag der Primärschlüssel des letzten Eintrags der Tabelle „`abteilung`“ in der Spalte „`abteilung`“ der Tabelle „`person`“ gespeichert. So kann später mit Abfragesprachen wie SQL eine Referenz genutzt werden.

Folgend wird der beschriebene Workflow von SaPPy in Python implementierte dargestellt.

Python 6.112: Python-Workflow von „SaPPy“

```
def parse(xml_file):

  # Loading the Input
  sConfig = SappyConfiguration('configuration.json')

  # Initializing the Output structure which is going to get
      filled
  sOutput = SappyOutput()
  for table in sConfig.getTableNames():
    primaryKey = sConfig.getPrimaryKeyOfTable(table)
    sOutput.addTable(table, sConfig.getColumnsOfTable(table),
        primaryKey)

  # Setting up the Parser
```

```
13      parser = make_parser()
14      sHandler = SappyHandler(sConfig, sOutput)
15      parser.setContentHandler(sHandler)
16
17      # Parsing
18      parser.parse(xml_file)
19
20      # Return the result
21      return sOutput.getAsListOfTupels()
```

Zu erkennen ist hier, dass SaPPy eine XML-Datei übergeben wird. Im ersten Schritt lädt das Programm die Konfigurationsdatei „`configuration.json`“, in welcher alle zum Parsen benötigten Informationen enthalten sind. Im Anschluss wird die Output-Struktur mit Hilfe der Konfigurationsdatei befüllt. Hierfür werden befüllbare Tabellen-Objekte erzeugt. Im darauffolgenden Schritt wird die Implementierung des Sax-Handlers initialisiert und eine Parser-Instanz mit Hilfe des Handlers erzeugt. Nach dem hierauf folgenden Parsen, werden die erhaltenen Daten in einer beliebigen Form zurückgegeben. In diesem Fall als eine Python-Liste, welche pro Eintrag ein Tupel enthält.

6.5.6. Hadoop Streaming Praktische Umsetzung

In diesem Kapitel wird die praktische Umsetzung von Hadoop Streaming beschrieben. Dazu wird zu Beginn grundlegend beschrieben, wie der Hadoop Streaming Job ausgeführt wird. Anschließend wird erläutert, welche Parameter für das Ausführen genutzt werden und worin ihre Aufgabe besteht. Im nächsten Schritt werden die Klassen näher vorgestellt. Abschließend werden aufgetretene Probleme und deren Lösungen aufgezeigt. Zum Schluss folgt ein abschließendes Fazit.

Ausführung

Hadoop Streaming wird mit einem zusätzlichen Python-Programm gestartet, genannt *launcher.py*. In diesem Programm sind die Defaultwerte und Konfigurationen, die für den Start eines Hadoop Streaming Jobs benötigt werden, hinterlegt. Diese können zum Teil mit Hilfe von zusätzlichen Parametern verändert werden. Bei der Ausführung des Launchers, generiert dieser eine Mapperklasse mit dem Dateinamen *mapper_generated.py* und eine Reducerklasse mit dem Dateinamen *reducher_generated.py*. Die Mapperdatei besteht aus dem Parser *XMLParser.py* und der Datei *mapper_mergedxmls_pyxmlparser.py* die den eigentlichen Mapjob ausführt. Die Reducerdatei besteht aus der Datei *reducer_simple.py*. Allerdings wird diese Datei im Streamingbefehl nicht berücksichtigt, da eine Javadatei verwen-

det wird, die zu einem Key-Value-Paar eine Outputdatei erzeugt. Näheres wird dazu im Abschnitt *Javaoutputklasse* erläutert. Anschließend werden die generierten Dateien in das Unixformat konvertiert und somit ausführbar. In der Konsole werden Login-Informationen angezeigt, welche den Nutzer informieren, ob die erläuterten Prozesse erfolgreich waren oder nicht. Da nun die erforderliche Mapper- und Reducerklasse generiert wurde, kann nun der Befehl zum Starten des Hadoop Streaming Jobs ausgeführt werden. Der Launcher führt deshalb zum Schluss den Hadoop Streaming Befehl aus, wodurch der Hadoop Streaming Job gestartet wird.

Parameter

Wenn der Launcher aufgerufen wird, dann ist der Nutzer in der Lage mit verschiedenen Parametern bestimmte Aktionen auszuführen. Mit dem Befehl ‚*launcher.py –"Parameter"*' können diese Aktionen ausgeführt werden. In der folgenden Tabelle werden die Parameter und ihre jeweiligen Funktionen erläutert.

Parameter	**Funktion**
processmergedxmls	Mit diesem Parameter können zusammengefügte XML-Dateien geparst werden.
generateonly	Wenn dieser Parameter der Launcherdatei übergeben wird, dann wird nur der Hadoop Streaming Befehl in der Konsole ausgegeben und nicht sofort ausgeführt.
runlocal	Dieser Befehl führt den Hadoop Streaming Job lokal aus. Dadurch wird der generierten Mapperdatei *mapper_generated.py* eine Inputdatei, die eine valide XML sein muss, übergeben. Die Printbefehle aus der Mapperdatei werden auf der Konsole ausgegeben, bis die XML vollständig geparst wurde.
pathincrement	Hier kann der gewünschte Outputpfad angegeben werden. Der Standardwert gibt die aktuelle Uhrzeit und eine eindeutige ID zum Zeitpunkt der Ausführung an.
user	Hier kann der Inputordner ausgewählt werden. Der Ordner muss aber auch vorhanden sein und der Usernamen muss mit dem Ordnernamen übereinstimmen.
parser	Durch den Parser Parameter kann der Nutzer entscheiden, welcher Parser verwendet werden soll.

mode	Hier kann entschieden werden, ob ein Spark oder ein Hadoop Streaming Job ausgeführt werden soll.
singlexml	Mit diesem Parameter können mit Hadoop Streaming einzelne XML geparst werden.

Tabelle 6.3.: Parameter des launcher-Skriptes

Mapperklasse

In diesem Abschnitt wird die *Mapperklasse* näher erläutert. Wie bereits erwähnt besteht die durch den Launcher generierte Mapperdatei aus dem Parser und der eigentlichen Mapperklasse. Die Main-Klasse *def_mapper()* liest zu Beginn die Inputdatei und speichert deren Inhalt in einem String.

Folgend wird das Einlesen der Inputdatei dargestellt.

Python 6.113: Hinzufügen des XML-Input

```
1   ## adding line for line to xmlstring
2   for line in sys.stdin:
3     line = line.strip()
4     xml += line  #+ '\n'
```

Anschließend wird in einem verschachtelten Try-Catch-Block eine Integervariable für die maximale Dateigröße initiiert. Im nächsten Schritt werden in der Funktion *remove_bad_symbols()* alle Symbole entfernt, die nicht in dem erlaubten Set an Symbolen gehören. Dadurch wird vermieden, dass die am Parser übergebene XML-Datei ungültig ist. Danach wird die gemergte XML-Datei in einzelne XML gesplittet. Sollten in der XML mehrere XML-Prefixes stehen, dann wäre die XML ebenfalls ungültig. Aus diesem Grund werden die XML-Prefixes gelöscht.

Nachfolgend wird ein Aufteilen der gemergten XML verdeutlicht.

Python 6.114: Aufteilen einer gemergten XML-Datei

```
1   try:
2     MAX_SIZE_BYTES = 10 * 1024 * 1024
3     xml_ready = xml
4     xml_ready = remove_bad_symbols(xml_ready)
5     splits = xml_ready.split(xml_prefix)
6     len_closing_tag = len('</digifoto>')
7     xml_parsed_counter = 0
```

```
8
9     for value in splits:
10      if value.startswith('<digifoto'):
11      xml_single = value
12      index = 0
13      index = xml_single.find('</digifoto>', index)
14
15    if index != -1 and index != 0:
16      ## making a clear xml for output
17      clear_xml_single = xml_single.replace(
18      '<?xml version="1.0" encoding="utf-8"?>', '')
19      clear_xml_single = xml_prefix + clear_xml_single[:index +
20      len_closing_tag]
21      timestamp_filename_array = get_filename(clear_xml_single)
22      xml_file_name = timestamp_filename_array[0]
23      timestamp = timestamp_filename_array[1]
24    else:
25      print('{0}\t{1}{2}'.format("exception", unique_key,
26      ' digifoto was not found'))
27
28    ## increment parsed_counter and set with_exception to false for
          logger
29    with_exception = False
30    xml_parsed_counter += 1
```

Die *clear_xml_single* Variable enthält eine XML aus der gemergten XML. Damit die *clear_xml_single* vom Parser geparst werden kann, muss ein XML-Prefix vor dem ersten Tag der XML gesetzt werden. Im nächsten Schritt werden Dateiname und Zeitstempel geparst. Dazu wird aber nicht der Parser aufgerufen, sondern eine zusätzliche Methode im Mapper, die die *clear_xml_single* Variable übergeben bekommt. In dieser Methode werden Dateiname und Zeitstempel geparst. Wenn kein `digifoto` gefunden wird, dann entsteht eine Exception, die in der Output-CSV Exception geprintet wird. Für den Logger wird *xml_parsed_counter* inkrementiert und die Variable *with_exception* auf False gesetzt. Im nächsten verschachtelten Try-Catch-Block wird überprüft, ob die XML kleiner ist als die maximal erlaubte Dateigröße. Ist das der Fall, dann wird die *clear_xml_single* dem Parser übergeben. Sollte der Parser keine Exception auslösen, dann liefert der Parser ein Python Dictionary. Dieses Dictionary wird dann an die Outputmethode *output_to_reducer* übergeben. Wenn die XML zu groß ist, dann wird eine Exception geprintet. Sollte der Try-Catch-Block in die Exception springen, dann wird ebenfalls eine Exception geprintet. Es wird nun gezeigt, wie die *clear_xml_single* Variable dem Parser übergeben und das Dictionary an die Outputmethode übergeben wird.

Python 6.115: XML-Output

```
## check xmlsize if ok then proceed with output
if len(xml_single) < MAX_SIZE_BYTES:
  xml_as_dict = parser(clear_xml_single)
  output_to_reducer(xml_as_dict, delimeter)
else:
  with_exception = True
  msg = ' estimated size of memory footprint of XML DOM is too
      big, size of the single XML file was bigger then ' + str(
      MAX_SIZE_BYTES) + ' bytes and was ' + str(len(xml_single))

print('{0}\t{1}{2}'.format('exception', unique_key,msg))

except Exception as ex:
  with_exception = True
  print('{0}\t{1}{2}'.format('exception', unique_key, str(ex)))
```

In der Outputmethode werden innerhalb einer for-Schleife die Key-Value-Paare des Dictionary durchgegangen und geprintet. Zusätzlich wird noch eine Logdatei erzeugt, die in der Methode *write_xml_processing_log_to_hdfs* geprintet wird. In der Logdatei wird die aktuelle Version, die eindeutige uuid, der XML-Dateiname und der Zeitstempel geprintet. Hinzu kommt, dass in der Logdatei angegeben wird, ob die XML eine Exception erzeugt hat und um die wievielte XML es sich in der gemergten XML handelt. Wie die Outputmethode die Key-Value-Paare printet, wird im nachfolgenden Code gezeigt.

Python 6.116: Output -und Logmethode

```
def output_to_reducer(xml_as_dict, delimeter):
  """ Proper output for reducer"""

  ## only iterate csv dict keys for output
  csvKeys = ['errorlog','session','screen_action','printjob','
      printerstatus','focused_product','device','dir','product',
      'raw_image','edit_image','video_list','connectivitytest']
  csv_output = ""
  for key in csvKeys:
    if xml_as_dict[key] is not None:
      for sublist in xml_as_dict[key]:
        if sublist is not None:
          csv_output += string.join(
            (str(x) for x in sublist), (delimeter))
          print('{0}\t{1}'.format(key, delimeter + csv_output))
```

```
14              csv_output = ""
15            else:
16              logging.debug('[i] value ' + key +
17                      ' was None and was not processed')
18        else:
19          logging.debug('[i] value ' + key +
20                  ' was None and was not processed')
21
22    def write_xml_processing_log_to_hdfs(unique_key='NoValue',
23                        counter='NoValue',
24                        xml_file_name='NoValue',
25                        timestamp='NoValue',
26                        with_exception=False):
27      """ Writing special CSV file in HDFS  with logs related to XML
            processing"""
28
29      delimeter = '\001'
30
31      ## checking parser version
32      if __version__ is None:
33        curren_version = 'NoValue'
34      else:
35        curren_version = __version__
36
37      ## logger output
38      msg = '"UNIQUE_KEY:{unique_key}"{delimeter}"PARSER_VERSION:{
            parser_version}"{delimeter}"XML_FILE_NAME:{xml_file_name
            }"{delimeter}"TIMESTAMP:{timestamp}"{delimeter}"COUNTER:{
            counter}"{delimeter}"WITH_EXCEPTION:{with_exception}"\
39            '.format(delimeter=delimeter,
40              unique_key=unique_key,
41              parser_version=curren_version,
42              xml_file_name=xml_file_name,
43              timestamp=timestamp,
44              counter=counter,
45              with_exception=with_exception)
46
47      print('{0}\t{1}'.format('logs', msg))
```

Diese Outputmethode wird aber nur für den *Pyxmlparser* verwendet. Für den Parser *SaPPy* wird die Methode *parse_with_sappy()* verwendet. Als erstes wird der Input, der in der *configuration_session.json* definiert ist, geladen. Im nächsten Schritt wird in einer For-Schleife die Outputstruktur initialisiert, die mit den Werten der Attributen aus der XML im Parsingprozess befüllt werden muss. Der Parsingprozess findet in der Methode *parseString()* statt. Im letzten Schritt wird für den Ha-

doop Streaming Output der Output als Key-Value-Paare geprintet. In dem folgenden Code wird gezeigt, wie mit *SaPPy* der Output als Key-Value-Paare geprintet wird.

Python 6.117: XML-Output-SaPPy

```
1  def parse_with_sappy(xml_file):
2  """ Parsing given XML file with SaPPy parser.
3
4  Args:
5  xml_file: content of the given XML
6  """
7
8  # Loading the Input
9  sappyConfiguration = SappyConfiguration('configuration_session.
       json')
10 tableMetaData = sappyConfiguration.getDestinationTableMetaData()
11
12 # Initializing the Output structure which is going to get filled
13 sappyOutput = SappyOutput()
14 for table in sappyConfiguration.getTableNames():
15   primaryKey = sappyConfiguration.getPrimaryKeyOfTable(table)
16   sappyOutput.addTable(table, sappyConfiguration.
         getColumnsOfTable(table), primaryKey)
17
18 # Setting up the Parser
19 #logging.info('Setting up the Parser Object')
20 #parser = make_parser()
21 sappy_handler = SappyHandler(sappyConfiguration, sappyOutput)
22 #parser.setContentHandler(handler)
23
24 # Parsing
25 #logging.info('Handler was created and set')
26 #logging.info('Parsing')
27 parseString(xml_file, sappy_handler)
28 #print (xml_file)
29
30 #sappyOutput.getTable('screen_action').printAll()
31 sappyOutput.printAllAsKeyValue(escapeNewLines=True)
```

Javaoutputklasse

Hadoop Streaming generiert seine Output-Dateien standardmäßig mit dem Dateinamen *part-r-00001*. Sollten mehrere Outputdateien im Ordner erzeugt werden,

wird fortlaufend nummeriert. Die Projektgruppe benötigt aber CSV-Dateien mit einem eindeutigen Namen, die mit dem Tabellennamen aus dem Datenbankschema übereinstimmen. Deshalb musste eine Javaoutputklasse erstellt werden, die mit dem dazugehörigen Tabellennamen eine CSV-Datei erstellt.

Java 6.118: Javaoutputklasse

```
1  package de.pgdash.parser.custom;
2
3  import org.apache.hadoop.fs.Path;
4  import org.apache.hadoop.io.Text;
5  import org.apache.hadoop.mapred.lib.MultipleTextOutputFormat;
6
7  /**
8  *
9  * @Projectname:
10 * @Date: 19.09.2016
11 * @Author: Kevin Aland
12 * <p>
13 * Java-Program, which customize the MultipleTextOutputFormat for
       generating CSV-Files with Keyname.
14 * <p>
15 */
16
17 public class CustomMultiOutputFormat
18 extends MultipleTextOutputFormat<Text, Text> {
19
20   /**
21    * Generate the file output file name based on the given key
         and the leaf file name.
22    * The default behavior is that the file name does not depend
         on the key. It was customizing so it's generate CSV-Files
         with Keyname.
23    * @param key the key of the output data
24    * @param value the value of the output data.
25    * @param name the leaf file name.
26    * @return String Filename
27    */
28   @Override
29   protected String generateFileNameForKeyValue(Text key, Text
         value, String name) {
30
31     String session = key.toString().split("\t")[0];
32     return session + ".csv";
33   }
34 }
```

Anhand der Keys, die in der *mapper_generated.py* geprintet werden, kann die CSV-Datei mit dem Keynamen generiert werden. Damit die Klasse in Hadoop Streaming benutzt werden kann, müssen ein paar Schritte vorgenommen werden.

1. Das *JAVA_HOME* Verzeichnis muss gesetzt werden. Das Verzeichnis kann mit dem Befehl export *JAVA_HOME=/usr/jdk64/jdk1.7.0_67* gesetzt werden.
2. Im nächsten Schritt muss die Javaklasse im Hadoopumfeld kompiliert werden. Der Befehl lautet
   textitJAVA_/bin/javac -cp (hadoop classpath) -d . CustomMultiOutputFormat.java.
3. Im letzten Schritt muss aus der compilierte Klasse ein Jarfile gemacht werden. Mit diesem Befehl kann aus der kompilierten Klasse ein Jarfile gemacht werden *JAVA_HOME/bin/jar cvf parsercustom.jar de/pgdash/parser/custom/CustomMultiOutputFormat.class*.

Im Hadoop Streaming Befehl muss die compilierte Javaklasse und das Jarfile angegeben werden. Mit dem Befehl *-libjars parsercustom.jar -outputformat de.pgdash.parser.custom.CustomMultiOutputFormat* kann dies durchgeführt werden.

Probleme und Lösungen

Hadoop Streaming generiert seinen Output mit einfachen Printbefehlen. Als Ergebnis, aus dem Parsingvorgang, wird für den nächsten Prozess eine CSV-Datei mit einem eindeutigen Namen benötigt. Deshalb mussten die richtigen Werte für die CSV-Datei generiert werden. Der Parser musste deshalb Werte zurückliefern, die eindeutig einen Key zugeordnet werden können. Um dieses Problem zu lösen, hat die Projektgruppe sich entschlossen, dass Pythondictionary als Rückgabewert zu nutzen. Das Pythondictionary kann Key-Value-Paare abbilden und ist deshalb als Rückgabewert für den Parser geeignet. Das nächste Problem ergab sich aus der Kombination des Parsers und der Mapperklasse im Hadoop Streaming Job. Das Aufrufen der Mapperklasse und des Parsers im Hadoop Streaming Job funktionierte nicht. Dieses Vorhaben konnte nicht umgesetzt werden, da der Parser nicht an alle Nodes verteilt wurde. Um dieses Problem zu lösen, wurde der erste Schritt in Richtung Launcher Entwicklung gemacht. Der Launcher kombinierte Parser und Mapperklasse in eine Datei. Dadurch konnte der Hadoop Streaming Job ausgeführt werden. Der Hadoop Streaming Output sorgte ebenfalls für Probleme. Dieser generierte immer standardmäßig eine Datei, die nicht für den ELT-Prozess genutzt werden konnte. Das Tool HIMP, das für den Import von CSV-Daten in Hive-Tabellen zuständig ist, brauchte eine CSV-Datei mit dem

gleichen Dateiname wie der Tabellenname in der die CSV-Datei importiert werden soll. Die Javaoutputklasse konnte dieses Problem lösen und CSV-Dateien mit dem passenden Dateinamen generieren. Beim Parsen der XML-Dateien bei großen Datenmengen brach der Hadoop Streaming Job ab. Die Fehlermeldung lautete Out of Memory. Danach wurde ausführlich überprüft, wie viel Speicher der Minidomparser für das Parsen der XML verwendet. Das Resultat dieser Überprüfung war, dass der Minidomparser zu viel Speicher blockiert und nicht frei gibt. Das Problem lag aber nicht direkt bei Hadoop Streaming, sondern beim Parser. Deshalb musste die Projektgruppe testen, ob der Parser mit einer anderen Parser API weniger Speicher blockiert.

Fazit

Hadoop Streaming konnte in Kombination mit dem SAX Parser SaPPy, eine XML-Datenmenge von 15 GB in ca. 2 Stunden auf dem Alpha-Cluster parsen. Es tauchten einige Probleme, wie in dem Abschnitt 6.5.6 beschrieben, auf. Aber diese Hürden konnten schnell überwunden werden. Durch die simple Verarbeitung mit der Standardeingabe, Standardausgabe und der Aufteilung in Key/Value-Paaren konnte schnell ein Lösung gefunden werden.

6.5.7. Spark

In diesem Kapitel wird beschrieben, wie Apache Spark innerhalb des ELT-Prozesses zum Parsen der XML-Dateien benutzt wird. Zudem werden Aufgaben und Funktionsweise des Spark-Programmes beschrieben. Die während der Entwicklung aufgetretenen Probleme und die entsprechenden Lösungen werden ebenfalls dargestellt. Zum Schluss wird beschrieben, wie das Spark-Programm ausgeführt wird und ein Fazit gezogen.

Einsatz Spark im ELT-Workflow

Die Abbildung 6.15 zeigt einen Ausschnitt des ELT-Prozesses und wie Spark in den ELT-Prozess integriert ist. Dabei hat das Spark-Programm die Aufgabe die XML-Dateien aus dem *ready2parse* Ordner zu laden. Die XML-Dateien können einzelne oder mehrere gültige XML-Dateien enthalten. Befinden sich mehrere XML-Dateien in einer Datei, müssen diese in einzelne XML-Datei Strings gesplittet werden. Zu den XML-Dateien wird SAPPY gebracht. Dabei erwartet SaPPy als Input eine einzelne XML-Datei als String, weswegen viele konkatenierte XML in einer Datei gesplittet werden müssen. Nachdem die einzelnen XML-Strings geparst

worden sind, werden die Ergebnisse in CSV-Dateien in den Ordner *parsedData* gespeichert.

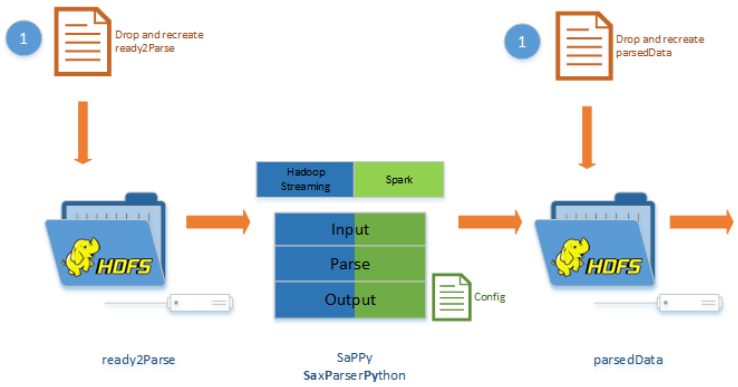


Abbildung 6.15.: Spark im ELT-Workflow

Spark Launcher Skript

Wie im Hadoop Streaming Kapitel 6.5.6 beschrieben, ist ein Launcher-Skript programmiert worden, welches die entsprechenden Skripte für Hadoop Streaming und Spark zur Ausführung erstellt. In Tabelle 6.3 sind die Parameter des Launcher Skriptes beschrieben. Im folgenden wird das vom Launcher generierte Skript für das Parsen der XML-Dateien mit Spark und SAPPY beschrieben.

Der Python Code Abschnitt 6.28 zeigt, dass über Parameter der HDFS Präfix für das Alpha oder das Beta Cluster angegeben werden kann. Dies ermöglicht die einfache Migration und Konfiguration des Skriptes vom Alpha zum Beta Cluster.

Python 6.119: pySpark Launcher: HDFS_PREFIX einsetzten

```
1
2   HDFS_PREFIX_LOCAL = 'file:///'
3   HDFS_PREFIX_ALPHA = 'hdfs://grouphadoopmaster.local:8020'
4   HDFS_PREFIX_BETA  = ''
5
6   def pyspark_sappy(sc, input_path, output_path, cluster):
7
8   ## check cluster type and assign HDFS_PREFIX or raise an
        Exception when no cluster defined
9   hdfs_prefix = None
10
11  if (cluster == 'LOCAL'):
12  hdfs_prefix = HDFS_PREFIX_LOCAL
13
```

```
14    if (cluster == 'ALPHA'):
15    hdfs_prefix = HDFS_PREFIX_ALPHA
16
17    if (cluster == 'BETA'):
18    hdfs_prefix = HDFS_PREFIX_BETA
19
20    if hdfs_prefix == None:
21    raise Exception('You need to specify type of cluster.')
22
23    ## print out cluster type
24    print ('[i] start cluster "{0}"'.format(cluster))
25
26    ## concatenate HDFS_Prefix with the given input path to have
          complete input and output paths
27    hdfs_input_path = '{0}{1}'.format(hdfs_prefix, input_path)
28    hdfs_output_path = '{0}{1}'.format(hdfs_prefix, output_path)
```

Der aufgeführte Code zeigt die Funktion *xmlFile*, die für das splitten von mehreren XMLs in einer Datei zuständig ist und die erzeugten RDDs zurück gibt. An dieser Stelle wird durch die newAPIHadoopFile-Funktion jede Datei in eine Partition im RDD verwaltet. Dies ist ein Kriterium für die hohe Parallelisierung auf dem Cluster, da die Partitionen von den Executorn parallel bearbeitet werden können.

Python 6.120: pySpark Launcher: Hadoop File API Methode Lesen der gemergten Dateien

```
1    def xmlFile(sc, path):
2    """ Read files inside a folder based on Hadoop File API with </
         pgdash-merger-append-file> as a delimiter ."""
3
4    return sc.newAPIHadoopFile(path, "org.apache.hadoop.mapreduce.
         lib.input.TextInputFormat","org.apache.hadoop.io.
         LongWritable"\
5    ,"org.apache.hadoop.io.Text",conf={"textinputformat.record.
         delimiter": '</pgdash-merger-append-file>'}).map(lambda
         num_line:num_line[1])
```

In diesem Code Block ist der Aufruf der Funktion *xmlFile* zu sehen. Zusätzlich werden Informationen wie Anzahl der Partition und Anzahl der XML-Files auf die Konsole ausgegeben.

Python 6.121: pySpark Launcher: Dateien lesen, Zahl der gemergeten und einzelnen Dateien drucken

```
1  ## Read input path based on Hadoop File API, and clean every
       single XML-Text
2  initRDD = xmlFile(sc, hdfs_input_path).map(clean_singles)
3
4  ## get number of XML single files
5  num_of_sXMLs = initRDD.count()
6
7  print "### Number of merged XML Files ###"
8  print initRDD.getNumPartitions()
9
10 print "### Number of single XML Files ###"
11 print str(num_of_sXMLs)
```

In diesem Code Block ist der Aufruf der Map-Phase mit dem Parsen der XML Dateien (Zeile 13) zu sehen. Dies ist der spezielle Umgang mit Spark, wie komplexe Berechnung wie das Parsen von XML zu den Daten gebracht werden können. Das Zwischenspeichern der geparsten Daten auf die Festplatte ist in Zeile 16 dargestellt.

Python 6.122: pySpark Launcher: Parsen und auf der Festplatte persistieren

```
1  def parse_and_gc(x):
2
3  xmlString = remove_bad_symbols(x)
4
5  try:
6  res = parse_with_sappy(xmlString)
7  gc.collect()
8  return res
9  except Exception as ex:
10 return ('errorlog', "file can't be parsed\001" + str(ex))
11
12 ## parse single xml texts using SAPPY
13 parsedRDD = initRDD.map(parse_and_gc)
14
15 ## persist the results in Disk, to avoid re-calculating them
       more than one time
16 persRDD = flattedRDD.persist(pyspark.StorageLevel.DISK_ONLY)
```

In dem hier aufgeführten Code-Abschnitt werden die Key-Value Paare gefiltert und für jeden Key, der dem Tabellennamen entspricht, werden die Daten im HDFS

gespeichert.

Python 6.123: pySpark Launcher: Ergebnisse im HDFS als CSV-Dateien speichern

```
1   ## get a list of distinct keys from the (k,v) pairs, keys are
        the table names
2   csvKeys = persRDD.keys().distinct().collect()
3
4   ## loop over the parsedRDD and filter out the results according
        to the csvKey which implements the CSV file name
5   for csvKey in csvKeys:
6   persRDD.filter(lambda (k,v) : k == csvKey ).map(lambda (k,v) : v
        ).saveAsTextFile( hdfs_output_path + "/" + str(csvKey) )
```

In diesem Code Block ist dargestellt, dass alle generierten Dateien zu einem Key, in eine CSV-Datei integriert und abgespeichert werden mit dem Namen der Tabelle. Dies muss gemacht werden, da mehrere Dateien - für jede Partition eine Datei - beim Speichern im HDFS entstehen.

Python 6.124: pySpark Launcher: Text-Dateien in jedem Ordner in einem CSV sammeln

```
1   ## loop over every output folder to combine all files in one
        file and store them directly under the given output folder
2   for csvKey in csvKeys:
3   sourceName = ( hdfs_output_path + "/" + str(csvKey) + "/part-*"
        )
4   destName   = ( hdfs_output_path + "/" + str(csvKey) + '.csv')
5
6   os.system( "hadoop fs -cat {0} | hadoop fs -put - {1}".format(
        sourceName, destName))
7   os.system( "hadoop fs -rm -r -skipTrash -f {0}".format(
        hdfs_output_path + "/" + str(csvKey)))
```

In diesem Code Block sind die Alpha- und Beta Cluster spezifischen Einstellungen im SparkContext für Spark Driver und Executor zu sehen. Zudem ist in Zeile eins zu sehen, dass Input und Ouput Pfad und der Anwendungsname (AppName) übergeben werden. Diese Einstellungen bestimmen maßgeblich die Performance, da an dieser Stelle definiert wird mit welchen physischen Ressourcen des Clusters die Spark-Anwendung zum Parsen der XML-Dateien ausgeführt werden soll.

Python 6.125: pySpark Launcher: SparkContext Konfigurationen für Local, Alpha, und Beta Modes

```
def launch_pyspark(appName, input_path, output_path, cluster):
if cluster == None:
raise Exception('You need to specify type of cluster.')

if cluster == 'LOCAL':
conf = (SparkConf()
.setMaster("local")
.setAppName(appName)
.set("spark.cores.max", "2")
)

if cluster == 'ALPHA':
conf = (SparkConf()
.setMaster("yarn-client")
.setAppName(appName)
.set("spark.driver.memory", "3g")
.set("spark.driver.cores", "1")
.set("spark.executor.cores", "1")
.set("spark.executor.instances", "5")
.set("spark.executor.memory", "3g")
)

if cluster == 'BETA':
conf = (SparkConf()
.setMaster("yarn-client")
.setAppName(appName)
.set("spark.driver.memory", "20g")
.set("spark.driver.cores", "4")
.set("spark.executor.cores", "1")
.set("spark.executor.instances", "38")
.set("spark.executor.memory", "5g")
)

sc = SparkContext(conf = conf)

pyspark_sappy(sc, input_path, output_path, cluster)
sc.stop()
```

Entwicklung mit Spark

Während der Entwicklung und Test -Phase, wurden die Spark-Shell, Spark-UI, und Spark-Submit genutzt. In diesem Abschnitt werden diese Themen erläutert.

Spark shell

Die Spark-Shell ist eine interaktive Shell, um Spark Befehle und Programme auszuführen. Mit dieser Möglichkeit können Spark Funktionen und kleine Programme entwickelt, getestet und erforscht werden. Die Ergebnisse der Befehle oder Programme werden direkt in der Konsole ausgegeben. Die Spark-Shell unterstützt die Programmiersprachen Python und Scala.

Die Spark-Shell für Python wird mit dem Befehl „pyspark“ gestartet. Die Abbildung 6.16 zeigt wie die pyspark-Shell aussieht. In der Shell werden die Versionen von Spark und Python gezeigt.

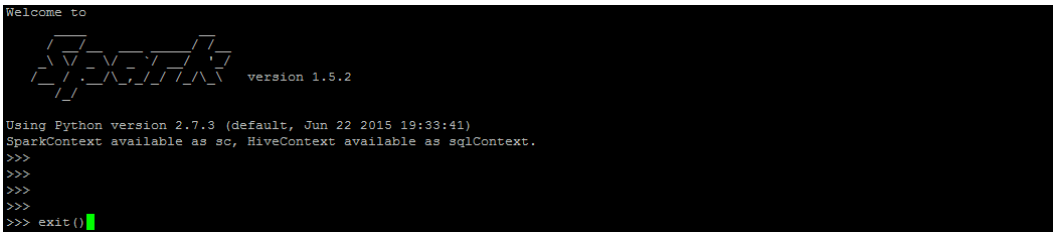


Abbildung 6.16.: Spark-Shell

Spark-UI

Die Spark-UI ist eine Oberfläche, um die Spark Programme zu überwachen. In der Spark UI können Startzeit und Endzeit und somit die gesamte Laufzeit des Programmes nachgeschaut werden. Die Jobs, die Stages und die Tasks können für das Spark Programm angeschaut werden und im Fehlerfall kann in die Logs des Programms geschaut werden. Zudem werden für den Ablauf des Programmes die verwendeten Ressourcen protokolliert. Die Spark UI hilft beim Entwickeln von Spark Programmen, um diese zu verstehen und zu optimieren. (siehe Abbildung 6.17 im Hadoop Cluster unter Standardport „http://localhost:18080“ aufzurufen)

Die in Abbildung 6.17 zu sehenden Tabs der Oberfläche werden im folgenden inhaltlich kurz beschrieben:

Jobs: Hier werden alle Jobs der Anwendung aufgelistet, mit Informationen über Ausführung des Jobs sowie Zeit, Dauer, Input und Output Größe des Jobs.

Stages: Hier werden alle Stages der Anwendung aufgelistet, mit den Informationen über die Ausführung der Stage.

Executors: Hier werden alle Executor mit ihren Cluster-Node Adressen und der von ihnen verarbeiteten Tasks angezeigt.

spark-submit

Spark-Submit wird genutzt, um die Spark-Anwendungen im Spark Cluster auszuführen. Über die Shell können Spark-Programme mit dem Befehl spark-submit

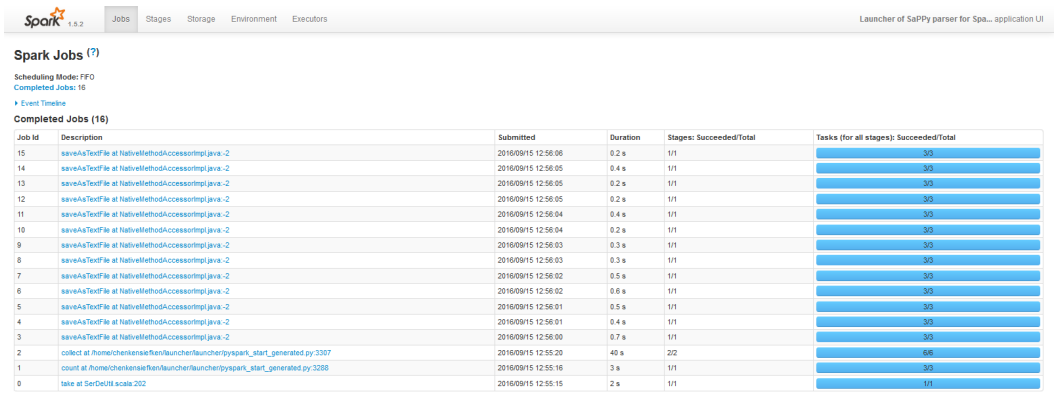


Abbildung 6.17.: Spark-WebUI

ausgeführt werden. Dabei werden mit dem spark-submit Befehle weitere Parameter für die Executor und den Driver wie Memory und Anzahl der Cores übergeben.

OutOfMemory Problem während der Entwicklung

In diesem Abschnitt wird das OutOfMemory Problem beschrieben, welches in der Projektgruppe während der Entwicklung mit Spark häufig aufgetreten ist. Das „OutOfMemory“ Problem taucht auf, wenn ein oder mehrere Executor keinen ausreichenden RAM mehr zur Verfügung haben. Die Ausführung der Anwendung wird bei diesem Problem gestoppt, und mit der Fehlermeldung „Executor Lost“ beendet.

Ursachen des OutOfMemory Problems in der Umsetzung:

- Die Nutzung der „cache()“ Funktion, die die Daten im RAM zwischenspeichert, wenn die Menge der zwischenzuspeichernden Daten größer als der verfügbare RAM ist.
- Die Minidom-Bibliothek für das Parsen der XML-Dateien erzeugte ebenfalls OutOfMemory Probleme. Die Minidom-Bibliothek hat ein RAM Problem bei großen XML-Dateien erzeugt, die Minidom-Bibliothek exponentiell viel RAM reserviert für das Parsen einer XML-Datei.

Um das OutOfMemory Problem zu vermeiden, ist es wichtig bei der Programmierung des Spark Programms an die physischen Ressourcen des Clusters zu denken, die SparkConf entsprechend den Ressourcen zu konfigurieren und das Spark Programm ressourcenschonend zu programmieren.

Logging

Ein wichtiger Bestandteil der Spark-Anwendung ist das integrierte Logging. Logs sind wichtig für das Debuggen im Fehlerfall und zur Optimierung der Spark Programme. Spark mit Yarn hat eine Möglichkeit des Loggings, die mit folgendem Befehl aufgerufen werden kann.

Bash 6.126: Yarn Logs

```
yarn logs –applicationId <application ID>
```

Jede Spark-Anwendung wird automatisch mit der von Spark benutzten Standardbibliothek geloggt. Die Logs werden in der Shell ausgegeben und gleichzeitig in einer Log-Datei gespeichert. Jeder Zeit kann der Nutzer die Logs der Anwendung aufrufen, um die Anwendung zu debuggen.

Ausführung des Parsers mit Spark

Um die Daten mit dem SaPPy Parser und Spark zu Parsen muss zunächst das Python Skript zum Ausführen generiert werden. Dies geschieht mit Hilfe des Launcher Skriptes und dem dargestellten Befehl in Code Block 6.37.

Bash 6.127: Launcher Aufruf Beispiel

```
python launcher.py  ––mode SPARK \
––parser SAPPY \
––generateonly \
––input ready2Parse ––output parsedData
```

Das durch den Launcher generierte Skript zum Ausführen mit Spark kann, wie in dem aufgeführten Code Block dargestellt, ausgeführt werden.

Bash 6.128: Spark Programm starten

```
spark–submit  pyspark_start_generated.py \
––input ready2Parse \
––output parsedData \
––cluster BETA
```

Fazit

Spark ist ein mächtiges In-Memory Tool im Hadoop Umfeld. Die hohe Performance durch die In-Memory Lösung birgt gleichzeitig auch Komplexität und Risiken durch die geschilderten OutOfMemory Probleme. Für die Aufgabe zum XML-Parsen konnte Spark erfolgreich und performant im Projekt eingesetzt werden.

6.5.8. Hive-Datenimport: HIMP

Das Programm *HIMP* (kurz für **Hi**ve **Imp**orter) ist eine java-basierte Eigenentwicklung, dessen Aufgabe das Importieren von CSV-Dateien in Hive-Tabellen ist.

Nachdem die XML-Dateien durch den *XMLParser* (siehe Kapitel 6.5.2) in CSV-Dateien geparst und in einem Verzeichnis des HDFS abgelegt wurden, sollen diese Dateien durch Hive aufgenommen werden. Um Modularität zu wahren, wird dies nicht durch den XMLParser selbst übernommen, sondern in einem anderen dynamischeren und unabhängigen Prozess durchgeführt. Eine Importierung nach jeder einzelnen Übersetzung des XML-Parsers hätte zur Folge, dass eine große Menge an Prozessen gestartet würde, welche die kleinen Dateien jeweils einzeln in die Hive-Tabellen übertragen. Dies entspricht nicht dem Prinzip von Apache Hadoop und MapReduce, dessen Vorteil unter anderem die auf einem Computercluster verteilte Verarbeitung großer Dateien ist. Ein Lesen vieler kleiner Dateien, die im HDFS verteilt sind, kann sogar zu starken Performanceproblemen führen. Die CSV-Dateien sollen daher zuerst mit großen Mengen an Inhalten der geparsten XML-Dateien gefüllt werden und anschließend durch Apache Hive importiert werden.

Eine Recherche nach Tools für die Durchführung des regelmäßgen Imports ergab kein Ergebnis, höchstwahrscheinlich aufgrund der Einzigartigkeit dieser Aufgabe und der technischen Anforderungen (siehe 6.5.8). Es wurde entschieden, eine Entwicklung in Java durchzuführen, da in dieser Programmiersprache Vorkenntnisse durch den zu bearbeitenden Entwickler vorhanden sind.

Zur Einordnung in den Transformationsprozess und zum Verständnis des Vorgehens wird HIMP folgend graphisch als Prozess dargestellt:

Zu erkennen ist, dass HIMP nur eine kleine Aufgabe im Prozess des Importierens durch Hive darstellt. Es soll sowohl nach CSV-Dateien gesucht, als auch Hive zum Import aufgefordert werden. Dabei ist der Inhalt der Dateien und der Tabellen unerheblich, sodass bei Änderungen HIMP nicht mitgeändert werden muss, sondern lediglich die Quelldateien und Zieltabellen.

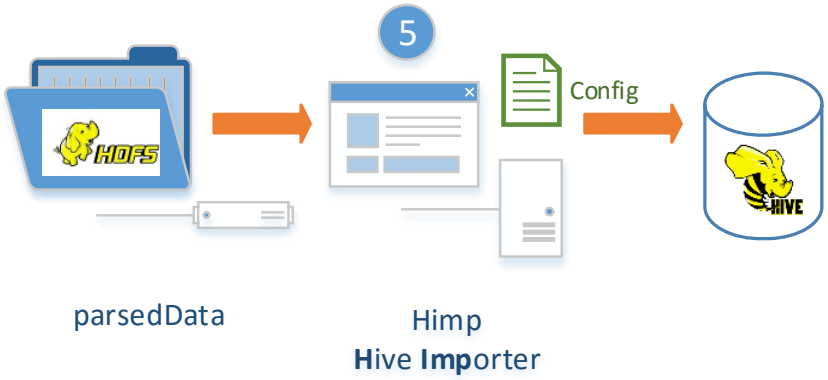


Abbildung 6.18.: Einordnung von HIMP im Transformationsprozess

Technische Anforderungen

Es wurde sich nach dem Betrachten der Quell-XML-Dateien auf eine relationale Struktur für Hive geeinigt. Der XMLParser übersetzt die XML-Dateien in genau dieses Format und speichert die Daten in das spaltenbasierte CSV-Format, welches den Tabellen in Hive entspricht. Es hat keine Überprüfung oder Fehlerbehandlung des strukturellen Inhalts stattzufinden. Die Dateinamen entsprechen den Tabellen in Hive.
Stattdessen soll ein vordefiniertes Verzeichnis nach CSV-Dateien mit bestimmten Dateinamen untersucht werden und jeweils eine vordefinierte Aktion durchgeführt werden: In diesem Fall die Überführung der Daten nach Apache Hive.

Der gesamte Workflow und insbesondere die Frequenz der Überprüfung sollen in dem Programm selbst nicht berücksichtigt werden. Durch eine andere Schicht, einem Management des Workflows, soll dies abgewickelt werden. HIMP führt daher pro Aufruf nur *einmalig* seine Aktion durch.

Implementation und genutzte Bibliotheken

Im Vordergrund der Entwicklung stand die Dynamik des Programms. Es sollte bei Änderungen des relationalen Modells oder des Workflows keine Änderungen an HIMP benötigt sein. Daher wurde sich für eine größtmögliche Steuerung des Programmes über externe Konfigurationsdateien entschieden.

Apache Hive bietet bereits eine Funktion zum Importieren von CSV-Dateien in passende Schema an [Apa16i]. Diese kann problemlos genutzt werden. Der HiveQL-Befehl sieht hierfür wie folgt aus:

SQL 6.129: HiveQL für das Importieren von CSV-Dateien in Apache Hive

```
1   LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE]
2   INTO TABLE tablename [PARTITION (partcol1=val1, partcol2=val2
        ...)]
```

Zusätzlich bietet Apache Hive die Ausführung von HiveQL-Befehlen über die Shell an [Clo16b]. Folgend wird diese Funktion aufgezeigt:

Bash 6.130: HiveQL Ausführung von der Shell

```
1 hive -e 'HiveQL-statement'
```

Die Kombination dieser beiden Möglichkeiten bieten sich als Schnittstelle zum Importieren von CSV-Dateien nach Hive aus externer Quelle an und werden vn HIMP genutzt. Dies kann durch die Konfigurationsdatei, welche die Ausführung des Programms steuert, erkannt werden. Durch diese wird ersichtlich, wie die Ausführung des Programmes beabsichtigt wird:

Config 6.131: Konfigurationsdatei von HIMP

```
1  # Configuration file for HIMP
2
3  # List of files that HIMP has to look for, split by comma.
4  himp.filenames=session.csv,screen_action.csv,printjob.csv,
       printerstatus.csv,focused_product.csv,video_list.csv,device.
       csv,dir.csv,image.csv
5
6  # HDFS host and directory that HIMP will be looking in.
7  himp.host=http://localhost:50070
8  himp.dir=/webhdfs/v1/user/ukopplin/csv
9
10 # Action that will be taken when a file is found. Possible
       variables: %FILENAME%, %FILENAME_WITHOUT_EXTENSION%.
11 himp.action=hive -e "LOAD INTO HDFS FROM %FILENAME% INTO %
       FILENAME_WITHOUT_EXTENSION%"
```

Ersichtlich wird, dass zuerst alle Dateien festgelegt werden, nach welchen gesucht wird. Anschließend wird ein Host und ein Verzeichnis festgelegt, in welchem nach diesen Dateien gesucht wird. Weiterhin ist noch festzulegen, was geschehen soll, sobald eine Datei gefunden wird. Praktisch wird hierbei auf die angesprochenen Optionen der Ausführung von HiveQL aus der Kommandozeile zu-

rückgegriffen. Dabei werden zwei Variablen zur Verfügung gestellt: Der Dateiname der gefundenen Datei und der Dateiname ohne Dateiendung. Da die Tabellenbezeichnung laut Festlegung der Projektgruppe immer dem Dateinamen (ohne Endung) entspricht, wurden diese Variablen eingeführt. Durch diese breite Konfigurationsmöglichkeit ist das Programm variabel einsetzbar, wird aber voraussichtlich in der Projektgruppe an keiner zweiten Stelle benötigt.

Die Umsetzung in Java erfolgt wie durch die Anforderungen vorgegeben. Es wird in dem vorgegebenen Verzeichnis nach bestimmten Dateien gesucht und pro gefundener Datei ein Shell-Kommando (betriebssystemunabhängig) ausgeführt. Die Main-Methode des Programmes verdeutlicht den Workflow und wird daher folgend dargestellt.

Java 6.132: Java-Workflow von HIMP

```
1   /**
2    * Main Method. The configuration file, which was specified in
         the run
3    * parameters is being loaded. The specified location in the
         file is being
4    * read and the command executed in the shell.
5    *
6    * @param args
7    *            The run parameters. The config file is expected as
         the first
8    *            argument.
9    */
10  public static void main(String[] args) {
11    try {
12
13      // Loading and setting up the configuration
14      Properties properties = loadConfig(args);
15      String[] filenames = properties.getProperty("himp.filenames"
            ).split(",");
16      final String host = properties.getProperty("himp.host");
17      final String dir = properties.getProperty("himp.dir");
18      final String action = properties.getProperty("himp.action");
19
20      // Reading all filenames from the directory in the HDFS
21      List<String> filesInDirectory = getAllFiles(host, dir);
22
23      // Searching the filenames for files in the config
24      for (String file : filesInDirectory) {
25        if (arrayContains(filenames, file)) {
26
27          // Logging the finding of a file.
```

```
28            LOG.info("Found a file: ".concat(file));
29
30            // Generating the shell command
31            String command = action.replace(VAR_FILENAME, file);
32            command = command.replace(VAR_FILENAME_WITHOUT_EXTENSION
                  , FilenameUtils.removeExtension(file));
33
34            // Executing the command in the shell
35            executeCommand(command);
36          }
37        }
38      } catch (Exception e) {
39        // Logging
40        LOG.error("Error: ", e);
41        System.exit(0);
42      }
43    }
```

Es wird für jede im Verzeichnis gefundene Datei ein Shell-Command generiert und ausgeführt. Ein weiteres kritisches Segment im Quellcode ist die Durchsuchung des Verzeichnisses. Das Logging erfolgt durch die Bibliothek *log4j*. Zudem wird die Bibliothek *Apache Commons IO* für das Wegkürzen der Dateiendung verwendet, um Fehler zu vermeiden.

Der Zugriff erfolgt über *WebHDFS*, einer Schnittstelle, durch welche sich das HDFS über HTTP-Abfragen steuern und auslesen lässt. Hierfür werden keine weiteren externen Bibliotheken benötigt. Folgend wird die zugehörige Methode in Java dargestellt:

Java 6.133: Zugriff auf WebHDFS und parsen von JSON in HIMP

```
1   /**
2    * Returns all filenames that are found in a specified path of a
         HDFS. For
3    * this, WebHDFS is used, a HTTP-interface for communicatin with
         HDFS.
4    * Responses from WebHDFS are given in JSON-format. This will be
         parsed and
5    * the required information will be stored and returned in a
       ArrayList.
6    *
7    * @param host
8    *            The given host of the HDFS, for example
9    *            "http://localhost:50075"
10   * @param dir
```

```
11      *            The directory the Watchdog searches in.
12      * @return All the Filenames of the files that have been found.
13      * @throws IOException
14      *             Exception for when there is a communication
            problem with the
15      *             file system.
16      * @throws URISyntaxException
17      */
18     private static List<String> getAllFiles(final String host, final
            String dir)
19         throws IOException, URISyntaxException {
20
21       // Building the WebHDFS URL
22       final String url = host.concat(dir).concat("?op=LISTSTATUS");
23
24       // Logging
25       LOG.info("Getting file metadata from WebHDFS URL: ".concat(url
             ));
26
27       // Setting up the list that will be filled and returned
28       final List<String> fileNames = new ArrayList<String>();
29
30       // Generating the URL to be called to access the WebHDFS
31       URI uri = new URI(url);
32
33       // Executing the HTTP-request and storing the JSON-response in
             a
34       // JSON-Object
35       JSONTokener tokener = new JSONTokener(uri.toURL().openStream()
             );
36       JSONObject root = new JSONObject(tokener);
37
38       // The structure of the WebHDFS-responses are known and parsed
             to get
39       // the filename of each file found in the directory
40       JSONArray arrayOfFileStatuses = (JSONArray) ((JSONObject) root
             .get("FileStatuses")).get("FileStatus");
41       for (int i = 0; i < arrayOfFileStatuses.length(); i++) {
42         JSONObject file = (JSONObject) arrayOfFileStatuses.get(i);
43
44         // Adding the filename of a found file in the list
45         fileNames.add((String) file.get("pathSuffix"));
46       }
47
48       // returning the list.
49       return fileNames;
50     }
```

Die Antworten des WebHDFS erfolgen im JSON-Format. Zum Lesen und Parsen wurde hierfür die Bibliothek *org.json* verwendet.

Fazit

Die java-basierte Eigenentwicklung erfüllt seine Aufgabe und ist durch eine ausgelagerte Konfigurationsdatei ohne neue Kompilierung veränderbar, um eine Verwendung für weitere Aufgaben zu ermöglichen. Es wird fest im Workflow der Datentransformation implementiert und spielt dabei eine zuverlässige Rolle. Das Programm selbst ist ressourcenschonend, da dieses nicht selbst die Importierung der Daten vornimmt, sondern dieses Apache Hive überlässt, welches über ein auf dem Cluster verteiltes Verarbeiten eine performante Importierung großer Dateien vornimmt.

6.6. Data Analytics

In diesem Kapitel soll das Vorgehen der Projektgruppe im Bereich Analytics dargestellt werden. An dieser Stelle wird exemplarisch das Vorgehen dargestellt, da Analysen mit konkreten Inhalten auf Grund des Geschäftsgeheimnisses gegenüber dem Unternehmenspartner CEWE nicht dargestellt werden können. Neben des ELT-Prozesses war es die Aufgabe der Projektgruppe Wissen und Erkenntnisse aus den Informationen zu generieren. Der Bereich deskriptive Analytics ist mit Kennzahlensteckbriefen strukturiert worden und die Kennzahlen sind mit Hive umgesetzt worden. Weitere Analysen im Bereich deskriptive und predictive Analytics wurden mit der Spark MLlib umgesetzt.

6.6.1. Kennzahlensteckbriefe

Grundlage der Kennzahlenanalyse sind die Kennzahlensteckbriefe. Diese beschreiben jede erhobene Kennzahl durch folgende Unterpunkte:

Aspekt	Beschreibung
ID	Eindeutiges Kürzel der Kennzahl, beispielsweise *KPI-01*.
Kennzahl	Aussagekräftiger Name der Kennzahl.
Definition	Beschreibt kompakt, was die Kennzahl beschreiben soll.

Verwendungszweck, Aussagekraft und -grenzen	Sagt aus, in welchem Kontext die Zahl von Nutzen ist, wie sie gedeutet werden kann und welche Aussagekraft sie nicht beziehungsweise nur bedingt hat.
Berechnung	Die Berechnung erklärt, aus welchen Werten sich die Zahl berechnen lässt.
Auswertungseinschränkungen	Beschreibt Einschränkungen, die bei der Interpretation der Werte berücksichtigt werden müssen.
Datenquelle(n)	Hier wird aufgezählt, woher die Daten und Werte zur Berechnung der Kennzahl stammen.

Tabelle 6.4.: Aspekte der Kennzahlensteckbriefe

Exemplarisch wird hier der Steckbrief der ersten Kennzahl dargestellt:

Aspekt	**Beschreibung**
ID	KPI-01
Kennzahl	Drucke pro Tag in Monat X
Definition	Beschreibt die Auftragsleistung eines DFM
Verwendungszweck, Aussagekraft und -grenzen	Die Kennzahl dient zum einen zur besseren Vergleichbarkeit einzelner DFM miteinander, kann dabei Kapazitätsgrenzen aufzeigen oder hohe Leerlaufzeiten aufdecken. Weiter kann bei wiederholten Ausreißern auf mögliche Probleme aufmerksam gemacht werden. Eine Bewertung kann durch einen Vergleich mit dem Mittelwert aller Kennzahlenwerte erfolgen.
Berechnung	Anzahl Prints in Monat X / Tage von Monat X
Auswertungseinschränkungen	Die Kennzahl wird beeinflusst durch Öffnungszeiten der DFM-Standorte. Längere Ausfallzeiten werden nicht berücksichtigt. Es gibt hierbei keine Differenzierung verschiedener Produkttypen.
Datenquelle(n)	Hive-Tabellen: parsed_xml_data

Tabelle 6.5.: Beispiel: KPI-01 Drucke pro Tag in Monat X

Die Umsetzung der Kennzahl in Zeppelin veranschaulicht Abbildung 6.19.

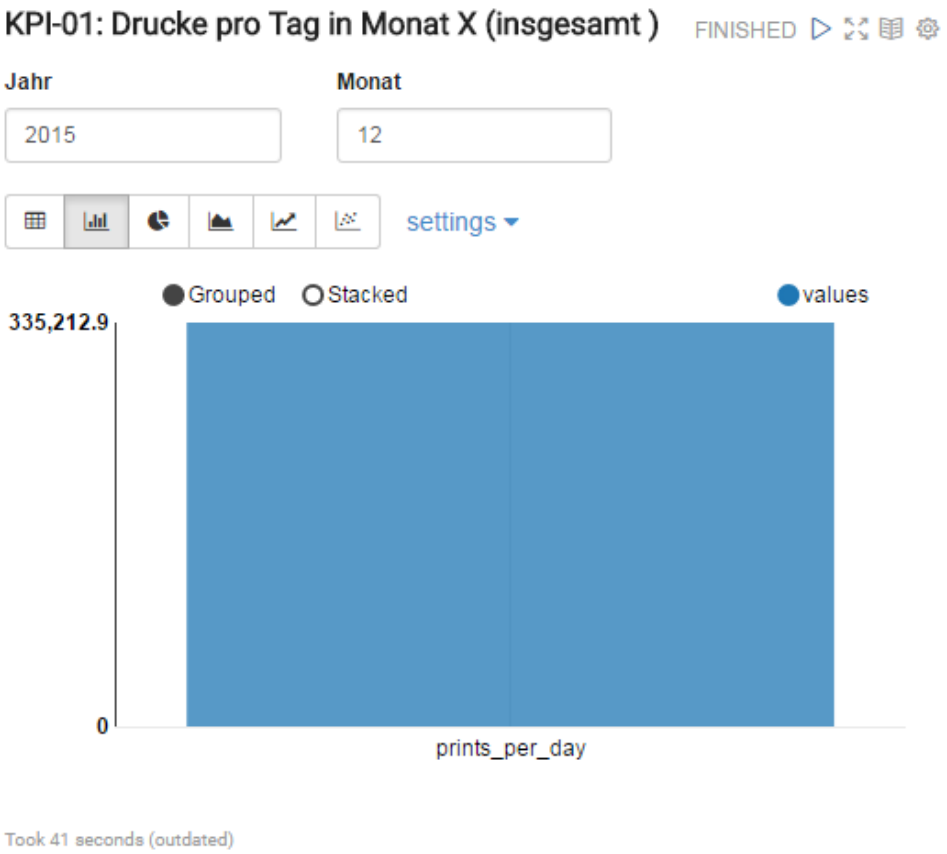


Abbildung 6.19.: Visualisierung von KPI-01 in Zeppelin

Auch komplexere Kennzahlen wie beispielsweise die Häufigkeit von Headcooling-Prozessen in einzelnen Sessions lassen sich mittels Steckbrief beschreiben und anschließend in Zeppelin umsetzen und visualisieren.

6.6.2. Analyseverfahren mit Hive

In diesem Abschnitt werden die im Rahmen der Projektgruppe mit Apache Hive durchgeführten Analysen dargestellt und erläutert. Das Data Warehouse Hive bietet hierfür einen SQL-Dialekt namens *HiveQL* [Atl15] an. Die durchgeführten Analysen drehen sich unter anderem um die Berechnung zahlreicher Kennzahlen, welche in einem interaktiven Notizbuch von Apache Zeppelin visualisiert werden. Im Folgenden wird dies anhand der ersten Kennzahl detailliert dargestellt.

KPI-01: Drucke pro Tag in Monat X

Die Kennzahl zeigt, wie viele Fotos in einem bestimmten Monat durchschnittlich pro Tag gedruckt wurden. So lassen sich zum Beispiel zwei verschiedene Standorte jeweils zur Weihnachtszeit miteinander vergleichen. Hierfür werden die beiden Datenbanken *parsed_xml_data* und *default* verwendet. Die genutzten Tabellen sind *product* sowie *session* aus der ersten und die View *Tage_pro_Monat* aus der zweiten Datenbank. Die im dargestellten SQL-Ausschnitt verwendeten Dollarzeichen stellen Variablen (bspw. für den zu untersuchenden Monat) dar, über die der Benutzer interaktiv in Apache Zeppelin den jeweiligen Wert anpassen kann. Ein einfaches ad-hoc generiertes GUI-Element ermöglicht folglich die Manipulation des SQL auch für unerfahrene Anwender.

Abfrage 6.134: Drucke pro Tag in Monat X

```
1  select x.dfmid, x.prints from
2  (select s.year, s.month, t.days, s.dfmid, (sum(nvl(p.
       printobjectscomplete,0)) / t.days) as prints
3  from parsed_xml_data.product p, parsed_xml_data.session s,
       default.Tage_pro_Monat t
4  where p.uuid = s.uuid and (s.orderstate = 'printed' or s.
       orderstate = 'success')
5  and s.year = ${Jahr=2015} and s.month = ${Monat=12}
6  and s.year = t.year and s.month = t.month
7  group by s.year, s.month, t.days, s.dfmid) x
8  order by x.prints ${Sort=desc,asc|desc}, x.dfmid
9  limit ${Grenze=7}
```

6.6.3. Analyseverfahren mit Spark (Data Mining)

Mit der Spark MLlib sind Algorithmen wie Entscheidungsbäume, K-Means und die Assoziationsanalyse mit Testdaten umgesetzt worden, sodass diese Umsetzungen als Vorlage für die Analysen mit CEWE-Daten dienten. Primär sollte sich auf die Python API von Sparks MLlib konzentriert werden. Beim implementierten Entscheidungsbaum Algorithmus wiesen die Python, Scala und Java API in der Spark MLlib 1.6.2 Unterschiede im Funktionsumfang aus. Sodass im Bereich Spark MLlib die Scala API, die mit dem größten Funktionsumfang, benutzt worden ist.

In Scala Code Block 6.135 ist der implementierte FP-Growth Algorithmus mit der Spark MLlib und der Scala API zu sehen. Dieser Algorithmus konnte im Bereich Warenkorbanalyse und zur ersten Erforschung der Clickstreamdaten benutzt

werden. Das Scala Programm wird zu einer ausführbaren JAR-Datei gebuildet. Damit es Use Case unabhängig ist, kann dem Programm der Support und das SQL für die Datenselektion per Parameter übergeben werden. (siehe Zeile 14 und 18) Dabei erwartet das Programm ein SQL welches zwei Spalten selektiert. In der ersten Spalte die ID, die die Items verbindet und in der zweiten Spalte die Items selbst. In Zeile 16 werden die Daten für den FP-Growth Algorithmus aufbereitet. In Zeile 20 wird der FP-Growth Algorithmus mit den Daten ausgeführt. Der Zeilenabschnitt 25 bis 26 zeigt das extrahieren und ausgeben der Frequent Itmesets mit dem entsprechenden Support. In Zeilenabschnitt 33 bis 38 werden die Assoziationsregeln unter Berücksichtigung der Konfidenz gebildet und ebenfalls ausgegeben.

Scala 6.135: FP-Growth - Spark MLlib Scala API

```
1  /*
2  The program gets two arg-parameters.
3  The first is the support.
4  The second is the SQL-Statement with the data for the FP-Growth
       algorithm.
5  The SQL Statement (e.g. for a market basket analysis) should
       deliver in the first column the basket id and in the second
       column for e.g. the product.
6  */
7  object assoziationsanalyse {
8    def main(args: Array[String]) {
9
10     val conf = new SparkConf().setAppName("Assoziationsanalyse")
11     val sc = new SparkContext(conf)
12     val hiveContext = new HiveContext(sc);
13
14     val sql_string = args(2)
15     val transactions = hiveContext.sql(sql_string);
16     val transactions_erg = transactions.rdd.map(row => (row.get
           (0),row.get(1))).groupByKey().map(element => element._2.
           mkString(",").split(","))
17
18     val support = args(0).toDouble
19     val fpg = new FPGrowth().setMinSupport(support)
20     val model = fpg.run(transactions_erg)
21
22     /*
23     Get the Frequent Itemsets
24     */
25     model.freqItemsets.collect().foreach { itemset =>
26       println(itemset.items.mkString("[", ",", "]") + ", " +
```

```
                itemset.freq)
27        }
28
29
30        /*
31        Get the Associatonrules with the defined confidence.
32        */
33        val minConfidence = args(1).toDouble
34        model.generateAssociationRules(minConfidence).collect().
              foreach { rule =>
35          println(
36          rule.antecedent.mkString("[", ",", "]")
37          + " => " + rule.consequent .mkString("[", ",", "]")
38          + ", " + rule.confidence)
39        }
40      }
41
42    }
```

6.7. Data Visualization

Zur visuellen Darstellung der Ergebnisse einiger Data Mining- sowie der Kennzahlenanalysen wurde das Tool Zeppelin ausgewählt. Es bietet Interpreter für Spark, Scala, Hive und weitere Sprachen sowie die optische Aufbereitung der Ergebnisse in Form unterschiedlicher Diagrammarten wie beispielsweise Kreis-, Stab- oder Punktdiagramme. Neben diesen standardmäßigen Darstellungen wurden weitere Diagrammformate, unter anderem Balkendiagramme, der kostenlosen JFreeChart-API hinzugefügt. [Obj14] Weiter konnten Kartendiagramme eingebunden werden um den regionalen Bezug der Daten verständlich darzustellen.

Die Erarbeitung und Umsetzung von Analysen sind in den Abschnitten 6.6.3 und 6.6.2 dokumentiert.

6.7.1. Zeppelin-Erweiterung: vis–à–vis

Apache Zeppelin besitzt keine Möglichkeit zur Visualisierung von geographischen Zusammenhängen. Somit war es nötig, eine eigene Java-Bibliothek für die Einbindung von Google Maps in die Notizbücher von Zeppelin zu schreiben. Diese Bibliothek wird *vis–à–vis* (französisch für *von Angesicht zu Angesicht*) genannt und steht nun standardmäßig im Dashboard zur freien Nutzung für folgende Visualisierungen zur Verfügung:

– Tabellengenerierung per Spark

- Normalisierte gestapelte Balkendiagramme(siehe 6.21)
- Kartendiagramme (mittels Google Maps)(siehe 6.20)

In der Abbildung 6.20 ist eine Karte zu sehen, welche die Verbindungsfehler (in den Rohdaten der DFM „connectivity-test“ genannt) am 7. Dezember 2015 im Umkreis von Hamburg für einen Endanwender anschaulich visualisiert. Dies geschieht zur Zeit mit Hilfe von Google Maps [8], könnte in der Zukunft jedoch auch auf das freie OpenStreetMap[9] umgestellt werden. In letzterem Falle würde die Firma CEWE selbst einen eigenen Server für das Kartenmaterial bereitstellen oder diesen von einem Dienstleister mieten.

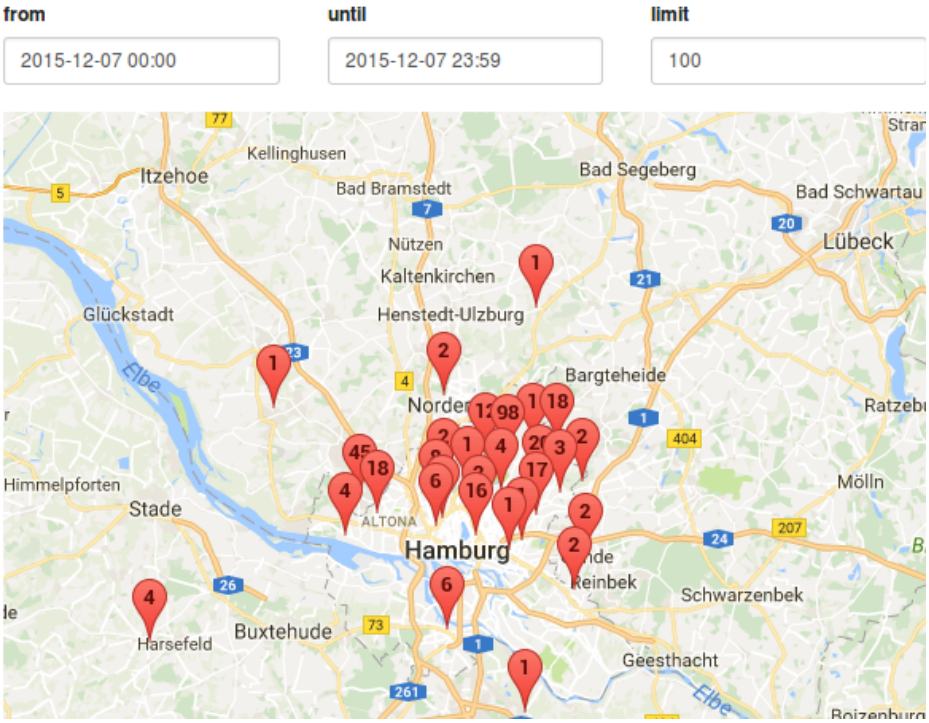


Abbildung 6.20.: Auswertung der Connectivity-Tests im Umkreis von Hamburg

[8] https://developers.google.com/maps/?hl=de
[9] https://www.openstreetmap.de/

Abfrage 6.136: Connectivity Test

```
import scala.collection.JavaConversions._
val geo = sqlContext.sql("""select g.lat, g.lon, k.vis_ranking,
    concat('red', count(c.`timestamp`))
from parsed_xml_data.connectivitytest c, parsed_xml_data.session
     s, ext_data.kunde k, parsed_xml_data.plz_geo g
where c.xtcitest_connectivity = 'failed'
and c.uuid = s.uuid
and s.location is not null
and substr(s.location,0,length(s.location)-2) = k.dealer
and k.plz = g.plz
and c.`timestamp` >= '""" + z.input("from","2015-12-07 00:00") +
     "' and c.`timestamp` <= '" + z.input("until","2015-12-07
    23:59") + """'
group by lat, lon, vis_ranking
order by count(c.`timestamp`) desc
limit """ + z.input("limit","20")).map { r => r.toSeq.toArray }
import pgdash.visavis._
println(Charts.map(geo.collect().toList))
```

In der Abbildung 6.21 sind die Bestellstatus aus allen Sessions des gleichen Zeitraums mit Hilfe eines gestapelten normalisierten Balkendiagramms je nach Kunde prozentual aufgeteilt. Abseits von absoluten Zahlen tritt die Größe und Umsatzstärke eines einzelnen Kunden in den Hintergrund und es wird die relative Häufigkeit der Orderstates deutlich. Mit den Bordmitteln von Apache Zeppelin war diese Veranschaulichung nicht auf diese Art und Weise möglich, so dass sich die selbst geschriebene Erweiterung den zahlreichen Möglichkeiten der bekannten Java-Diagrammbibliothek *JFreeChart*[10] bedient.

Abfrage 6.137: Bestellstatus

```
import pgdash.visavis._
import scala.collection.JavaConversions._
val data = sqlContext.sql("""
select a.dealer, a.bestellstatus, a.anzahl from
(select k.vis_ranking as Dealer, s.orderstate as Bestellstatus,
    count(*) AS Anzahl
from parsed_xml_data.session s, ext_data.kunde k
where substr(s.location,0,length(s.location)-2) = k.dealer
and s.`timestamp` >= '2015-12-07 00:00' and s.`timestamp` <=
    '2015-12-07 23:59'
group by s.orderstate, k.vis_ranking) a,
```

[10] http://www.jfree.org/jfreechart/

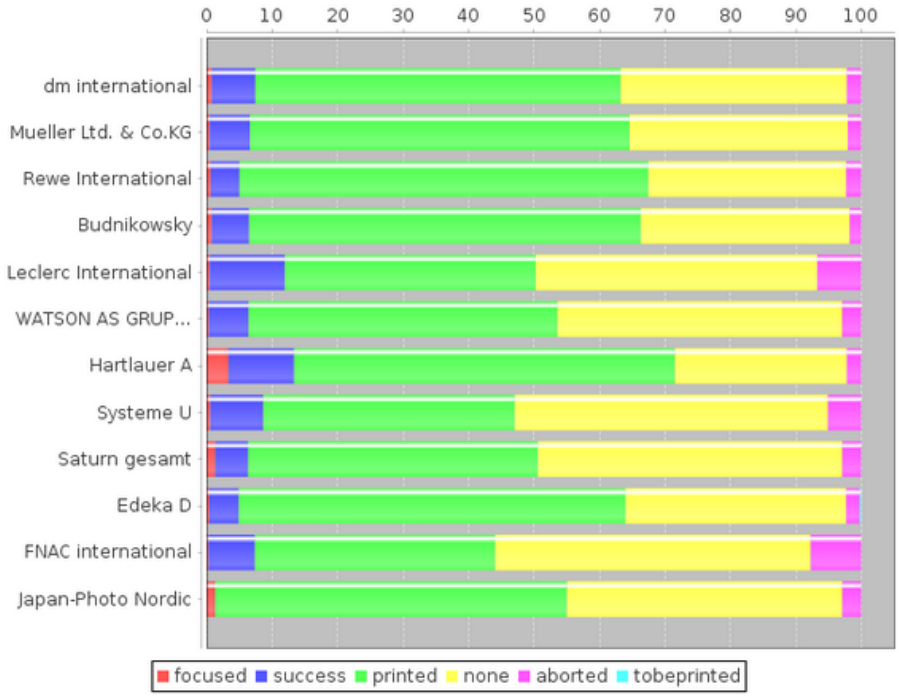


Abbildung 6.21.: Prozentuale Aufteilung der Bestellstatus nach Kunde

```
10    (select k.vis_ranking as Dealer, count(*) AS Anzahl
11    from parsed_xml_data.session s, ext_data.kunde k
12    where substr(s.location,0,length(s.location)-2) = k.dealer
13    and s.'timestamp' >= '2015-12-07 00:00' and s.'timestamp' <=
          '2015-12-07 23:59'
14    group by k.vis_ranking) b
15    where a.Dealer = b.Dealer order by b.anzahl desc limit 60
16    """).map { row => row.toSeq.toArray }
17    println(Charts.stacked(data.collect().toList, null, null, null))
```

6.8. Workflowmanagement

Zur Steuerung, Überwachung und Einplanung der genutzten Tools innerhalb des ELT-Prozesses wurde Airflow verwendet. Zunächst wurde Airflow auf Basis einer SqlLite-Datenbank mit einem Sequentiell-Executor genutzt.

In der Projektgruppe wurden drei DAGs erstellt: Ein DAG für den Standard-Workflow, ein zweiter DAG für den Repeat-Workflow und ein dritter *alpha_deployer* zur jeweiligen Aktualisierung der Tools vor jedem Test des ELT-Prozesses. Nachfolgend wird der *dag_alpha_deployer.py* DAG dargestellt.

Python 6.138: alpha deployer Quellcode

```
from airflow import DAG
from airflow.operators import BashOperator
from datetime import datetime, timedelta

default_args = {
  'owner': 'hdusermaster',
  'depends_on_past': True,
  'start_date': datetime(2016, 6, 28),
  'email': ['sadok.ben.yahya@uni-oldenburg.de'],
  'email_on_failure': True,
  'email_on_retry': True,
  'retries': 1,
  'retry_delay': timedelta(minutes=5),
  'schedule_interval': None,
}

dag = DAG('dad', default_args=default_args)

AlphaFolderStructure = BashOperator(

  task_id='alpha_folder_structure',
  bash_command="""
  cd /home/hdusermaster
  sudo python alpha_deployer.py --cas --test
  """,
  retries=1,
  dag=dag
)


AlphaFolderStructure.doc_md = """
  AlphaFolderStructure is the instanciation of the
      alpha_deployer.py to automatically create or delete the
      alpha folder structure.
  """

AXTDep = BashOperator(


  task_id='axt_deploy',
  bash_command="""
  cd /home/hdusermaster
  sudo python alpha_deployer.py --tool AXT --test
  """,
  retries=1,
  dag=dag
```

```
45    )
46
47    AXTDep.doc_md = """
48      axt_deploy is the instanciation of the alpha_deployer.py to
            automatically deploy or update (if necessary) AXT.
49      """
50
51    PYXMLPARSERDep = BashOperator(
52
53
54      task_id='pyxmlparser_deploy',
55      bash_command="""
56      cd /home/hdusermaster
57      sudo python alpha_deployer.py --tool PYXMLPARSER --test
58      """,
59      retries=1,
60      dag=dag
61    )
62
63    PYXMLPARSERDep.doc_md = """
64      pyxmlparser_deploy is the instanciation of the alpha_deployer.
            py to automatically deploy or update (if necessary)
            PYXMLPARSER.
65      """
66
67    HIMPDep = BashOperator(
68
69
70      task_id='himp_deploy',
71      bash_command="""
72      cd /home/hdusermaster
73      sudo python alpha_deployer.py --tool HIMP --test
74      """,
75      retries=1,
76      dag=dag
77    )
78
79    HIMPDep.doc_md = """
80      himp_deploy is the instanciation of the alpha_deployer.py to
            automatically deploy or update (if necessary) HIMP.
81      """
82    SAPPYDep = BashOperator(
83      task_id='sappy_deploy',
84      bash_command="""
85      cd /home/hdusermaster
86      sudo python alpha_deployer.py --tool SAPPY --test
87      """,
88      retries=1,
```

```
89      dag=dag)
90
91    AIRFLOWDep = BashOperator(
92      task_id='airflow_deploy',
93      bash_command="""
94      cd /home/hdusermaster
95      sudo python alpha_deployer.py --tool AIRFLOW --test
96      """,
97      retries=1,
98      dag=dag)
99
100   ZEPPELINDep = BashOperator(
101     task_id='zeppelin_deploy',
102     bash_command="""
103     cd /home/hdusermaster
104     sudo python alpha_deployer.py --tool ZEPPELIN --test
105     """,
106     retries=1,
107     dag=dag)
108   AXTDep.set_upstream(AlphaFolderStructure)
109   PYXMLPARSERDep.set_upstream(AXTDep)
110   HIMPDep.set_upstream(PYXMLPARSERDep)
111   SAPPYDep.set_upstream(HIMPDep)
112   AIRFLOWDep.set_upstream(SAPPYDep)
113   ZEPPELINDep.set_upstream(AIRFLOWDep)
```

Sofern alle Tools im ELT-Prozess eingesetzt werden sollen, muss hierfür der folgende Befehl genutzt werden:

Bash 6.139: Starten des gesamten ELT workflow

```
1   sudo airflow backfill dad -s 2016-07-04 -e 2016-07-04
```

Durch den Aufruf mit *Backfill*werden die Abhängigkeiten der Tasks des gesamten Prozesses beachtet. Hier *dad* ist die *alpha_deployer dag_id*. Die *-s* bzw. *-e* Parametern definieren das Start- bzw. Enddatum für den Lauf. Je nach Bedarf gibt es die Möglichkeit einzelne Task gekapselt auszuführen.

Bash 6.140: Starten einzelner Tasks des ELT-Workflow

```
1   sudo airflow test dad [task\_id] 2016-07-04
```

Die *Task_id* könnte durch folgende Tasks ersetzt werden:

- alpha_folder_structure
- axt_deploy
- himp_deploy
- pyxmlparser_deploy
- sappy_deploy
- zeppelin_deploy

Wie bereits erwähnt gibt es den Standard- und den Repeat-Workflow. Für beide Workflows wurde ein DAG erstellt.

- Produktion DAGs
  - alpha_data_pipeline_dag.py
  - alpha_repeat_data_pipeline.py
- Entwicklung/Test DAGs
  - alpha_data_pipeline_dag_test.py
  - alpha_repeat_data_pipeline_test.py

Als Beispiel wird im Folgenden Python-Skript der DAG für den Standard-Workflow dargestellt.

Python 6.141: standard workflow DAG

```
1
2   from airflow import DAG
3   from airflow.operators import BashOperator, BranchPythonOperator
4   from datetime import datetime, timedelta
5
6
7   default_args = {
8     'owner': 'hdusermaster',
9     'depends_on_past': True,
10    'start_date': datetime(2015, 12, 7),
11    'email': ['sadok.ben.yahya@uni-oldenburg.de'],
12    'email_on_failure': True,
13    'email_on_retry': True,
14    'retries': 1,
15    'retry_delay': timedelta(minutes=5),
16    'schedule_interval': timedelta(1)
17  }
18
19  dag = DAG('avdp', default_args=default_args)
20
21
22  RunHdfsOrdnerDeletion = BashOperator(
```

```
23
24      task_id='run_hod',
25      bash_command="""
26      sudo su hdfs -c 'hadoop fs -rm -r -skipTrash /ready2Parse'
27      sudo su hdfs -c 'hadoop fs -rm -r -skipTrash /parsedData'
28      sudo su hdfs -c 'hadoop fs -mkdir /ready2Parse'
29      sudo su hdfs -c 'hadoop fs -mkdir /parsedData'
30      sudo su hdfs -c 'hadoop fs -chmod -R 777 /ready2Parse'
31      sudo su hdfs -c 'hadoop fs -chmod -R 777 /parsedData'
32      """,
33      retries=3,
34      dag=dag
35    )
36
37
38    RunHdfsOrdnerDeletion.doc_md = """
39      Empty the /ready2Parse and /parsedData after each DAG run
40      """
41
42    RunAXT = BashOperator(
43
44      task_id='run_axt',
45      bash_command="""
46      java -Xms512m -jar /opt/pgdash/axt/axt/target/axt-0.0.1-
            SNAPSHOT-jar-with-dependencies.jar "/opt/pgdash/axt/axt/
            config_standard.properties" "{{ds}}" "{{ds}}" false true
            10
47      """,
48      retries=3,
49      dag=dag
50    )
51
52    RunAXT.doc_md = """
53      The run_axt is the instanciation of the AXT tool.
54      """
55
56    RunHadoopStreamingSappy = BashOperator(
57
58      task_id='run_hadoop_streaming_sappy',
59      bash_command="""
60      cd /opt/pgdash/launcher/launcher
61      python launcher.py --mode STREAM --parser SAPPY --input /
            ready2Parse/ --output /parsedData/csv/
62      """,
63      retries=3,
64      dag=dag
65    )
66
```

```
67
68    RunHadoopStreamingSappy.doc_md = """
69      The run_hadoop_streaming_sappy is the instanciation of the
            SAPPY sax based python parser executed through Hadoop
            Streaming.
70      """
71
72    RunSparkSappy = BashOperator(
73
74      task_id='run_spark_sappy',
75      bash_command="""
76      cd /opt/pgdash/launcher/launcher
77      python launcher.py --mode SPARK --parser SAPPY --input /
            ready2Parse/ --output /parsedData/csv/
78      """,
79      retries=3,
80      dag=dag
81    )
82
83
84    RunSparkSappy.doc_md = """
85      The run_spark is the instanciation of the SAPPY sax based
            python parser executed through Apache Spark.
86      """
87
88
89    RunCSVRipper = BashOperator(
90
91      task_id='run_csv_ripper',
92      bash_command="""
93      cd /opt/pgdash/himp
94      sudo ./csvripper.sh
95      """,
96      retries=1,
97      dag=dag
98    )
99
100   RunCSVRipper.doc_md = """
101      The run_csvripper_a is the instanciation of the csvripper
             script which cuts the first column of csv file.
102      This is neccessary because running the parser with hadoop
             streaming produces output files with an unused column.
103      """
104
105
106   RunHIMPA = BashOperator(
107
108     task_id='run_himp_a',
```

```
109     bash_command="""
110     cd /opt/pgdash/himp/runnable
111     sudo ./start.sh {{ds}}
112     """,
113     retries=3,
114     dag=dag
115   )
116   RunHIMPB = BashOperator(
117
118     task_id='run_himp_b',
119     bash_command="""
120     cd /opt/pgdash/himp/runnable
121     sudo ./start.sh {{ds}}
122     """,
123     retries=3,
124     dag=dag
125   )
126
127
128   RunHIMPA.doc_md = """
129     The run_himp_a is the instanciation of the HIMP java programm.
130     """
131   RunHIMPB.doc_md = """
132     The run_himp_b is the instanciation of the HIMP java programm.
133     """
134
135
136   def please_decide(**kwargs):
137
138     return 'run_hadoop_streaming_sappy'
139
140
141   RunBranching = BranchPythonOperator(
142
143     task_id='run_branching',
144     provide_context=True,
145     python_callable=please_decide,
146     dag=dag
147   )
148   RunBranching.doc_md ="""
149   The run_branching is the instanciation of the
        BranchPythonOperator, which in this case, allow to choose
        between parsing with Apache Spark und Hadoop Streaming
150   """
151
152   RunAXT.set_upstream(RunHdfsOrdnerDeletion)
153   RunBranching.set_upstream(RunAXT)
154
```

```
155
156     RunHadoopStreamingSappy.set_upstream(RunBranching)
157     RunCSVRipper.set_upstream(RunHadoopStreamingSappy)
158     RunHIMPA.set_upstream(RunCSVRipper)
159
160
161     RunSparkSappy.set_upstream(RunBranching)
162     RunHIMPB.set_upstream(RunSparkSappy)
```

Da zuerst der *LocalExecutor* genutzt worden ist und dann der *CeleryExecutor*, wurde eine MySQL Datenbank als Backend für Airflow eingeführt. Die standardmäßige SqlLite Datenbank unterstützt keine parallelen Zugriffe und somit hätte auch die Verteilungsfunktion des CeleryExecutor nicht genutzt werden können.

7. Hadoop Ecosystem der Projektgruppe

Während des Projektes wurden Anforderungen an die zu nutzenden Softwarekomponenten erstellt. Anhand dieser Anforderungen wurden verschiedene Komponenten untersucht, getestet und schließlich eine Auswahl getroffen. In diesem Kapitel wird die finale Architektur vorgestellt. In Abschnitt 7.1 wird das Hadoop Ecosystem der Projektgruppe beschrieben. Im folgenden Abschnitt 7.2 wird anschließend der ELT-Lauf im Detail erläutert.

7.1. Architektur Hadoop Ecosystem der Projektgruppe

Das Ergebnis der Projektgruppe bildet ein Anwendungssystem, das aus verschiedenen Tools des Hadoop Ecosystems besteht. Abbildung 7.1 zeigt alle ausgewählten Tools und wie diese in Beziehung zueinander stehen.

Die Abbildung zeigt verschiedene Layer, die die verschiedenen Tools beinhalten. Durch die Nummerierung in der Abbildung werden im Folgenden die Layer und ihr Zusammenspiel beschrieben. Der große graue Bereich kennzeichnet das Anwendungssystem. Im unteren Bereich der Abbildung sind die zu verarbeitenden Dateien dargestellt. Der mit der Nummer 1 gekennzeichnete FTP-Server stellt die XML-Dateien bereit. Der Datentransport in das HDFS wird durch das Tool AXT durchgeführt. Das HDFS bildet den Layer zur Datenspeicherung. In diesem Layer werden die Daten transformiert und strukturiert gespeichert. Die Schritte 2 bis 5 sind Teil des ELT-Laufs, der detailliert in Unterabschnitt 7.2 erläutert wird. Auf der Basis des strukturierten Datenbanklayers werden Analysen durchgeführt. Dabei werden Kennzahlen ermittelt und tiefer gehende Analysen mit Apache Hive und der Spark MLlib (Machine Learning Library) durchgeführt (Nummer 6). Die Ergebnisse der Analysen werden in einem Dashboard präsentiert. Für die Erstellung des Dashboards wird Apache Zeppelin genutzt. Wie in Nummer 7 zu sehen ist, werden die Daten für das Dashboard aus dem Analytics Layer sowie direkt aus dem HDFS angeliefert. Um die Daten der DFM-Geräte mit weiteren externen Daten anzureichern wurde eine Schnittstelle zum bestehenden DWH des Unternehmenspartners geschaffen. Nummer 8 zeigt, dass die Software Sqoop diese Aufgabe übernimmt. Zur Steuerung des gesamten Workflows von der Quelle bis in das HDFS wird das Tool Airflow verwendet (Nummer 9).

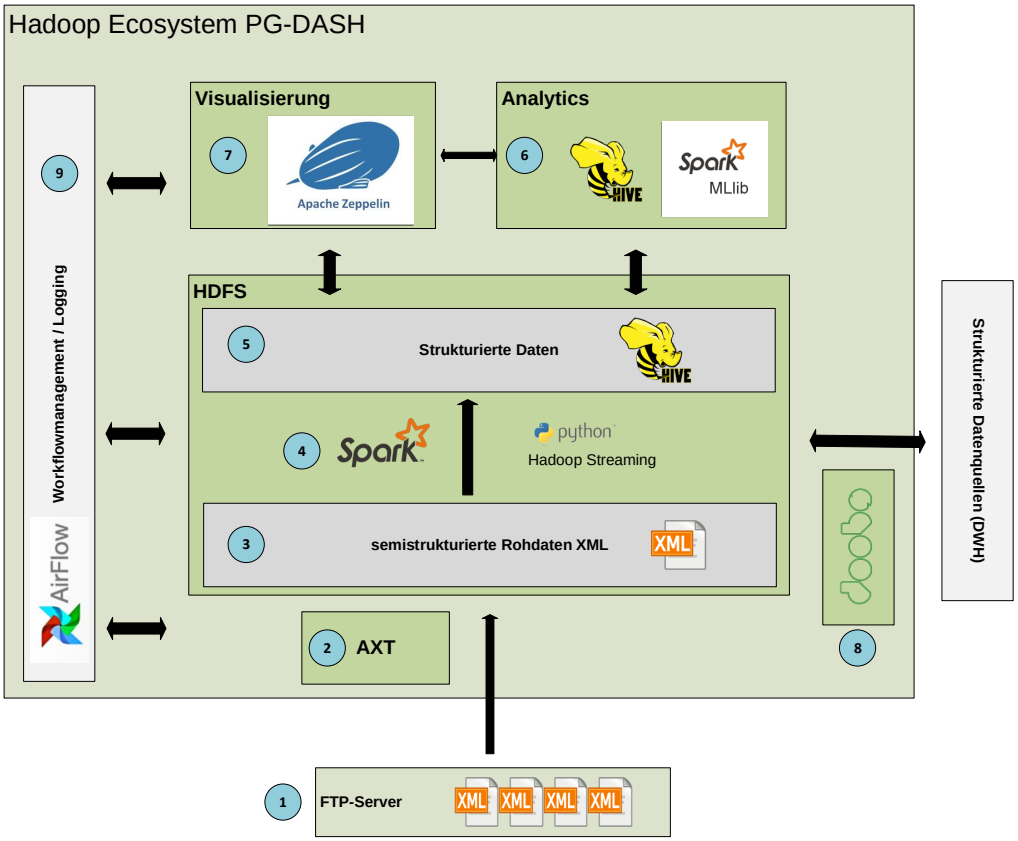


Abbildung 7.1.: Hadoop Ecosystem der PG Dash

7.2. ELT Flow

Nachdem im vorherigen Abschnitt die Gesamtarchitektur vorgestellt wurde, wird in den Folgenden Abschnitten ein detaillierter Blick auf den ELT-Lauf geworfen. Dabei werden zwei verschiedene Workflows differenziert. Der Standardworkflow (Abshcnitt 7.2.1) beschreibt die tägliche Verarbeitung der XML-Dateien. Der Repeat-Workflow definiert das Vorgehen bei einer wiederholten Verarbeitung von Daten und wird in Abschnitt 7.2.2 betrachtet.

7.2.1. Standardworkflow

In diesem Abschnitt wird beschrieben, wie die XML-Dateien vom FTP-Server in das HDFS geladen, geparst und die Daten in der Hive Schicht persistiert werden. Der Workflow wird in Abbildung 7.2 beschrieben.

Die Abbildung zeigt von links nach rechts den Datenfluss des Prozesses. Die genutzten Tools sind jeweils mit Nummern gekennzeichnet. Ordnerstrukturen im HDFS sind mit dem entsprechenden Logo gekennzeichnet.

Der dargestellte Workflow wird von dem Workflowmanagement Tool Airflow (siehe Kapitel 5.5.2) gesteuert und einmal täglich gestartet. Als erster Schritt des Prozesses werden die Ordner ready2Parse und parsedData gelöscht und anschließend neu erstellt und mit entsprechenden Rechten versehen. Dadurch soll gewährleistet werden, dass immer nur Daten eines Tages verarbeitet werden.

Die eigentliche Transport der Daten vom FTP Server, der die Quelle des ELT-Prozesses darstellt, startet mit AXT (Nummer 2 in der Abbildung). AXT überträgt die XML Dateien eines Tages in das HDFS. Dabei werden zwei Output Varianten durchgeführt. Zum einen werden die XML Dateien im Originalzustand in den Ordner XML Archive gespeichert. Dabei werden sie in eine Ordnerstruktur, die nach Jahr, Monat und Tag gefächert ist, einsortiert. Zusätzlich werden die XML Dateien in den Ordner ready2Parse transportiert. Dabei werden die XML Dateien bearbeitet. Axt bietet zum einen die Möglichkeit eine Menge an XML Dateien zu einer größeren Datei zusammenzuführen. Dies bietet den Vorteil, dass im weiteren Verlauf wenige größere Prozesse, anstatt vieler kleiner Prozesse, durchgeführt werden müssen. Zum anderen kann AXT die XML Dateien als einzelne Dateien übertragen. In beiden Fällen wird ein Gerüst um die XML Dateien gebaut. In diesem Gerüst sind der Dateiname und die Größe der Datei in Form von XML Tags enthalten. Die Funktionsweise von AXT wird im Kapitel 6.4.2 detailliert beschrieben.

Ab Prozessschritt Nr. 3 beginnt die Extraktion der Daten aus den XML Dateien. Dies übernimmt SaPPy. SaPPy wurde in der Programmiersprache Python implementiert und so konstruiert, dass dieser sowohl über Spark, als auch über Hadoop Streaming ausgeführt werden kann. Die Eigenarten dieser Methoden werden im Unterkapitel 6.5.2 beschrieben. Der Parser an sich besteht aus drei Schichten: Die Input Schicht liest die vorbereiteten Daten aus dem Ordner ready2Parse und leitet nacheinander eine XML Datei an die zweite Schicht weiter. Hier werden die Informationen aus den XML-Dateien extrahiert und an die dritte Schicht, den Output weitergeleitet. Je nach Ausführungsvariante (Spark oder Hadoop Streaming) wird ein entsprechendes Output Modul genutzt, um die Daten im CSV Format in den Ordner parsedData zu speichern. Die Struktur der geparsten Dateien orientiert sich bereits am Datenbankschema im relationalen Layer (siehe Kapitel 6.5.1). Für jede Datenbanktabelle wird eine Datei mit gleichem Namen erzeugt. Für das Laden der Daten in den relationalen Layer ist das Tool HImp (HiveImporter) zuständig (Nr. 4). HImp lädt alle Dateien in die entsprechenden Datenbanktabellen. Die Funktionsweise von HImp wird in Kapitel 6.5.8 erläutert. Der relationale Layer bildet die Grundlage für die Analyse der Daten und Data Mining.

7.2.2. Repeat Workflow

Der Repeat Workflow wurde designt, um bereits verarbeitete Tage erneut zu parsen. Diese Anforderung kann auftreten, wenn Verarbeitungen fehlerhaft waren, oder wenn vergangene Tage erneut mit einer neuen Parser Version verarbeitet werden sollen. Beispielsweise, wenn der Parser um neue Attribute erweitert wurde. Der Repeat Workflow wurde so erstellt, dass nicht der gesamte Prozess wiederholt werden muss, aber bereits bestehende Softwarekomponenten wiederverwendet werden können.

In der Abbildung 7.3 wird der Workflow dargestellt. Auch hier sind die Prozessschritte der Ausführungsreihenfolge nach mit Nummern versehen.

Der Prozess beginnt mit dem Ablöschen der Ordnerstrukturen (Nr. 1). Anschließend werden die bereits im HDFS gespeicherten Rohdaten des entsprechenden Tages geladen. Dies wird durch AXT ausgeführt. Die Aufteilung der Ordner in die Baumstruktur (Jahr - Monat - Tag) ermöglicht es der AXT schnell die benötigten Dateien zu finden. Die anschließenden Schritte 3 und 4 sind identisch mit denen des Standardworkflows. Um keine doppelten Daten im Datenbanklayer zu halten, wird die Partition der Tageszeitscheibe vor dem Import der Daten gelöscht und neu erstellt. Diese Funktionalität ist in HImp definiert.

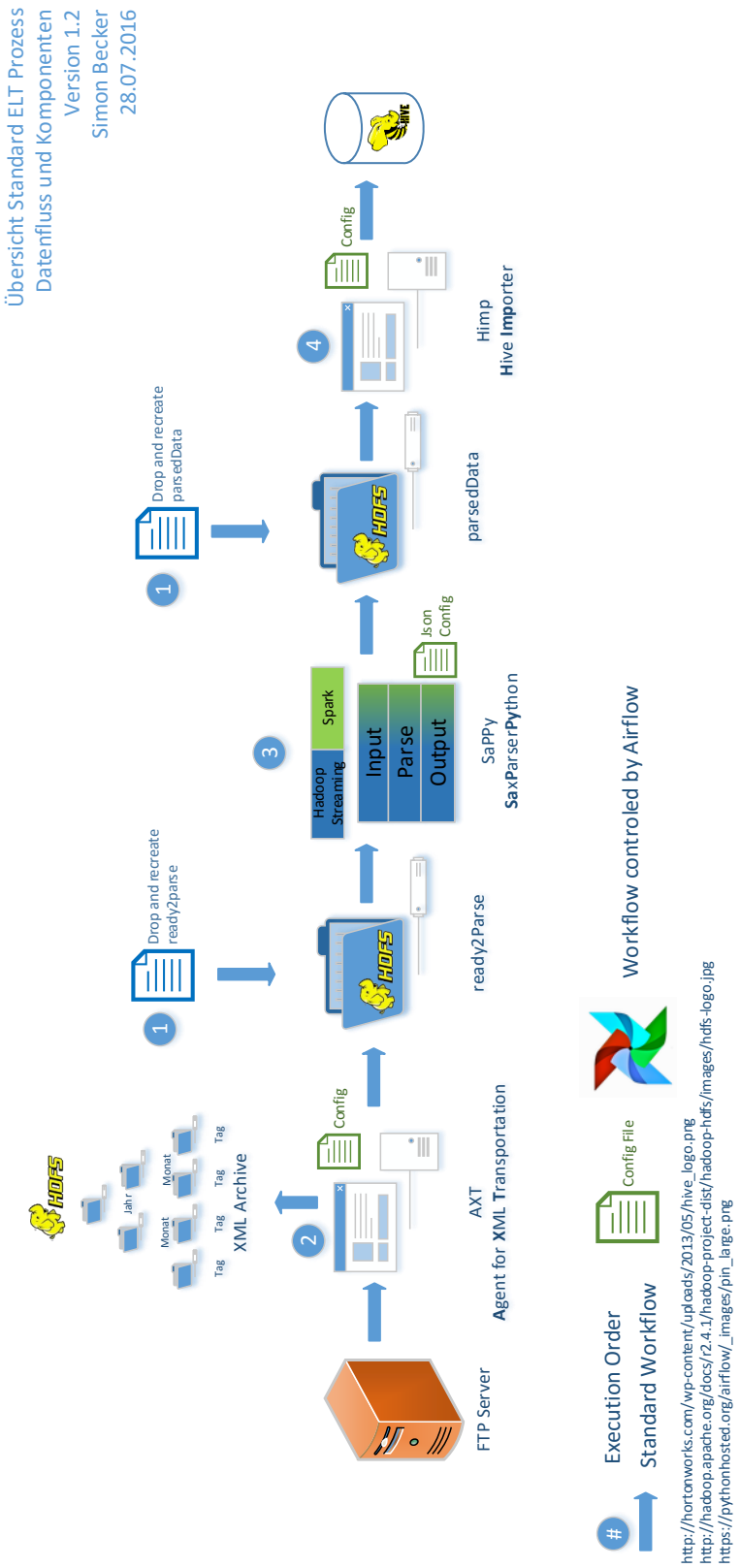


Abbildung 7.2.: Standard ELT Workflow

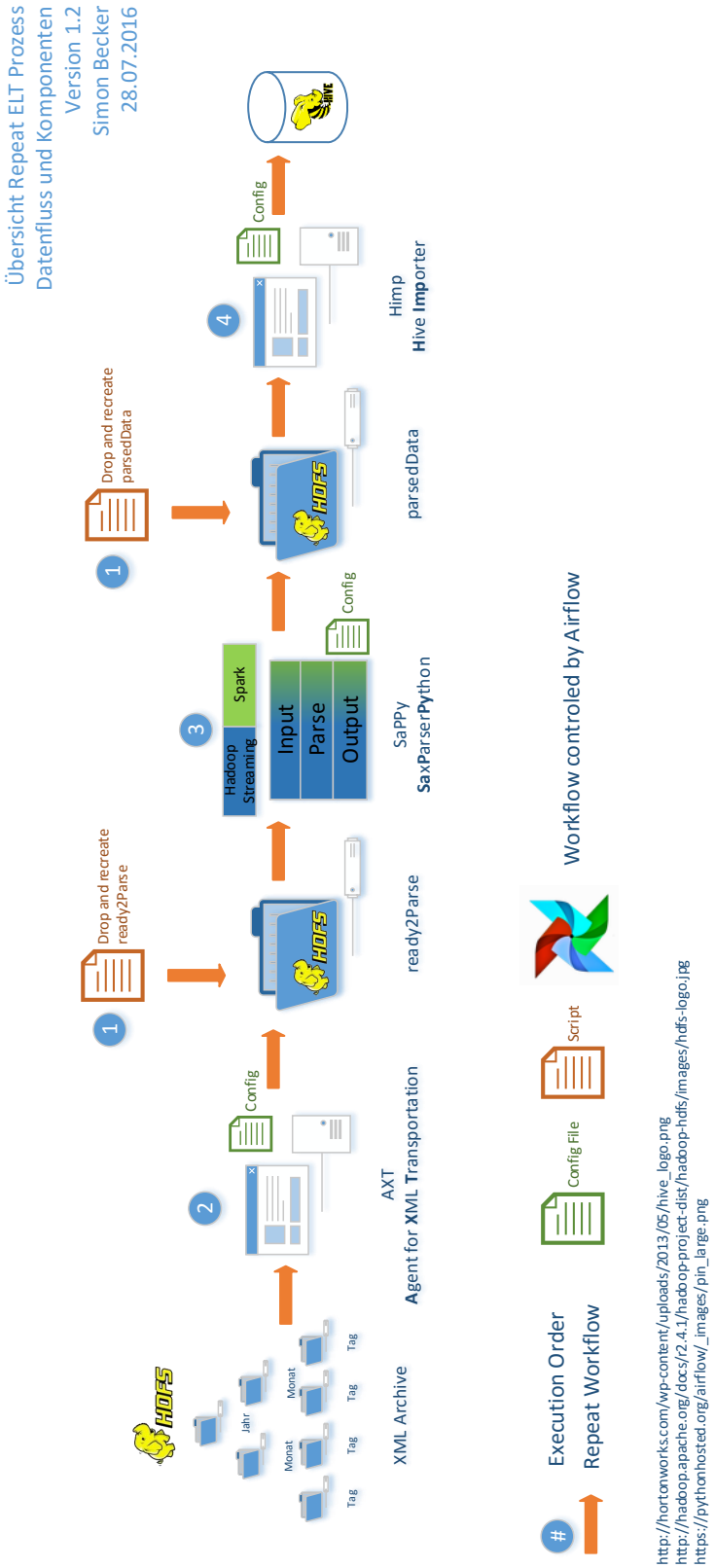


Abbildung 7.3.: Repeat ELT Workflow

8. Test & Evaluation

Bei der Abnahme eines Projektes sollten die Anforderungen des Kunden erfüllt worden sein und die Software entsprechend funktionieren. Hierbei kann das Testmanagement helfen, indem es durch verschiedene Arten von Tests aufweist, dass das System weitestgehend fehlerfrei funktioniert. Die unterschiedlichen Testarten sind im Folgenden kurz erläutert:

Komponententest Es wird eine einzelne Komponente des Systems getestet. Hierbei werden die Funktionalitäten der Komponente jeweils isoliert betrachtet, um auftretende Fehler leichter zuordnen zu können.

Integrationstest Das Zusammenspiel der verschiedenen Komponenten und deren gegenseitige Beeinflussung wird geprüft. Ziel dabei ist es, mögliche Fehler in den Schnittstellen aufzuweisen.

Systemtest Das System als Ganzes wird getestet. Hierdurch sollen die Vollständigkeit, Leistung, Sicherheit, Benutzbarkeit und Dokumentation des Systems kontrolliert werden.

Positivtest Es wird geprüft, ob die Funktionalität wie geplant realisiert worden ist und fehlerfrei durchgeführt werden kann.

Negativtest Es wird mit Absicht ein fehlerhafter Zustand hergestellt, um zu prüfen, ob die Komponente bzw. das System den Fehler erkennt und entsprechend abfängt, ohne dass das gesamte System abstürzt.

Es gilt jedoch zu beachten, dass ein vollständiges Testen des Systems nicht möglich ist. Wenn keine Fehler gefunden wurden, heißt das nicht automatisch, dass das System auch wirklich fehlerfrei und brauchbar ist. Das kann durch das Systemumfeld und dem Anwendungsbereich variieren.

Im Rahmen der Projektgruppe wird das Testmanagement hauptsächlich dafür genutzt, eine Qualitätssicherung für die in dem ELT-Prozess beteiligten Komponenten durchzuführen. Hier ist es besonders wichtig, dass dieser Prozess reibungslos funktioniert, da dies eine Kernfunktionalität des Projekts darstellt. Diese Form des Testens besteht im Projektrahmen hauptsächlich aus Komponententests, da die beteiligten Komponenten von der Projektgruppe selbst entwickelt worden sind. Zum Abschluss steht ein Integrationstest, welcher die Komponenten zusammenfasst.

Das Ziel der einzelnen Testfälle ist es, Fehler aufzudecken und Optimierungspotential aufzuweisen. Hierbei wurden die Negativ-Tests besonders genutzt, um die Robustheit der Tools zu überprüfen.

8.1. Anforderungen

Im Rahmen unserer Projektgruppe sollen die einzelnen Komponenten des ELT-Prozesses getestet werden. Dabei soll jede Komponente unabhängig von den anderen getestet werden können. Wichtig hierbei ist, dass neben der Funktionalität und Bedienbarkeit der Komponente auch überprüft wird, ob die Anforderungen an die Komponente erfüllt worden sind. Zusätzlich soll der gesamte Workflow im Rahmen eines Integrationstests geprüft werden.

Im Folgenden sind die groben Anforderungen an das Testmanagement definiert. Die einzelnen Testfälle und ihre Ziele sind in den jeweiligen Unterkapiteln näher beschrieben.

Kürzel	Beschreibung
TG5-01	Die Komponente AXT muss im Hinblick auf ihre Funktionalität getestet werden.
TG5-02	Die Komponente AXT muss von einem Außenstehenden anhand der Readme bedient werden können.
TG5-03	Die Komponente AXT muss alle komponentenspezifischen Anforderungen erfüllen.
TG5-04	Die Komponente XML-Parser muss im Hinblick auf ihre Funktionalität getestet werden.
TG5-05	Die Komponente XML-Parser muss von einem Außenstehenden anhand der Readme bedient werden können.
TG5-06	Die Komponente XML-Parser muss alle komponentenspezifischen Anforderungen erfüllen.
TG5-07	Die Komponente HIMP muss im Hinblick auf ihre Funktionalität getestet werden.
TG5-08	Die Komponente HIMP muss von einem Außenstehenden anhand der Readme bedient werden können.
TG5-09	Die Komponente HIMP muss alle komponentenspezifischen Anforderungen erfüllen.
TG5-10	Die Komponenten müssen zusammen in einem definierten Workflow interagieren.

Tabelle 8.1.: Anforderungen an das Testmanagement

8.2. Durchführung

Um eine strukturierte Vorgehensweise innerhalb des Testmanagements zu gewährleisten, wurde für jede Komponente ein Testdrehbuch mit verschiedenen Testfällen erstellt. Anhand der Testdrehbücher werden die Komponenten auf mögliche Testfälle geprüft. Das Vorgehen und die Ergebnisse wurden protokolliert.

Die Positivtests werden in einer Testumgebung der Alpha-Installation durchgeführt.

Für die Negativtests wird auf einem lokalen Gerät eine virtuelle Maschine aufgesetzt, welche innerhalb einer Hadoop-Sandbox die eigens entwickelten Komponenten installiert. Anschließend können hier problemlos Fehlerfälle künstlich erzeugt werden, ohne dass die anderen Tätigkeiten der Projektgruppe davon beeinträchtigt werden.

Sollte ein Test nicht erfolgreich sein, wird Rücksprache mit verantwortlichen Entwickler gehalten. Je nachdem, ob es sich in dem Fall um einen Fehler oder eine Erweiterung handelt, wird eine hierdurch entstehende Aufgabe priorisiert. Die Priorisierung hilft sowohl dem Entwickler als auch dem Tester, einen Überblick zu erhalten, welche Aufgaben zu welcher Zeit erledigt werden können. Nachdem die Implementierung erfolgt ist, wird der Testfall von einem Testbeauftragten erneut durchgeführt.

8.2.1. AXT

Die Eigenentwicklung AXT transportiert die XML-Dateien von einem FTP-Server in das HDFS. Hier werden die Daten aus Sicherheitsgründen redundant gespeichert. Als Parameter erhält AXT hierfür folgende Werte:

- Einen Dateipfad zur Konfigurationsdatei, die verwendet werden soll
- Ein Datum, für den Tag, der von dem FTP-Server gelesen werden soll
- ein Datum, für den Tag, unter dem die Dateien auf dem HDFS abgelegt werden sollen
- Kennzeichen: Standard- oder Repeat-Workflow
- Kennzeichen: Merge-Option
- Kennzeichen: Multithreading

Die Unterscheidung zwischen Standard- und Repeat-Workflow ist notwendig, da beim Standard-Workflow die Dateien von einem FTP-Server in das HDFS sowohl gesichert als auch gemergt und in den Zielordner abgelegt werden. Beim Repeat-Workflow werden die Dateien, die im HDFS nun abgelegt sind betrachtet. Der FTP-Server ist für den Repeat-Workflow nicht notwendig.

Es sind die Testfälle aus der Tabelle 8.2 durchgeführt worden.

Testfälle AXT				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
		Positiv-Tests		
P1	Dateiübertragung aus der Quellstruktur im FTP für YYYY-MM-DD ins HDFS prüfen	Die auf dem FTP-Server angelegten Ordnerstrukturen beinhalten für jeden Unterordner „Tag“ einzelne XML-Dateien	Auf dem FTP-Server in den zu testenden Ordner Dateien ablegen und diese im HDFS in der Zielstruktur aufsuchen	Die Testdaten aus der Quellstruktur sind in der Zielstruktur unter denselben Testordnern wiederzufinden
P2	Zielstruktur im HDFS für YYYY-MM-DD anlegen		Auf dem FTP-Server in den zu testenden Ordner Dateien ablegen und prüfen, ob die Zielstruktur im HDFS der Quellstruktur entspricht.	Die Dateien sind in der Ordnerstruktur YYYY-MM-DD / fileName im HDFS abgelegt.
P3	Dateien vom FTP werden im Zwischenordner zwischengespeichert	Zwischenordner in der Config-Datei richtig gepflegt	Während der Ausführung von AXT prüfen, ob Dateien im Zwischenordner abgelegt werden.	Dateien aus dem FTP sind in dem lokalen Zwischenordner gespeichert

Testfälle AXT				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
P4	Dateien werden ins HDFS übertragen	Config-Datei richtig gepflegt	Verbindung auf Hadoop Master, Ausführen von AXT über entsprechendes Statement	Im HDFS sind die Dateien in der gewünschten Ordnerstruktur abgelegt
P5	Prüfung der richtigen Anzahl der Threads	Beim Aufruf wird ein Parameter mitgegeben, beispielsweise die Zahl 5	AXT wird mit dem Parameter 5 ausgeführt	Unter dem Ordner tmp_download befinden sich 5 Ordner jeweils von 0-4, die basierend auf den Thread-Parameter erstellt werden
P6	Übertragung und Entpacken von ZML-Dateien	ZML-Dateien im Quellordner	Verbindung auf Hadoop Master, Ausführen von AXT über entsprechendes Statement	ZML-Dateien werden entpackt und als XML-Dateien im HDFS-Zielordner abgelegt
P7	Prüfen, ob nicht-XML-Datei übertragen wird	eine nicht-XML-Datei wird in den Testdatensatz integriert	Property einstellen auf „alle Dateiformate“; AXT ausführen	Es werden nicht-XML-Dateien übertragen

Testfälle AXT				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
P8	Quellcode-Kommentare prüfen	Kommentare müssen auf Englisch im Quellcode vorhanden sein	Quellcode aufrufen und Kommentare überprüfen nach sun checkstyle conventions	Nach Überprüfung wird festgestellt, dass die sun checkstyle conventions genutzt wurde
P9	Logging-Möglichkeit	Ordner für Logging und Berechtigungen zu schreiben	Prüfen, ob Log-Datei erstellt worden ist	Log-Dateien werden in die log4j.properties-Datei im Resource-Ordner der AXT geschrieben
P10	Output von AXT beim erfolgreichem/nicht erfolgreichem Durchlauf	AXT gibt einen Return Code zurück	AXT wird ausgeführt und am Ende beobachtet, was ausgegeben wird	Great Success oder Return Code -1
P11	Config-Datei vorhanden und entsprechend gepflegt		Prüfen, ob die Config-Datei gepflegt ist	Quell- und Zielordner in der Config-Datei gepflegt
P12	XML-Dateien werden aus dem HDFS gemergt (repeat-Workflow)	Es wird ein Ordner übergeben, der XML-Dateien enthält. Beim Ausführen der AXT den Repeat-Workflow als Parameter angeben	AXT wird im repeat-Modus ausgeführt	ein oder mehrere gemergte XML-Dateien im ready2parse-Ordner

Testfälle AXT				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
P13	Ordnerstruktur aus dem FTP ins HDFS kopieren (Jahr, Monat, Tag)	Es wird ein Ordner übergeben, der Unterordner enthält, welche XML-Dateien enthalten	AXT wird ausgeführt	Ordnerstruktur im HDFS wie auf FTP vorhanden
P14	XML-Dateien werden on the fly gemergt und in Ordner ready2parse abgelegt	Ordner ready2parse vorhanden	Im Ornder ready2parse schauen, ob die XML-Dateien gemergt sind	ein oder mehrere gemergte XML-Dateien im ready2parse-Ordner
		Negativ-Tests		
N1	FTP-Server zu Beginn nicht erreichbar	FTP-Server zu Beginn abgeschaltet	FTP-Server abschalten, anschließend AXT starten	Kontrollierter Programmabbruch mit folgendem Fehler: log server not available
N2	Fehlerhafte Config-Datei zum FTP-Server	Die Quelle in der Config-Datei wird bewusst falsch angegeben	Config-Datei zum FTP-Server ist falsch, AXT wird gestartet	Verbindung zum Server wird versucht herzustellen. Rückmeldung, dass der Server nicht erreichbar ist.

Testfälle AXT				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
N3	FTP-Server fällt mittendrin aus (FTP-Server schließen)	Der FTP-Server wird mittendrin abgeschaltet	AXT wird gestartet, mittendrin wird die Verbindung zum FTP-Server unterbrochen	Meldung, dass der FTP-Server nicht erreichbar ist
N4	HDFS wird abgeschaltet		HDFS-Server abschalten, anschließend AXT starten	Rückmeldung, dass der HDFS-Server ist nicht erreichbar
N5	HDFS wird in der Config-Datei falsch angegeben	Das Ziel in der Config-Datei wird bewusst falsch angegeben.	Property zum HDFS-Server ist falsch, AXT wird gestartet	Rückmeldung, dass der HDFS-Server ist nicht erreichbar
N6	Keine Berechtigung auf lokalen Ordner zum Zwischenspeichern	Job nicht als AXT-User ausführen	Ausführen von AXT mit einem falschen Benutzer	Es wird ein Berechtigungsproblem als Fehlermeldung angegeben
N7	Keine Berechtigung auf HDFS zum Ablegen	Job nicht als AXT-User ausführen	Ausführen von AXT mit einem falschen Benutzer	Fehlermeldung: Berechtigungsproblem

Testfälle AXT				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
N8	Prüfen, dass eine nicht-XML-Datei nicht übertragen wird	Es wird versucht eine nicht-XML-Datei zu übertragen, obwohl in der Config-Datei dieser Fall ausgeschlossen ist	Config-Datei Einstellen auf "nur XML"; AXT ausführen	Fehlermeldung, dass keine nicht-XML-Datei versucht wurde, zu übertragen
N9	keine XML-Files im HDFS (repeat-Workflow)	Zielordner angelegt und keine Dateien enthalten	AXT wird ausgeführt und erhält als Parameter einen Ordnerpfad, welcher leer ist.	Wenn nichts gemerget wird, kommt auch keine Datei. Es erscheint eine Information, dass der Zielordner leer ist
N10	Ordner ready2parse nicht vorhanden	Der Ordner ready2parse wird auf dem HDFS gelöscht	AXT wird ausgeführt	Ziel-Ordner für Output wird erstellt
N11	Mergen von nicht-XML-Dateien	Es wird versucht mehrere nicht-XML-Dateien zu einer Datei zu mergen	AXT wird ausgeführt und erhält als Parameter einen Ordnerpfad, welcher nicht-XML-Dateien enthält.	Fehlermeldung, dass versucht wird, nicht-XML-Dateien zu mergen

Tabelle 8.2.: Testdrehbuch für AXT

Im Rahmen der durchgeführten Tests ist festzuhalten, dass die Positivtests bis auf vier Testfälle (P5, P6, P8, P9) erfolgreich abgeschlossen wurden. Hierbei ist anzumerken, dass diese vier Testfälle in den Nachtests auch erfolgreich abgeschlossen wurden. Grund für die Nachtest waren unter anderem fehlende Implementierungen (P5 und P9), unvollständige Dokumentation des Quellcodes (P6) oder das Entdecken von Fehlern (P8).

Im Falle von P6 wurde der Entwickler bezüglich der Quellcode-Kommentare auf die Einhaltung von Code-Konventionen hingewiesen. In Nachtests wurde sichergestellt, dass die Hinweise in den Quellcode eingearbeitet wurden.

Im Falle von P8 wurde der Entwickler nach den ersten Tests darauf hingewiesen, dass die Quellcode-Kommentare nicht einheitlich wie andere Entwicklungen innerhalb der Projektgruppe gestaltet sind. Es wurde ebenfalls in JIRA ein Task eröffnet, der als Bug definiert wurde und dem Entwickler zugewiesen wurde. Nachdem dieser Bug behoben wurde, bekam der Tester vom Entwickler die Benachrichtigung und konnte im Nachtest diesen Testfall erfolgreich abschließen.

8.2.2. XML-Parser

Der XML-Parser dient dazu, die XML-Dateien in eine relational strukturierte Form zu bringen. Als Input erhält der Parser die XML-Dateien, welche bei der Bedienung des DFMs generiert und mittels AXT in Hadoop eingespielt werden. Der Output des Parsers wird mehrere CSV-Dateien, welche anschließend über HIMP in Hive importiert werden. Zum Ausführen des Parsers wird der Launcher gestartet. Dabei werden ihm folgende Parameter übergeben:

- Modus, welcher angibt, ob der Parser mit Hadoop Streaming oder Spark ausgeführt wird
- Parser, um zu bestimmen, ob das Modul SaPPy oder Minidom verwendet wird
- Input, welcher die XML-Dateien enthält
- Output, wo die CSV-Dateien abgelegt werden sollen

Zum Testen des Parsers wurden die in Tabelle 8.3 beschriebenen Testfälle durchgeführt.

Testfälle Parser				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
		Positiv-Tests Parser-Modul		
P1	Vollständigkeit	Es wird jedes Element und jedes Attribut der XML-Datei geparst.	Es wird geprüft, ob alle Elemente und Attribute der XML-Datei im Output enthalten sind.	Es sind alle Elemente und Attribute im Output enthalten.
P2	Reproduzierbarkeit	Anhand des Outputs kann die ursprüngliche XML-Datei erstellt werden.	Mit Hilfe des Parser-Outputs wird manuell eine XML-Datei erzeugt und mit der Original-Datei verglichen.	Die XML-Dateien stimmen überein.
P3	Config-Datei prüfen	SaPPy muss fertig gestellt sein, weil nur SaPPy eine Config-Datei verwendet.	Prüfen, ob die Config-Datei gepflegt ist.	Die Config-Datei definiert alle Tabellen, die später in Hive verwendet werden. Es ist definiert, aus welchen Elementen in der XML-Datei die jeweiligen Attribute gelesen werden.

Testfälle Parser				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
P4	Quellcode-Kommentare prüfen		Quellcode prüfen, ob Kommentare zum Ablauf des Programms vorhanden sind.	Kommentare vorhanden.
P5	Output-Vergleich: Minidom-Parser-Modul vs. SaPPy-Parser-Modul	SaPPy ist fertig gestellt	Es wird der Output von SaPPy mit dem Output von dem Minidom-Parser-Modul verglichen.	Beide Outputs sind identisch.
Positiv-Tests gemeinsamer Launcher				
P6	Quellcode-Kommentare prüfen		Quellcode prüfen, ob Kommentare zum Ablauf des Programms vorhanden sind.	Kommentare sind vorhanden.
P7	Readme-Datei prüfen		Prüfen, ob die Readme-Datei vorhanden un mithilfe von Kommentaren sprechend gepflegt ist.	Anleitung zum Ausführen des Parsers in Readme-Datei gepflegt.
Positiv-Tests Hadoop Streaming				

Testfälle Parser				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
P8	Quellcode-Kommentare prüfen		Quellcode prüfen, ob Kommentare zum Ablauf des Programms vorhanden sind.	Kommentare sind vorhanden.
P9	CSV-Ripper	Der Output von Hadoop Streaming muss noch mittels CSV-Ripper bearbeitet werden, damit die CSV-Dateien von HImp weiterverarbeitet werden können.	Den CSV-Ripper mit dem Output des Parsers (Modus: Hadoop Streaming) ausführen und anschließend prüfen, ob jetzt der Tabellenname aus der ersten Spalte entfernt wurde.	Die erste Spalte, welche den Tabellennamen enthielt, wurde entfernt.
Positiv-Tests Spark				
P10	Quellcode-Kommentare prüfen		Quellcode prüfen, ob Kommentare zum Ablauf des Programms vorhanden sind.	Kommentare sind vorhanden.
Negativ-Tests Parser-Modul				

Testfälle Parser				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
N1	XML-File fehlerhaft	Die XML-Datei ist an bestimmten Stellen fehlerhaft, d.h. es handelt sich um eine invalide XML-Datei.	Eine fehlerhafte XML-Datei wird an den Parser übergeben.	Der Parser gibt eine Fehlermeldung aus, dass die Datei fehlerhaft sei.
N2	Prüfen von nicht-XML-Dateien		Dem Parser wird eine nicht-XML-Datei übergeben.	Es gibt eine Meldung, dass die Datei nicht geparst werden kann.
N3	fehlerhafte Config-Datei	SaPPy muss fertig gestellt sein, da nur SaPPy eine Config-Datei verwendet.	Die verwendete Config-Datei ist fehlerhaft.	Es wird eine entsprechende Fehlermeldung ausgegeben.
		Negativ-Tests gemeinsamer Launcher		
N4	kein Input	Der Input-Ordner ist nicht vorhanden.	Beim Parameter für den Input wird ein nicht existierender Ordner angegeben.	Es wird gemeldet, dass kein Input vorhanden ist.
N5	kein Output	Der Output-Ordner ist nicht vorhanden.	Beim Parameter für den Output wird ein nicht existierender Ordner angegeben.	Der Output-Ordner wird automatisch angelegt.

Testfälle Parser				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
N6	falsche Parameter		Der Launcher wird mit falschen Parametern aufgerufen.	Es wird gemeldet, dass der Parameter falsch ist.
		Negativ-Tests Hadoop Streaming		
N7	Der Master wird abgeschaltet.		Es wird ein Stromausfall o.Ä. simuliert, indem der Master deaktiviert wird.	Es wird eine entsprechende Meldung erstellt.
N8	Es wird ein Slave-Node abgeschaltet.		Es wird ein Stromausfall o.Ä. simuliert, indem der Master deaktiviert wird.	Es wird eine entsprechende Meldung erstellt.
N9	Es werden 2 Slave-Nodes abgeschaltet.		Es wird ein Stromausfall o.Ä. simuliert, indem der Master deaktiviert wird.	Es wird eine entsprechende Meldung erstellt.
N10	Es werden 3 Slave-Nodes abgeschaltet.		Es wird ein Stromausfall o.Ä. simuliert, indem der Master deaktiviert wird.	Es wird eine entsprechende Meldung erstellt.

Testfälle Parser				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
N11	Es werden 4 Slave-Nodes abgeschaltet.		Es wird ein Stromausfall o.Ä. simuliert, indem der Master deaktiviert wird.	Es wird eine entsprechende Meldung erstellt.
		Negativ-Tests Spark		
N12	Der Master wird abgeschaltet.		Es wird ein Stromausfall o.Ä. simuliert, indem der Master deaktiviert wird.	Es wird eine entsprechende Meldung erstellt.
N13	Es wird ein Slave-Node abgeschaltet.		Es wird ein Stromausfall o.Ä. simuliert, indem der Master deaktiviert wird.	Es wird eine entsprechende Meldung erstellt.
N14	Es werden 2 Slave-Nodes abgeschaltet.		Es wird ein Stromausfall o.Ä. simuliert, indem der Master deaktiviert wird.	Es wird eine entsprechende Meldung erstellt.
N15	Es werden 3 Slave-Nodes abgeschaltet.		Es wird ein Stromausfall o.Ä. simuliert, indem der Master deaktiviert wird.	Es wird eine entsprechende Meldung erstellt.

Testfälle Parser				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
N16	Es werden 4 Slave-Nodes abgeschaltet.		Es wird ein Stromausfall o.Ä. simuliert, indem der Master deaktiviert wird.	Es wird eine entsprechende Meldung erstellt.

Tabelle 8.3.: Testdrehbuch des XML-Parsers

Die meisten Testfälle sind erfolgreich durchgeführt worden und lieferten das erwartete Ergebnis. Die Testfälle, bei denen das Ergebnis nicht den Erwartungen entsprach, sind im Folgenden erläutert.

Bei den Testfällen P1 und P2 wurde die XML-Datei nicht zu 100% geparst und konnte auch nicht identisch rekonstruiert werden. Das liegt daran, dass zu Beginn des Projekts die relevanten Elemente und Attribute der XML-Datei definiert worden sind. Die Ergebnisse hierzu wurden in einer XML-Dokumentation (Ausschnitt in A.1) zusammengefasst. Im weiteren Verlauf wurden nur die relevanten Elemente und Attribute fokussiert. Weitere Elemente und Attribute, die in der XML-Datei enthalten sind, werden beim Parsen nicht berücksichtigt. Wenn diese Inhalte jedoch benötigt sind, kann die Konfigurationsdatei von SaPPy diesbezüglich erweitert werden. Hierzu kam es auch im Rahmen der Testdurchführung häufiger, da die ersten Tests zum SaPPy-Output negativ waren und erst nach Ergänzungen durch den Entwickler der Testfall letztendlich positiv verlief.

Das Testen der SaPPy-Config-Datei bei den Testfällen P3 und N3 hat ergeben, dass bei falscher Schreibweise eines Elements dieses nicht geparst wird. Wenn also beispielsweise in der Konfigurationsdatei „videolist“ und in der XML-Datei das Element „videoList“ heißt, wird das Element nicht geparst. Hier kam es mehrmals zu Rücksprache mit den Entwicklern, um auch die letzten Unstimmigkeiten zu beseitigen. Wenn innerhalb der Konfigurationsdatei ein Attribut zu einer Tabelle zugeordnet wird, welche nicht zu Beginn der Konfigurationsdatei deklariert worden ist, wird eine Meldung in einer Fehlertabelle innerhalb des Outputs vom Parser ausgegeben.

8.2.3. HIMP

HIMP ist eine Eigenentwicklung und dient dem Importieren der vom XML-Parser erzeugten CSV-Dateien in Hive-Tabellen. Beim Testen von HIMP wurden die in Tabelle 8.4 beschriebenen Testfälle durchgeführt.

Testfälle HIMP				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
		Positiv-Tests		
P1	Testdurchlauf mit einer generierten Beispieldatei	Die Testdatei ist im CSV-Format und hat einen zur Hive-Tabelle passenden Namen wie z.B. session.csv.	Config-Datei auf Verzeichnis anpassen, in dem sich die Testdatei befindet, und HIMP ausführen.	In Hive wird die entsprechende Tabelle mit den Daten aus der Testdatei gefüllt.
P2	Namensgleichheit	Die Dateinamen der CSV-Dateien entsprechen den Namen der Tabellen in Hive.	Die Namen der Tabellen in Hive mit den Namen der CSV-Dateien vergleichen.	Die Namen sind gleich.
P3	Logging-Möglichkeit		HIMP wird ausgeführt. Anschließend wird geprüft, ob die Log-Datei mit Einträgen vorhanden ist.	Log-Datei mit Einträgen
P4	Quellcode-Kommentare prüfen		Quellcode prüfen, ob Kommentare zum Ablauf des Programms vorhanden sind.	Kommentare vorhanden.

Testfälle HIMP				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
P5	Config-Datei prüfen		Prüfen, ob die Config-Datei vorhanden ist und mithilfe von Kommentaren sprechend gepflegt ist.	Verwendete Parameter wie Quellverzeichnis und Ziel-Datenbank in Config-Datei gepflegt.
P6	Readme-Datei prüfen		Prüfen, ob die Readme-Datei vorhanden und mithilfe von Kommentaren sprechend gepflegt ist.	Anleitung zum Ausführen von HIMP in Readme-Datei gepflegt.
		Negativ-Tests		
N1	Kein Input	Keine Daten im Quellverzeichnis vorhanden.	Parameter für Quellverzeichnis auf leeres Verzeichnis einstellen und HIMP ausführen.	HIMP meldet, dass kein Input vorhanden ist.
N2	Hive-Server nicht erreichbar		Hive-Server mittels Ambari abschalten und HIMP ausführen.	HIMP meldet, dass Hive nicht erreichbar ist.

Testfälle HIMP				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
N3	Quellverzeichnis ist nicht vorhanden		In der Config-Datei für das Quellverzeichnis einen Pfad angeben, welcher nicht existiert. HIMP ausführen.	HIMP meldet, dass das Quellverzeichnis nicht vorhanden ist.
N4	Testen des Programms auf äußere Einflüsse	HIMP ist gestartet und der Prozess läuft	Während der Prozess läuft, wird die Sandbox beendet.	Nachdem die Sandbox neu gestartet wird, führt das Programm den Prozess fort.

Tabelle 8.4.: Testdrehbuch für HIMP

Alle Testfälle konnten erfolgreich abgeschlossen werden. Die eigens entwickelte Software HIMP agierte in allen Fällen wie erwartet. Qualitativ konnte auch eine gewünschte Dokumentation des Codes und die korrekte Fehlerbehandlung festgestellt werden.

8.2.4. Integrationstest

In den vorherigen Abschnitten wurden die Komponententests zu den einzelnen Tools wie AXT, XML-Parser und HIMP detailliert getestet und dokumentiert. Dadurch, dass alle Funktionalitäten der einzelnen Tools getestet wurden, wurde somit auch das Zusammenspiel der Tools untereinander getestet und dokumentiert.

In diesem Abschnitt wird beschrieben, wie der gesamte ELT-Prozess mit Hilfe von Airflow getestet wird. Testdaten werden generiert und es wird anhand der Ergebnisse nach Ablauf des ELT-Prozesses verglichen und analysiert, ob ein erzeugter Testdatensatz in der Zielstruktur integriert ist.

Hierzu wurden folgende Testfälle betrachtet:

Testfälle Integrationstest				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
		Positiv-Tests		
P1	ELT-Standard-Workflow mit Hadoop Streaming (Sappy)	Workflow ist automatisiert	Workflow wird mittels Airflow gestartet	Workflow läuft erfolgreich durch
P2	ELT-Standard-Workflow mit Spark (Sappy)	Workflow ist automatisiert	Workflow wird mittels Airflow gestartet	Workflow läuft erfolgreich durch
P3	ELT-Repeat-Workflow mit Hadoop Streaming (Sappy)	Workflow ist automatisiert	Workflow wird mittels Airflow gestartet	Workflow läuft erfolgreich durch
P4	ELT-Repeat-Workflow mit Spark (Sappy)	Workflow ist automatisiert	Workflow wird mittels Airflow gestartet	Workflow läuft erfolgreich durch
P5	Log-Files		Prüfen, ob Log-Files der einzelnen Komponenten von Airflow aufgerufen werden können.	Log-Files sind zugänglich
P6	automatisierte Wiederholung des einzelnen Tools AXT (erster Schritt in ELT-Workflow)	Workflow ist automatisiert	Der Workflow soll täglich laufen. Überprüfen, ob Airflow den Workflow täglich startet	Workflow wird täglich von Airflow gestartet

Testfälle Integrationstest				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
P7	automatisierte Wiederholung des Workflows	Workflow ist automatisiert	Der Workflow soll täglich laufen. Überprüfen, ob Airflow den Workflow täglich startet	Workflow wird täglich von Airflow gestartet
			Negativ-Tests	
N1	Standard-Workflow: AXT vor Start abschalten	Workflow ist automatisiert	Workflow wird mittels Airflow gestartet. AXT ist jedoch nicht erreichbar.	Es wird eine Meldung ausgegeben, dass der Workflow nicht durchlaufen werden konnte. Diese Meldung ist auch im Log zu finden.
N2	Standard-Workflow: Spark (Sappy) vor Start abschalten	Workflow ist automatisiert	Workflow wird mittels Airflow gestartet. Spark (Sappy) ist jedoch nicht erreichbar.	Es wird eine Meldung ausgegeben, dass der Workflow nicht durchlaufen werden konnte. Diese Meldung ist auch im Log zu finden.

Testfälle Integrationstest				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
N3	Standard-Workflow: Hadoop Streaming (Sappy) vor Start abschalten	Workflow ist automatisiert	Workflow wird mittels Airflow gestartet. Hadoop Streaming (Sappy) ist jedoch nicht erreichbar.	Es wird eine Meldung ausgegeben, dass der Workflow nicht durchlaufen werden konnte. Diese Meldung ist auch im Log zu finden.
N4	Standard-Workflow: HIMP vor Start abschalten	Workflow ist automatisiert	Workflow wird mittels Airflow gestartet. HIMP ist jedoch nicht erreichbar.	Es wird eine Meldung ausgegeben, dass der Workflow nicht durchlaufen werden konnte. Diese Meldung ist auch im Log zu finden.
N5	Standard-Workflow: AXT nach Start abschalten	Workflow ist automatisiert	Workflow wird mittels Airflow gestartet. AXT wird mittendrin abgeschaltet.	Es wird eine Meldung ausgegeben, dass der Workflow nicht durchlaufen werden konnte. Diese Meldung ist auch im Log zu finden.

Testfälle Integrationstest				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
N6	Standard-Workflow: Spark (Sappy) nach AXT abschalten	Workflow ist automatisiert	Workflow wird mittels Airflow gestartet. Spark (Sappy) wird mittendrin abgeschaltet.	Es wird eine Meldung ausgegeben, dass der Workflow nicht durchlaufen werden konnte. Diese Meldung ist auch im Log zu finden.
N7	Standard-Workflow: Hadoop Streaming (Sappy) nach AXT abschalten	Workflow ist automatisiert	Workflow wird mittels Airflow gestartet. Hadoop Streaming (Sappy) wird mittendrin abgeschaltet.	Es wird eine Meldung ausgegeben, dass der Workflow nicht durchlaufen werden konnte. Diese Meldung ist auch im Log zu finden.
N8	Standard-Workflow: HIMP nach Hadoop Streaming (Sappy) abschalten	Workflow ist automatisiert	Workflow wird mittels Airflow gestartet. HIMP wird mittendrin abgeschaltet.	Es wird eine Meldung ausgegeben, dass der Workflow nicht durchlaufen werden konnte. Diese Meldung ist auch im Log zu finden.

Testfälle Integrationstest				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
N9	Standard-Workflow: HIMP nach Spark (Sappy) abschalten	Workflow ist automatisiert	Workflow wird mittels Airflow gestartet. HIMP wird mittendrin abgeschaltet.	Es wird eine Meldung ausgegeben, dass der Workflow nicht durchlaufen werden konnte. Diese Meldung ist auch im Log zu finden.
N10	Repeat-Workflow: AXT vor Start abschalten	Workflow ist automatisiert	Workflow wird mittels Airflow gestartet. AXT ist jedoch nicht erreichbar.	Es wird eine Meldung ausgegeben, dass der Workflow nicht durchlaufen werden konnte. Diese Meldung ist auch im Log zu finden.
N11	Repeat-Workflow: Hadoop Streaming (Sappy) vor Start abschalten	Workflow ist automatisiert	Workflow wird mittels Airflow gestartet. Hadoop Streaming (Sappy) ist jedoch nicht erreichbar.	Es wird eine Meldung ausgegeben, dass der Workflow nicht durchlaufen werden konnte. Diese Meldung ist auch im Log zu finden.

Testfälle Integrationstest				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
N12	Repeat-Workflow: Spark (Sappy) vor Start abschalten	Workflow ist automatisiert	Workflow wird mittels Airflow gestartet. Spark (Sappy) wird mittendrin abgeschaltet.	Es wird eine Meldung ausgegeben, dass der Workflow nicht durchlaufen werden konnte. Diese Meldung ist auch im Log zu finden.
N13	Repeat-Workflow: HIMP vor Start abschalten	Workflow ist automatisiert	Workflow wird mittels Airflow gestartet. HIMP ist jedoch nicht erreichbar.	Es wird eine Meldung ausgegeben, dass der Workflow nicht durchlaufen werden konnte. Diese Meldung ist auch im Log zu finden.
N14	Repeat-Workflow: AXT nach Start abschalten	Workflow ist automatisiert	Workflow wird mittels Airflow gestartet. AXT wird mittendrin abgeschaltet.	Es wird eine Meldung ausgegeben, dass der Workflow nicht durchlaufen werden konnte. Diese Meldung ist auch im Log zu finden.

Testfälle Integrationstest				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
N15	Repeat-Workflow: Hadoop Streaming (Sappy) nach Start abschalten	Workflow ist automatisiert	Workflow wird mittels Airflow gestartet. Hadoop Streaming (Sappy) wird mittendrin abgeschaltet.	Es wird eine Meldung ausgegeben, dass der Workflow nicht durchlaufen werden konnte. Diese Meldung ist auch im Log zu finden.
N16	Repeat-Workflow: Spark (Sappy) nach Start abschalten	Workflow ist automatisiert	Workflow wird mittels Airflow gestartet. Spark (Sappy) wird mittendrin abgeschaltet.	Es wird eine Meldung ausgegeben, dass der Workflow nicht durchlaufen werden konnte. Diese Meldung ist auch im Log zu finden.
N17	Repeat-Workflow: HIMP nach Hadoop Streaming (Sappy) abschalten	Workflow ist automatisiert	Workflow wird mittels Airflow gestartet. HIMP ist jedoch nicht erreichbar.	Es wird eine Meldung ausgegeben, dass der Workflow nicht durchlaufen werden konnte. Diese Meldung ist auch im Log zu finden.

Testfälle Integrationstest				
Nr	**Testfallbeschreibung**	**Voraussetzungen**	**Vorgehensbeschreibung**	**erwartetes Ergebnis**
N18	Repeat-Workflow: HIMP nach Spark (Sappy) abschalten	Workflow ist automatisiert	Workflow wird mittels Airflow gestartet. HIMP ist jedoch nicht erreichbar.	Es wird eine Meldung ausgegeben, dass der Workflow nicht durchlaufen werden konnte. Diese Meldung ist auch im Log zu finden.
N19	Airflow Rückmeldung bei Internetabbruch	Workflow ist automatisiert	Nach dem Starten des Workflows die Internetverbindung unterbrechen	Es wird eine Fehlermeldung ausgegeben.
N20	E-Mail-Benachrichtigung bei Fehler	E-Mail-Funktion eingeschaltet	E-Mails kontrollieren	Es trifft eine E-Mail-Benachrichtigungen ein.

Tabelle 8.5.: Testdrehbuch des Integrationstests

Der gesamte Integrationstest, sowohl die Positivtests als auch die Negativtests, verliefen erfolgreich. Das Vorgehen wurde mit dem Entwickler gemeinsam abgestimmt und dementsprechend auch dokumentiert.

Bei den Positivtests handelt es sich um die Testfälle P6 und P7, in denen die Automatisierung des Workflows zu gegebenen Zeitabständen getestet werden sollte.

Im Falle der Negativtests ist es Testfall N19, der nicht weiter beachtet wurde. Der Testfall endete mit einem unerwarteten Fehler, welcher hierauf folgend durch den Entwickler behoben wurde.

Zusammengefasst ist zu sagen, dass der Integrationstest erfolgreich verlaufen ist. Die erfolgreiche Testfalldurchführung im Rahmen der einzelnen Tools AXT, Parser und HIMP hat sich auf den Integrationstest widergespiegelt.

8.3. Fazit und Evaluation

Durch das Testmanagement konnten einige Fehler aufgedeckt und behoben worden, wodurch die Qualität und Stabilität insgesamt erhöht wurden. Die in Kapitel 8.1 beschriebenen Anforderungen wurden erfüllt.

Zusammengefasst ist das Kapitel Testmanagement erfolgreich abgeschlossen worden. Dadurch, dass die Tests der einzelnen Tools erfolgreich verlaufen sind, wirkte sich dieses auch positiv auf den Integrationstest aus. Das Zusammenspiel der einzelnen Tools im ELT-Prozess konnte somit zu positiver Erkenntnis gebracht werden.

9. Fazit

Die Projektgruppe konnte das Hadoop Ecosystem für das gestellte Anwendungsszenario von CEWE erfolgreich einsetzen. Basierend auf dem Hadoop Ecosystem wurde ein ELT-Prozess entwickelt, der die XML-Dateien täglich verarbeitet. Dabei werden die XML-Dateien von einem FTP-Server in das HDFS importiert. Anschließend werden die XML-Dateien geparst und die relevanten Informationen strukturiert in einem Hive-Datenbankschema persistiert. Weiterhin hat die Projektgruppe mit ersten explorativen und deskriptiven Analysen das Erkenntnis-Potenzial der Daten/Informationen aufgezeigt. Die Ziele der Projektgruppe wurden erreicht. Der Unternehmenspartner CEWE hat von der Projektgruppe aufgezeigt bekommen, dass der Einsatz des Hadoop Ecosystem für ihr gestelltes Anwendungsszenario funktional und praktikabel ist. Zum Abschluss des Projektes wurden alle benötigten Softwarekomponenten für den ELT-Prozess in die Systemlandschaft des Projektpartners migriert, um diesem die Ergebnisse weitere Verwendung und Fortentwicklung zur Verfügung zu stellen.

Die gezielte und strukturierte Vorgehensweise half, das Projekt erfolgreich umzusetzen. Ohne Vorwissen über Apache Hadoop behandelte jedes Projektmitglied ein fachbezogenes, relevantes Thema im Zuge einer Seminararbeit und gewann Verständnis über dieses und die Arbeitsweise von Apache Hadoop. Anschließend wurde ein gemeinsames Zielbild entwickelt. Ein ELT-Workflow, welcher zuverlässig die Informationen aus den Daten der DFM für Analysen bereitstellt. Hierfür wurden vorerst bestehende Softwarekomponenten des Hadoop Ecosystems anhand vorher aufgestellter Anforderungen evaluiert. Bestimmte Aufgaben wie beispielsweise der Import der XML-Dateien in das HDFS konnten nicht von bestehenden Tools erledigt werden. An dieser Stelle wurden Eigenentwicklungen der Projektgruppe implementiert.

Neben den erfolgreich erreichten sachlichen und technischen Zielen, hat das Projekt für die Stakeholder und Projektgruppenmitglieder weitere Mehrwerte geschaffen. Das einjährige Projekt hat allen Beteiligten Wissen über eine neue Technologie verschafft. Ein weiterer wichtiger Mehrwert ist die Weiterentwicklung von Kompetenzen der Projektgruppenmitglieder. Während des Projektes lernten die Projektgruppenmitglieder mit einem Unternehmen als „Kunden“ umzugehen, ein Jahr in einem zwölfköpfigen Team zu arbeiten, zu diskutieren, sich abzustimmen,

Aufgaben zu koordinieren oder eigene Ideen durchzusetzen und andere von diesen Ideen zu überzeugen und zu begeistern. Förderer der Kompetenzentwicklung waren die Projektbetreuer und der Unternehmenspartner in Form von Eugen Neigel und Fatih-Mehmet Inel. Durch regelmäßige und intensive Kommunikation mit den Projektbetreuern der Universität und den Projektpartnern CEWE konnte so ein gelungenes Ergebnis für alle Beteiligten erreicht werden.

10. Verwendete Software und Bibliotheken

Apache Airflow Airflow ist eine Python-basierte Plattform, welche die Erstellung, Verwaltung und Überwachung von Data Pipelines ermöglicht.

Author	Apache Software Foundation
Lizenz	Apache License 2.0
Webseite	`https://airflow.incubator.apache.org/`

Apache Commons IO Apache Commons IO ist eine Bibliothek, welche Entwicklern bei der Erstellung von Input-Output Anwendungen unterstützt.

Author	Apache Software Foundation
Lizenz	Apache License 2.0
Webseite	`https://commons.apache.org/proper/commons-io/`

Apache Flume Apache Flume ist eine Softwarelösung für die Sammlung, Aggregation und den Transport von Logdaten.

Author	Apache Software Foundation
Lizenz	Apache License 2.0
Webseite	`https://flume.apache.org/`

Apache Hadoop Apache Hadoop ist ein java-basiertes Framework für skalierbare, verteilt arbeitende Software. Es basiert auf dem MapReduce-Algorithmus von Google Inc. sowie auf Vorschlägen des Google-Dateisystems und ermöglicht die Durchführung intensiver Rechenprozesse mit großen Datenmengen auf Computerclustern.

Author	Apache Software Foundation
Lizenz	Open Source
Webseite	`http://hadoop.apache.org/`

Apache Hive Apache Hive ist eine Data-Warehouse Software für strukturierte Speicherung großer Datenmengen und die Verarbeitung von SQL-ähnlichen

Abfragen über ein Computercluster mit Hilfe des MapReduce Algorithmus.

Author Apache Software Foundation

Lizenz Open Source

Webseite `http://hive.apache.org/`

Apache Kafka Apache Kafka ist eine Software, welche der Verarbeitung von Datenströmen dient.

Author Apache Software Foundation

Lizenz Apache Liccense 2.0

Webseite `http://kafka.apache.org/`

Apache Log4j 2 Apache Log4j 2 ist eine Bibliothek, welche Entwicklern beim Logging der Prozesse ihrer Programme unterstützt.

Author Apache Software Foundation

Lizenz Open Source

Webseite `http://logging.apache.org/log4j/`

Apache Pig Apache Pig ist eine Plattform für die Analyse großer Datenmengen mit Hilfe einer eigens entworfenen Sprache.

Author Apache Software Foundation

Lizenz Apache License 2.0

Webseite `https://pig.apache.org/`

Apache Spark Apache Spark ist eine Softwarelösung für die Durchführung von Clustercomputing.

Author Apache Software Foundation

Lizenz Apache License 2.0

Webseite `http://spark.apache.org/`

Apache Storm Apache Storm ist eine Software, welche der Verarbeitung von Datenströmen dient.

Author Apache Software Foundation

Lizenz Apache License 2.0

Webseite `http://storm.apache.org/`

Apache Sqoop Apache Sqoop ist ein Tool zur Übertragung von Daten zwischen relationalen Datenbankmanagementsystemen und dem Hadoop Ecosystem.

Author Apache Software Foundation

Lizenz Open Source

Webseite `http://http://sqoop.apache.org/`

Apache Zeppelin Apache Zeppelin ist eine webbasierte Software zur Analyse und Darstellung von Daten.

Author Apache Software Foundation

Lizenz Apache License 2.0

Webseite `https://zeppelin.apache.org/`

Azkaban Airflow ist eine Plattform, welche die Erstellung, Verwaltung und Überwachung von Data Pipelines ermöglicht.

Author u.a. Ray Yang, Evan Li, George Zhang

Lizenz Apache License 2.0

Webseite `https://azkaban.github.io/`

Fluentd Fluentd ist ein Ruby-basierendes Framework für die Sammlung von Daten.

Author Treasure Data

Lizenz opensource under Apache 2.0

Webseite `http://www.fluentd.org/`

Gobblin Gobblin ist eine Softwarelösung für die Transportation von Daten aus verschiedenen Quellen in ein Hadoop-System.

Author u.a. Abhishek Tiwari, Chavdar Botev

Lizenz Apache License 2.0

Webseite `https://github.com/linkedin/gobblin`

Hibernate Hibernate ist ein Java-Framework, welches die objektrelationale Abbildung von Daten ermöglicht.

Author JBoss (Red Hat)

Lizenz LGPL

Webseite `http://hibernate.org/`

JUnit JUnit ist ein Framework für die Testautomatisierung von Java-Programmen.

Author	u. a. Kent Beck und Erich Gamma
Lizenz	Eclipse Public License 1.0
Webseite	`http://junit.org/junit4/`

Logstash Logstash ist ein Tool zur Verwaltung von Events und Logs.

Author	Elastic
Lizenz	Apache License 2.0
Webseite	`https://www.elastic.co/products/logstash`

MockFtpServer MockFtpServer ist ein Java-basiertes Framework zum Testen von Verbindungen zu FTP-Servern.

Lizenz	Apache License 2.0
Webseite	`http://mockftpserver.sourceforge.net/`

MockFtpServer MockFtpServer ist ein Java-basiertes Framework zum Testen von Verbindungen zu FTP-Servern.

Lizenz	Apache License 2.0
Webseite	`http://mockftpserver.sourceforge.net/`

MySQL MySQL ist ein Datenbankverwaltungssystem.

Author	Oracle Corporation, MySQL AB, Sun Microsystems
Lizenz	Duales Lizenzsystem (Proprietär und GPL)
Webseite	`https://www.mysql.com/`

JFreeChart Die JFreeChart-API ist eine Java-basierte freie Bibliothek zur Generierung verschiedener Diagrammarten.

Author	Object Refinery Limited
Lizenz	GNU Lesser General Public Licence (LGPL)
Webseite	`http://www.jfree.org/jfreechart/`

xml.sax Die Simple API for XML ist ein Python-basiertes Framework zum Event-basierten lesen von XML-Daten.

Author	David Megginson
Lizenz	Public Domain
Webseite	`http://sax.sourceforge.net/`

Index

Dieses Index zeigt die tabellarische Aufzählung möglichst aller wichtiger Einheiten dieser Projektdokumentation mit Hinweis auf ihre Einordnung und den Orten ihrer Erwähnungen.

Literatur

[Apa11] APACHE SOFTWARE FOUNDATION: *Storm Github Repo*. https://github.com/apache/storm, 2011. – letzter Zugriff am 14. März 2016

[Apa13a] APACHE HADOOP ; HADOOP, Apache (Hrsg.): *C API libhdfs*. https://hadoop.apache.org/docs/r1.0.4/libhdfs.html, 2013. – letzter Zugriff am 09. September 2016

[Apa13b] APACHE HADOOP ; HADOOP, Apache (Hrsg.): *WebHDFS REST API*. http://hadoop.apache.org/docs/current/hadoop-streaming/HadoopStreaming.html, 2013. – letzter Zugriff am 09. September 2016

[Apa13c] APACHE SOFTWARE FOUNDATION: *Installation*. https://sqoop.apache.org/docs/1.99.3/Installation.html, 2013. – letzter Zugriff am 27. April 2016

[Apa14a] APACHE SOFTWARE FOUNDATION: *Apache Hive*. https://hive.apache.org/, 2014. – letzter Zugriff am 14. März 2016

[Apa14b] APACHE SOFTWARE FOUNDATION: *Apache Hive LanguageManual*. https://cwiki.apache.org/confluence/display/Hive/LanguageManual, 2014. – letzter Zugriff am 14. März 2016

[Apa14c] APACHE SOFTWARE FOUNDATION: *Concepts*. http://storm.apache.org/documentation/Concepts.html, 2014. – letzter Zugriff am 14. März 2016

[Apa14d] APACHE SOFTWARE FOUNDATION: *Project Information*. http://storm.apache.org/about/simple-api.html, 2014. – letzter Zugriff am 14. März 2016

[Apa14e] APACHE SOFTWARE FOUNDATION: *Project Information*. http://storm.apache.org/about/multi-language.html, 2014. – letzter Zugriff am 14. März 2016

[Apa15a] APACHE SOFTWARE FOUNDATION: *Apache Spark*. http://spark.apache.org/, 2015. – letzter Zugriff am 14. März 2016

[Apa15b] APACHE SOFTWARE FOUNDATION: *Apache Spark Data Frame*. http://spark.apache.org/docs/latest/sql-programming-guide.html, 2015. – letzter Zugriff am 10. September 2016

[Apa15c] APACHE SOFTWARE FOUNDATION: *Apache Spark MLlib*. http://spark.apache.org/mllib/, 2015. – letzter Zugriff am 10. September 2016

[Apa15d] APACHE SOFTWARE FOUNDATION: *Apache Spark MLlib Guide*. http://spark.apache.org/docs/latest/mllib-guide.html, 2015. – letzter Zugriff am 10. September 2016

[Apa15e] APACHE SOFTWARE FOUNDATION: *Storm guarantees every tuple will be fully processed*. http://storm.apache.org/about/guarantees-data-processing.html, 2015. – letzter Zugriff am 14. März 2016

[Apa15f] APACHE SOFTWARE FOUNDATION: *Storm is fault-tolerant*. http://storm.apache.org/about/fault-tolerant.html, 2015. – letzter Zugriff am 14. März 2016

[Apa15g] APACHE SOFTWARE FOUNDATION: *Why use Storm?* http://storm.apache.org/, 2015. – letzter Zugriff am 07. März 2016

[Apa16a] APACHE AIRFLOW ; AIRFLOW, Apache (Hrsg.): *Apache Airflow (incubating) Documentation*. https://airflow.incubator.apache.org/, 2016. – letzter Zugriff am 13. September 2016

[Apa16b] APACHE AIRFLOW ; AIRFLOW, Apache (Hrsg.): *API Reference*. https://airflow.incubator.apache.org/code.html, 2016. – letzter Zugriff am 13. September 2016

[Apa16c] APACHE AIRFLOW ; AIRFLOW, Apache (Hrsg.): *Concepts*. https://airflow.incubator.apache.org/concepts.html, 2016. – letzter Zugriff am 13. September 2016

[Apa16d] APACHE HADOOP ; HADOOP, Apache (Hrsg.): *Apache Hadoop Main 2.7.2 API*. https://hadoop.apache.org/docs/r2.7.2/api/, 2016. – letzter Zugriff am 09. September 2016

[Apa16e] APACHE SOFTWARE FOUNDATION: *Apache Pig*. http://pig.apache.org/, 2016. – letzter Zugriff am 24. September 2016

[Apa16f] APACHE SOFTWARE FOUNDATION: *Apache Tez*. https://tez.apache.org/, 2016. – letzter Zugriff am 14. März 2016

[Apa16g] APACHE SOFTWARE FOUNDATION: *Azkaban 3.0 Documentation*. http://azkaban.github.io/azkaban/docs/latest/, 2016. – letzter Zugriff am 24. September 2016

[Apa16h] APACHE SOFTWARE FOUNDATION ; FOUNDATION, Apache S. (Hrsg.): *Hadoop Streaming*. http://hadoop.apache.org/docs/current/hadoop-streaming/HadoopStreaming.html, 2016. – letzter Zugriff am 15. März 2016

[Apa16i] APACHE SOFTWARE FOUNDATION: *LanguageManual DML*. https://cwiki.apache.org/confluence/display/Hive/LanguageManualDML, 2016. – letzter Zugriff am 15. März 2016

[Apa16j] APACHE SOFTWARE FOUNDATION: *Sqoop User Guide (v1.4.3)*. https://sqoop.apache.org/docs/1.4.3/SqoopUserGuide.html, 2016. – letzter Zugriff am 20. April 2016

[Apa16k] APACHE SPARK ; SPARK, Apache (Hrsg.): *Apache Spark Documentation*. http://spark.apache.org/docs/latest/programming-guide.html, 2016. – letzter Zugriff am 11. September 2016

[Apa16l] APACHE STORM ; STORM, Apache (Hrsg.): *Setting up a Storm Cluster*. http://storm.apache.org/releases/1.0.1/Setting-up-a-Storm-cluster.html, 2016. – letzter Zugriff am 13. September 2016

[Arv11] ARVIND PRABHAKAR: *Apache Flume – Architecture of Flume NG*. http://blog.cloudera.com/blog/2011/12/apache-flume-architecture-of-flume-ng-2/, 2011. – letzter Zugriff am 01. April 2016

[Atl15] ATLASSIAN CORPORATION PLC: *LanguageManual*. https://cwiki.apache.org/confluence/display/Hive/LanguageManual, Oktober 2015. – letzter Zugriff am 11. September 2016

[BDS15] BALLERSTEDT, Dries ; DITTMER, Dr. C. ; SCHULZ, Peter: Modernisierung klassischer BI-Architekturen durch Big Data: Fluch und Segen zugleich. In: *BI-Spektrum* 2015 (2015), Nr. 1, S. 12–15

[Beh14] BEHREND, Oliver: *Analytik OSF - Datenquellen und Kennzahlen -Status und Potenziale*. Interne CEWE-Präsentation, 04.11.2014, 2014. – letzter Zugriff am 01. September 2016

[Bit12] BITKOM: *Big Data im Praxiseinsatz – Szenarien, Beispiele, Effekte*. 2012

[Bit14] BITKOM: *Big-Data-Technologien - Wissen für Entscheider: Leitfaden*. 2014

[BNIK15] BEHREND, Oliver ; NEIGEL, Eugen ; INEL, Fatih-Mehmet ; KERSKEN, Matthias: *Projektgruppe Hadoop - Ersttermin bei CEWE*. Interne CEWE-Präsentation, 13.10.2015, 2015. – letzter Zugriff am 01. September 2016

[Bra14] BRATUS, Andrii: *RefineOnSpark: a simple and scalable ETL based on Apache Spark and OpenRefine*, UNIVERSITY OF TRENTO, DEPARTMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE, Diplomarbeit, 2014

[CEW16] CEWE STIFTUNG & CO. KGAA: *Geschäftsbericht 2015*. http://ir.cewe.de/download/companies/cewe/Annual%20Reports/DE0005403901-JA-2015-EQ-D-00.pdf, 2016. – letzter Zugriff am 6. April 2016

[Clo16a] CLOUDERA INC.: *Feature Differences - Sqoop 1 and Sqoop 2*. http://www.cloudera.com/documentation/enterprise/5-4-x/topics/cdh_ig_sqoop_vs_sqoop2.html, 2016. – letzter Zugriff am 20. April 2016

[Clo16b] CLOUDERA INC.: *Hive Command line Options*. https://archive.cloudera.com/cdh4/cdh/4/hive/language_manual/cli.html, 2016. – letzter Zugriff am 15. März 2016

[Dat15] DATABRICKS INC.: *Spark Survey Results 2015*. https://databricks.com/blog/2015/09/24/spark-survey-results-2015-are-now-available.html, 2015. – letzter Zugriff am 15. März 2016

[Ela16a] ELASTIC: *Deploying and Scaling Logstash*. https://www.elastic.co/guide/en/logstash/current/deploying-and-scaling.html#deploying-and-scaling, 2016. – letzter Zugriff am 27. Februar 2016

[Ela16b] ELASTIC: *Logstash: Collect, Enrich & Transport Data*. https://www.elastic.co/products/logstash, 2016. – letzter Zugriff am 27. Februar 2016

[Fou16a] FOUNDATION, The Apache S.: *Apache Zeppelin (incubating)*. https://zeppelin.incubator.apache.org, 2016. – letzter Zugriff am 28. April 2016

[Fou16b] FOUNDATION, The Apache S.: *Contributing to Apache Zeppelin (Website)*. https://zeppelin.incubator.apache.org/docs/0.6.0-incubating-SNAPSHOT/development/howtocontributewebsite.html, 2016. – letzter Zugriff am 28. April 2016

[Fou16c] FOUNDATION, The Apache S.: *Zeppelin (0.5.5-incubating*. http://zeppelin.incubator.apache.org/docs/latest/development/writingzeppelininterpreter.html, 2016. – letzter Zugriff am 08. Mai 2016

[Hor13] HORTONWORKS: *Storm Concept*. http://hortonworks.com/wp-content/uploads/2013/11/Storm_concepts-300x170.png, 2013. – letzter Zugriff am 07. März 2016

[Hum14] HUMMELTENBERG, Prof. Dr. W.: *Business Intelligence*. http://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/daten-wissen/Business-Intelligence. Version: 2014. – letzter Zugriff am 31. Oktober 2015

[Ico15] ICONIQ INC.: *Sqoop vs. Flume - Battle of the Hadoop ETL tools*. https://www.dezyre.com/article/sqoop-vs--battle-of-the-hadoop-etl-tools-/176, 2015. – letzter Zugriff am 27. April 2016

[Kee15] KEEDIO COMMUNITY: *Keedio FTP Flume Source*. https://www.keedio.org/keedio-ftp-flume-source/, 2015. – letzter Zugriff am 16. September 2016

[KKWZ15] KARAU, Holden ; KONWINSKI, Andy ; WENDELL, Patrick ; ZAHARIA, Matai: *Learning Spark*. First Edition. USA : O'REILLY, 2015

[Lea11] LEARNFLUME: *Apache Flume - Quick Guide*. http://www.tutorialspoint.com/apache_flume/apache_flume_quick_guide.htm, 2011. – letzter Zugriff am 01. April 2016

[Lin15a] LIN QIAO: *Gobblin: A Framework for Solving Big Data Ingestion Problem*. http://www.infoq.com/presentations/gobblin-linkedin?utm_source=infoq&utm_medium=slideshare&utm_campaign=slidesharesf, 2015. – letzter Zugriff am 25. März 2014

[Lin15b] LINKEDIN: *Gobblin*. http://gobblin.readthedocs.org/en/latest/?badge=latest, 2015. – letzter Zugriff am 25. März 2014

[Lin15c] LINKEDIN: *Gobblin*. http://gobblin.readthedocs.org/en/latest/Gobblin-Architecture, 2015. – letzter Zugriff am 25. März 2014

[Lin15d] LINKEDIN: *Gobblin*. http://gobblin.readthedocs.org/en/latest/user-guide/Gobblin-Deployment, 2015. – letzter Zugriff am 25. März 2014

[Lin15e] LINKEDIN: *Gobblin Constructs*. http://gobblin.readthedocs.org/en/readthedocs/Gobblin-Architecture/, 2015. – letzter Zugriff am 07. März 2016

[Mar14] MARZ, Nathan: *History of Apache Storm and lessons learned*. http://nathanmarz.com/blog/history-of-apache-storm-and-lessons-learned.html, 2014. – letzter Zugriff am 07. März 2016

[Maz15] MAZIAR, Kaveh: *ETL and Analysis of IoT data using OpenTSDB, Kafka, and Spark*. http://brage.bibsys.no/xmlui/bitstream/handle/11250/299607/Kaveh_Maziar.pdf?sequence=1&isAllowed=y, Juni 2015. – letzter Zugriff am 10. Februar 2016

[MSH14] MOCK, Michael ; SYLLA, Karl-Heinz ; HECKER, Dirk: Skalierbarkeit und Architektur von Big-Data-Anwendungen. In: *OBJEKTspektrum IT-Trends* 2014 (2014), Nr. 2014, S. 1–7

[OB15] OLIVER BEHREND, Fatih-Mehmet I. Eugen Neigel N. Eugen Neigel: *Analytik am Beispiel „Optimierung der CEWE FOTOSTATIONEN"*. Präsentation CEWE Vorstandssitzung, 14.09.2015, 2015. – letzter Zugriff am 01. September 2016

[Obj14] OBJECT REFINERY LIMITED: *Welcome To JFreeChart!* http://www.jfree.org/jfreechart/, 2014. – letzter Zugriff am 26. Juli 2016

[Pen15] PENTREATH, Nick: *Machine Learning with Spark*. First Edition. UK : Packt Publishing Ltd., 2015

[Pla15] PLATFORM, Confluent: *HDFS Connector*. http://docs.confluent.io/2.0.0/connect/connect-hdfs/docs/index.html, 2015. – letzter Zugriff am 18. September 2016

[Rao13] RAO, Jun: *Intra-cluster Replication for Apache Kafka*. http://de.slideshare.net/junrao/kafka-replication-apachecon2013, 2013. – letzter Zugriff am 02. März 2016

[RK15] RAJASETHUPATHY, Karthik ; KUKA, Christian: *The Hadoop Eco System*. https://www.kuka.cc/assets/shds/ShanghaiDataScience_20151103.pdf, November 2015. – letzter Zugriff am 11. Februar 2016

[RLOW15] RYZA, Sandy ; LASERSON, Uri ; OWEN, Sean ; WILLS, Josh: *Advanced Analytics with Spark*. First Edition. USA : O'REILLY, 2015

[Rom15] ROMANSHAPOSHNIK: *Incubator PMC report for January 2015*. https://wiki.apache.org/incubator/January2015, Januar 2015. – letzter Zugriff am 12. Mai 2016

[Sev14] SEVERALNINES.COM: *Archival and Analytics - Importing MySQL data into Hadoop Cluster using Sqoop*. http://severalnines.com/blog/archival-and-analytics-importing-mysql-data-hadoop-cluster-using-sqoop, 2014. – letzter Zugriff am 27. April 2016

[Sha14] SHARMA, Abhishek: *Apache Kafka: Next Generation Distributed Messaging System*. http://www.infoq.com/articles/apache-kafka, Juni 2014. – letzter Zugriff am 11. Februar 2016

[SK15] SANKAR, Krishna ; KARAU, Holden: *Fast Data Processing with Spark*. Second Edition. UK : Packt Publishing Ltd., 2015

[Sri13] SRIPATI, Praveen: *Sqoop2 vs Sqoop*. http://www.thecloudavenue.com/2013/10/sqoop2-vs-sqoop.html, 2013. – letzter Zugriff am 29. Juni 2016

[Sur13] SURAJIT, Paul: *Sqoop: Big data conduit between NoSQL and RDBMS*. http://www.ibm.com/developerworks/library/bd-sqoopconduit/, 2013. – letzter Zugriff am 20. April 2016

[Tre16a] TREASURE DATA INC.: *Buffer Plugin Overview*. http://docs.fluentd.org/articles/buffer-plugin-overview, 2016. – letzter Zugriff am 15. März 2016

[Tre16b] TREASURE DATA INC.: *Fluentd Documentation*. http://docs.fluentd.org/articles/#, 2016. – letzter Zugriff am 15. März 2016

[Tre16c] TREASURE DATA INC.: *Frequently Asked Questions*. http://www.fluentd.org/faqs, 2016. – letzter Zugriff am 15. März 2016

[Tre16d] TREASURE DATA INC.: *Installation*. http://docs.fluentd.org/v0.12/categories/installation, 2016. – letzter Zugriff am 15. März 2016

[Tre16e] TREASURE DATA INC.: *Multiprocess Input Plugin*. http://docs.fluentd.org/articles/in_multiprocess, 2016. – letzter Zugriff am 15. März 2016

[Tre16f] TREASURE DATA INC.: *Plugins*. http://www.fluentd.org/plugins, 2016. – letzter Zugriff am 15. März 2016

[Tre16g] TREASURE DATA INC.: *What is Fluentd?* http://www.fluentd.org/architecture, 2016. – letzter Zugriff am 15. März 2016

A. Anhang

A.1. Dateninhalt

Bei der XML-Dokumentation handelt es sich um eine Excel-Datei, in der die Struktur der XML-Dateien abgebildet ist. Aus Gründen der Lesbarkeit werden in Tabelle A.2 nur die drei Spalten Tag, Attribute des Tags und Beschreibung dargestellt. Zudem werden nicht alle Inhalte angezeigt. Die vollständige XML-Dokumentation befindet sich auf der Daten-CD.

Die Tabelle A.1 erläutert die Bedeutung der einzelnen Spalten der Excel-Datei.

Spalte	**Beschreibung**
Tag	Hier steht der Name des jeweiligen Tags.
Attribute des Tags	Hier sind die Attribute des Tags aufgelistet. Hat ein Tag mehrere Attribute, so werden mehrere Zeilen mit dem selben Tag und den jeweils verschiedenen Attributen erstellt.
Beschreibung	An dieser Stelle ist die Bedeutung der einzelnen Tags und Attribute beschrieben.
Pfad	Hier wird der hierarchische Pfad vom Wurzelelement bis zum Tag abgebildet.
Kategorie	Anhand der Kategorie (bspw. DFM Stammdaten, Clickstreamdaten, Produktdaten) kann später eine leichtere Einschätzung vorgenommen werden, welche Daten für das Data Mining herangezogen werden können.
Wertausprägungen	Hier stehen Beispielwerte, um einen Überblick über die Inhalte zu erhalten.
Abhängigkeit	Hier ist dokumentiert, ob ein Tag nur in Abhängigkeit zu anderen Tags oder Attributausprägungen auftreten kann.
Bemerkungen	Hier werden Besonderheiten und/oder Anmerkungen erfasst .

Tabelle A.1.: Spaltenbeschreibung der XML-Dokumentation

Tag	Attribute des Tags	Beschreibung
	Grundstruktur	
<digifoto>		Root Element
<tracker>		Zweites „Root“ Element mit ID
<terminal>		DFM Beschreibung
<ticket>		Beschreibung des Software Tickets; enthält die Spezifikation der Version
<path>		Beschreibung der Kundeninteraktion
<summary>		Zusammenfassung der Session, Status der Session, Produkte, Informationen über Bilder
	Tracker	
<tracker>	uuid	Eindeutige Session ID
	Terminal	
<terminal>	location	Kundennummer(6-stellig) + Suffix Gerätenummer (2-stellig)
<terminal>	membership	Hier dient ein willkürlich gewählter String zur Gruppierung von XML-Dateien.
<terminal>	id	ID des DFM (Numerischer Key)
<terminal>	keyacc	Kundengruppe numerisch 1-5 stellig

Tag	Attribute des Tags	Beschreibung
	Ticket	
<ticket>	version	Kurzfassung der Firmware Version
<ticket>	value	Entscheidendes Attribut sogenannte df-Nummer beschreibt genau welche Software im Einsatz ist; Version + DF-Nummer entscheidend
	Path	
<enter>	class	C++ Klasse, die einen Programmablauf beinhaltet. Zusammen mit dem Attribut step und label kann der Punkt im Programm beschrieben werden.
<enter>	fullscreen	Ansicht im Vollbildmodus
<enter>	nest	Nest gibt an, ob eine Verschachtelung von mehreren aufeinanderfolgenden enter-Tags vorliegt.
<enter>	when	Entstehungszeit des Tags
<enter>	step	Schritt im Programmablauf; hierfür existiert eine Ablaufdatei, die noch zur Verfügung gestellt wird.
<enter>	script	Dreiteiliges Programm: C++ Klasse, UI-File und im Attribut script ein benutztes javascript File
<enter>	label	aktuelles UserInterface
<enter>	header	header sichtbar oder nicht (pendant zu fullscreen)

Tag	Attribute des Tags	Beschreibung
<enter>	msec	Erweiterung des Tags when um die Millisekunden
<videoList>	videos	Gibt es am Start und während des Druckens, Komma separierte Liste der zu spielenden Videos in der aufgeführten Reihenfolge
<videoList>	when	Erstellung des Tags
<printjob>	sched	Die Tags kommen vom Drucksystem
<printjob>	sfc	Papierformat auf das gedruckt wird, Services
<printjob>	type	Produkttyp der gedruckt wird
<printjob>	ts	Wann der Druckauftrag angelegt wurde
<printerstatus>		CDATA Ausprägungen; asynchron; kann immer mal wieder kommen
<printerstatus>	status	In welchem Zustand befindet sich der Drucker (interessant sind die Cooling-Zeiten)
<printerstatus>	class	class und type ergibt den Drucker
<printerstatus>	type	class und type ergibt den Drucker
<printerstatus>	ts	Erstellung des Tags
	focusedProduct	Zeigt auf das Produkt, was gerade in Erstellung ist, aber noch nicht gekauft wird/wurde; neues Konzept
<focusedProduct>	class	Produktgruppe

Tag	Attribute des Tags	Beschreibung
<focusedProduct>	price	Preis des Produktes
<focusedProduct>	quantity	Anzahl der Produkte, abhängig vom Produkt (nur über class und type Summierung)
<focusedProduct>	type	Produkttyp
<focusedProduct>	cartQuantity	Stückzahl
<focusedProduct>	idx	Durchnummerierte Instanz der Produkte (wird nur hochgezählt); wie viele Produkte wurden während der Session ausgewählt und am Ende tatsächlich gedruckt/gekauft
<leave>	rc	Return Code, was passiert ist (ebenfalls zu verstehen mit der Ablaufdatei, die noch geliefert wird)
<leave>	class	C++ Klasse, die einen Programmablauf beinhaltet. Zusammen mit dem Attribut step und label kann der Punkt im Programm beschrieben werden
<leave>	nest	Nest gibt an, ob eine Verschachtelung von mehreren aufeinanderfolgenden leave-Tags vorliegt.
<leave>	when	Erstellung des Tags
<leave>	label	aktuelles UserInterface
<medium>	name	Herstellername des Mediums
<medium>	size	Speicherbelegung des Mediums in Byte

Tag	Attribute des Tags	Beschreibung
<medium>	when	Erstellung des Tags
<medium>	slot	Über welchen Anschluss wurde das Medium angeschlossen
<medium>	type	Typ des Mediums
<medium>	hash	Hashwert auf das Medium
<device>	sn	Eindeutige Seriennummer für ein Gerät; iPhone ist sauber, Samsung und HTC auch, einige nicht;
<device>	pid	Product ID
<device>	vid	Vendor ID
<dir>	files	Verschachtelung der <dir> Tags möglich, je nach Verzeichnisstruktur; Komplette Verzeichnisstruktur wird gescannt, wo ein Bild drin ist
<dir>	total_bytes	Größe des Verzeichnisses in Byte
<dir>	other_bytes	Größe des Verzeichnisses ohne Bilder in Byte
<dir>	image_bytes	Größe der Bilder im Verzeichnis in Byte
<dir>	dirs	Anzahl der weiteren Verzeichnisse im aktuellen Verzeichnis
<dir>	path	Name des Verzeichnisses
<dir>	hash	Hashwert über das Verzeichnis
<activate>	when	Erstellung des Tags

Tag	Attribute des Tags	Beschreibung
<activate>	msec	Erweiterung des Tags when um die Millisekunden
<activate>	label	Bezeichnung des activate Schrittes
<activate>	x	Koordinate
<activate>	y	Koordinate
<mifare>	seed	Prepaid Zahlungsverfahren
<mifare>	action	Die Aktion, die passiert. Ebenfalls in der Ablaufdatei beschrieben.
<mifare>	when	Erstellung des Tags
<mifare>	type	Wertausprägung CustomerCard ist für den Kunden. Es gibt auch Spezialkarten des Partners sogenannte DealerCards. Kunde kauft Guthaben beim Partner und bezahlt dann über die Guthabenkarte des Partners.
<mifare>	value	Wert der Guthabenkarte
<mifare>	blacklisted	Hier hat Betrug stattgefunden
<mifare>	id	Unique mifare-Karten ID.
<event>	id	HotPlugg Events; sehr technisch für USB; es wird etwas angeschlossen, aber nicht erkannt;
<event>	when	Erstellung des Tags

Tag	Attribute des Tags	Beschreibung
<event>	msec	Ergänzung des when Attributes um die Millisekunden
<data id='subSystem'>		Typ des Mediums, dass das Event ausgelöst hat. Hiernach sollte differenziert werden.
	Connectivitytests	
<connectivitytest>	timestamp	Zeitpunkt des ConnectivityTests
<xtcitest>	connectivity	Ergibt den Erfolg des Tests
<xtcitest>	durationInMSec	Dauer der Rückmeldung in Millisekunden
<xtcitest>	url	Zu testende URL
<httptest>	connectivity	Ergibt den Erfolg des Tests
<httptest>	durationInMSec	Dauer der Rückmeldung in Millisekunden
<httptest>	url	Zu testende URL
	Summary	
<order>	state	Endzustand der Session (success, aborted, printed etc.)
<orderlist>	orderid	Auftragsnummer
<imagesources>		Auflistung der Bilder, die im Warenkorb sind
<imagesource>	class	Bild oder Berechnet (Hintergründe, Layout etc.)
<imagesource>	id	Eindeutige ID für die Class

Tag	Attribute des Tags	Beschreibung
<properties>		enthält diverse Exif Daten des Bildes
<width>		Breite des Bildes in Pixel
<height>		Höhe des Bildes in Pixel
<filename>		Dateiname des Bildes
<originalfilename>		Original Dateiname des Bildes
<filesize>		Dateigröße des Bildes
<timestamp>		Erstellungszeit der Property
<imageTimeFromExifData>		Timestamp vom Files oder aus dem ExifHeader
<products>		Übersicht der Produkte im Warenkorb
<product>	class	Gruppierung von Produkten
<product>	subtype	Bspw. Bei einer Tasse die Innenfarbe
<product>	type	Produkttyp
<property id='printObjectsNeeded'>		Die Anzahl Bilder, die gedruckt werden sollen
<property id='printObjectsComplete'>		Die Anzahl, die gedruckt wurden
<property id='price'>	value	Gesamtpreis für das Produkt
<property id='currency'>	value	Währung für das Produkt
<property id='artnr'>	value	Artikelnummer des Produktes

Tag	Attribute des Tags	Beschreibung
<images>	count	Summe der Bilder
<property id =’currency’>		Währung der Session

A.2. Datenbankschema in Hive

Abbildung A.1.: Datenbankschema in Hive

A.3. Benchmark der XML-Bibliotheken in Python

lxml.etree

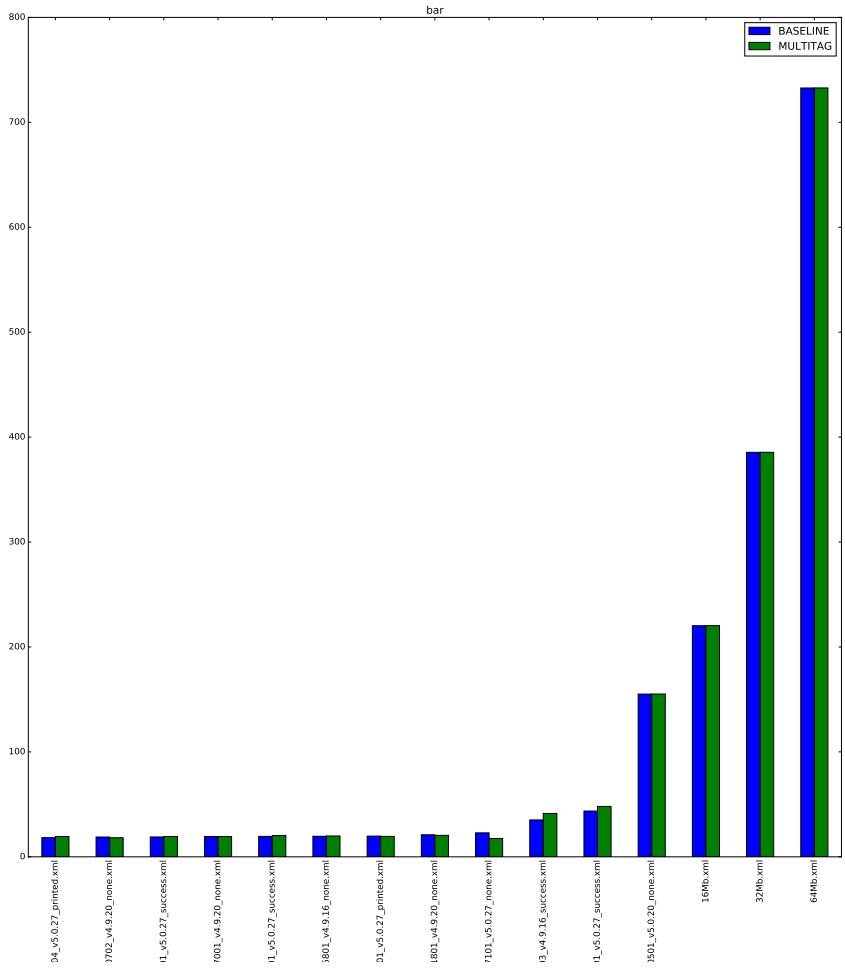


Abbildung A.2.: Benchmark des Frameworks „lxml.etree“

RAM-Benchmark des Frameworks lxml.objectify

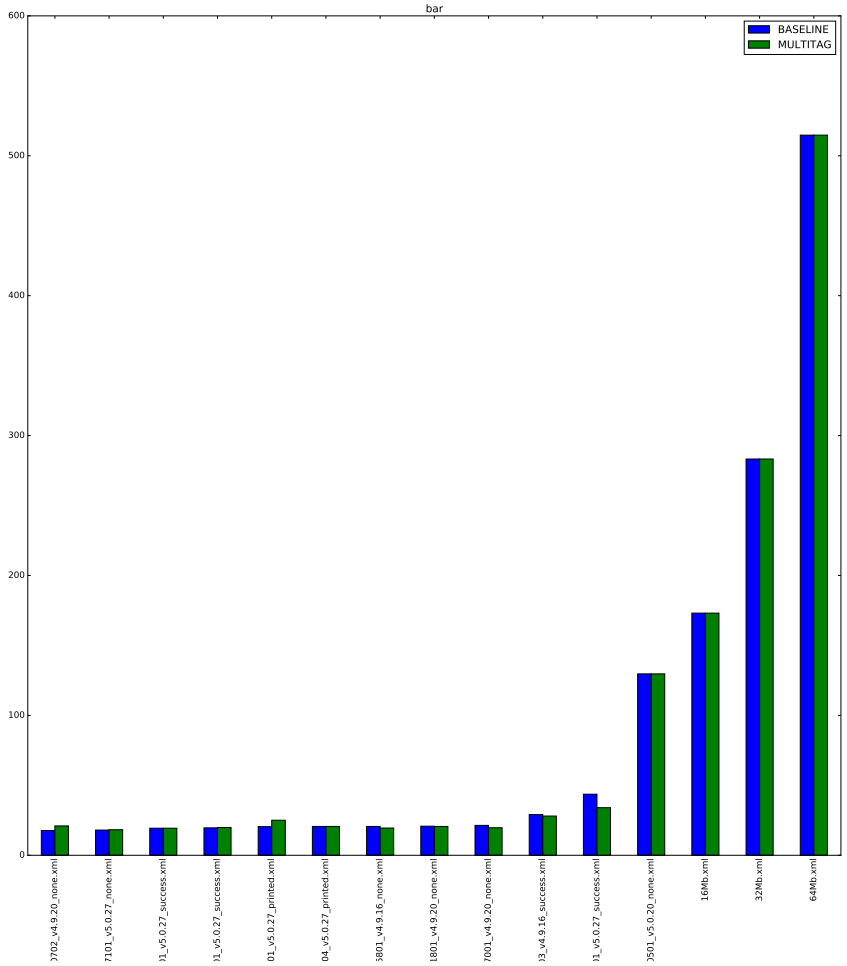


Abbildung A.3.: Benchmark des Frameworks „lxml.objectify“

RAM-Benchmark des Frameworks xml.dom.minidom

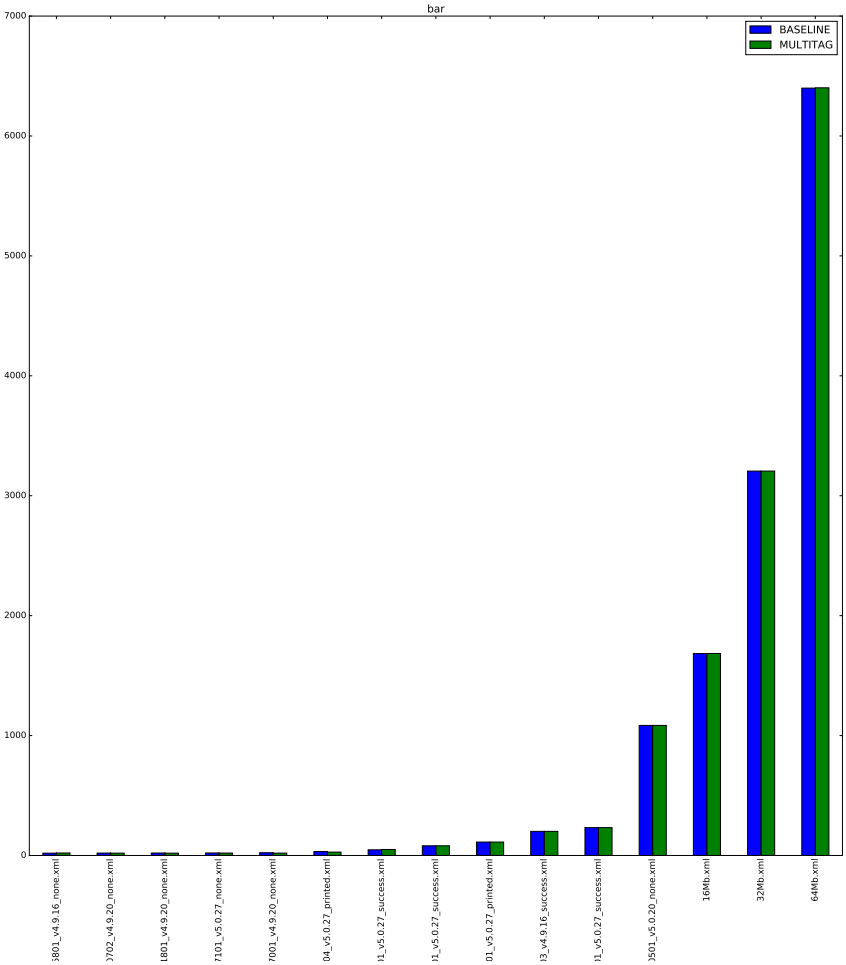


Abbildung A.4.: Benchmark des Frameworks „xml.dom.minidom“

RAM-Benchmark des Frameworks xml.dom.pulldom

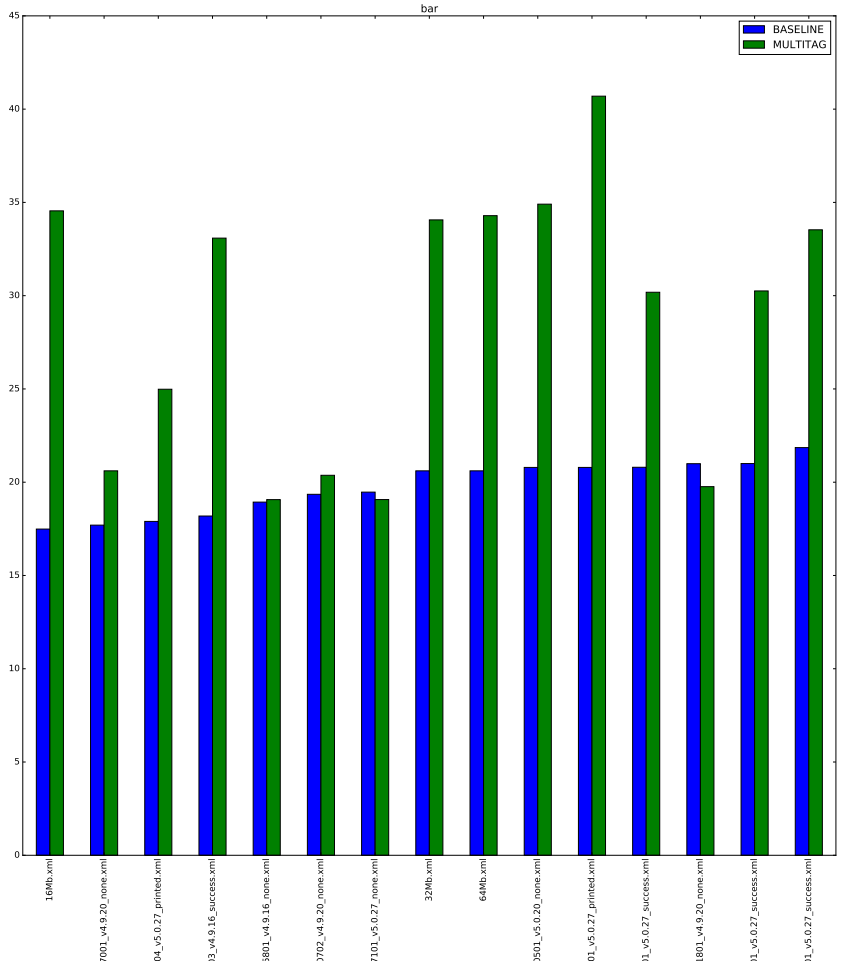


Abbildung A.5.: Benchmark des Frameworks „xml.dom.pulldom“

RAM-Benchmark des Frameworks xml.etree.ElementTree

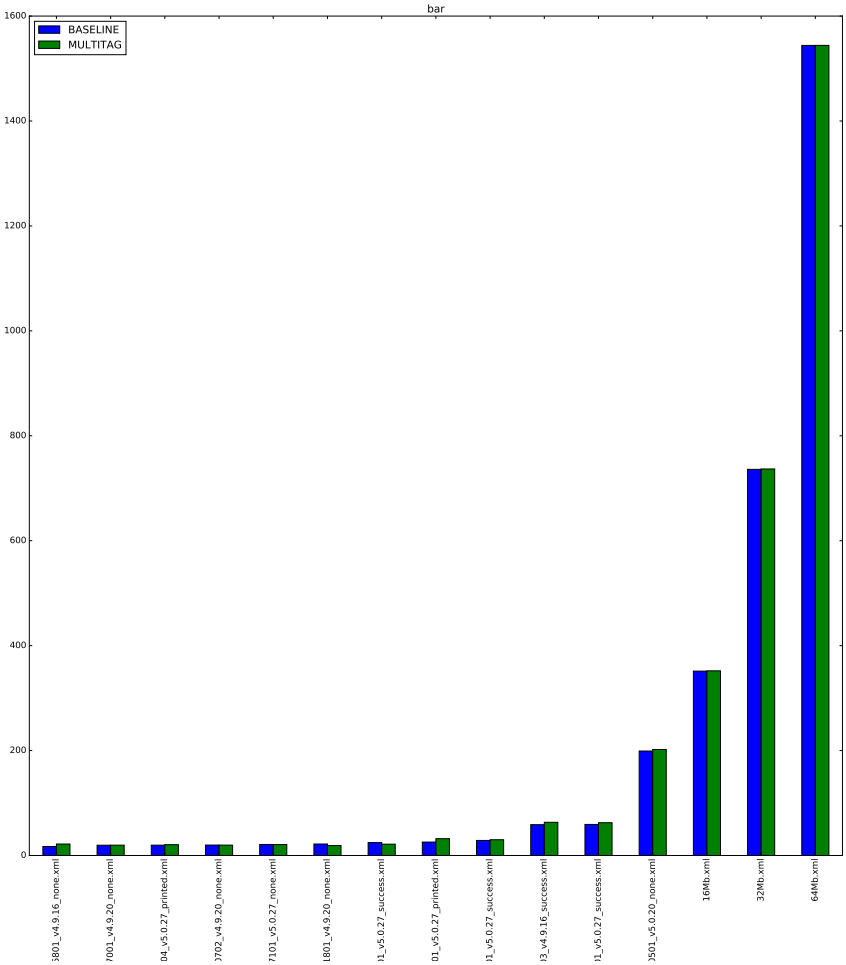


Abbildung A.6.: Benchmark des Frameworks „xml.etree.ElementTree“

RAM-Benchmark des Frameworks xml.etree.cElementTree

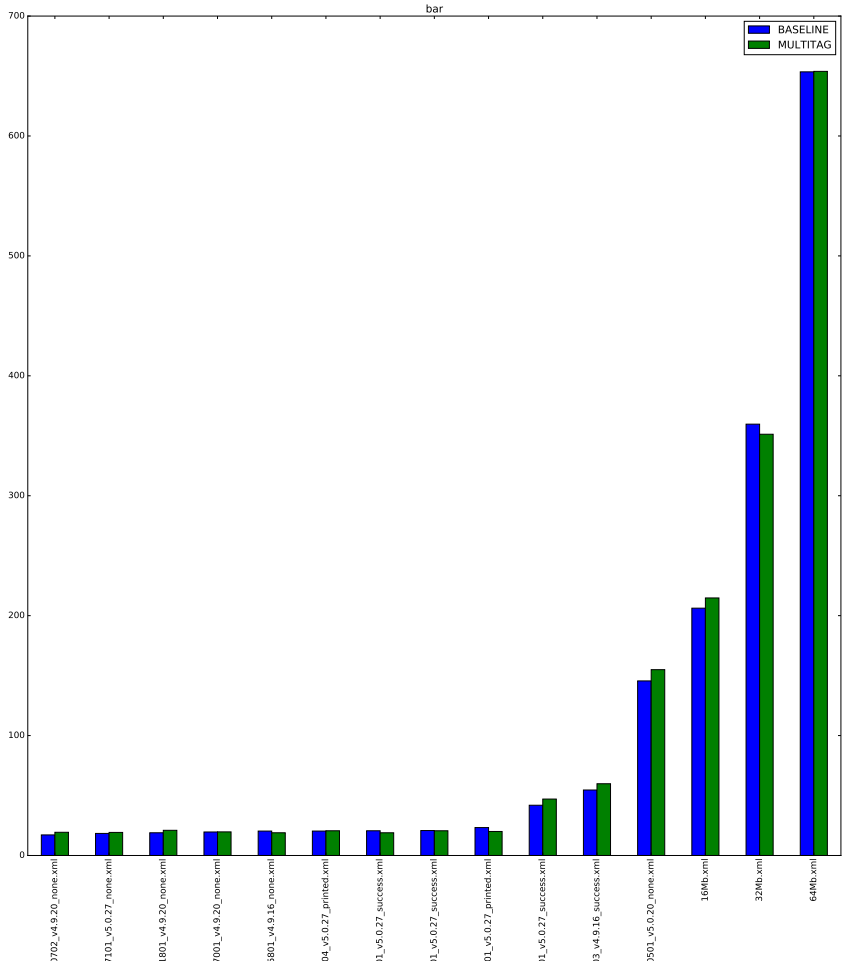


Abbildung A.7.: Benchmark des Frameworks „xml.etree.cElementTree“

RAM-Benchmark des Frameworks xml.sax

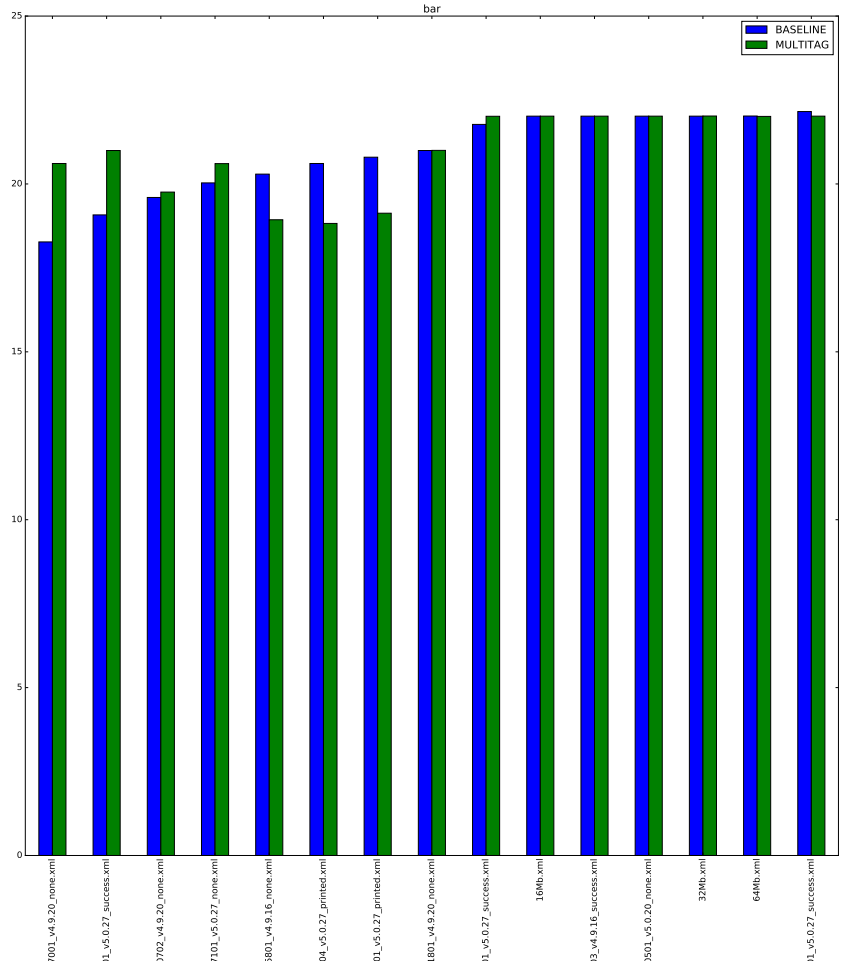


Abbildung A.8.: Benchmark des Frameworks „xml.sax“

RAM-Benchmark: Alle Frameworks mit 32MB

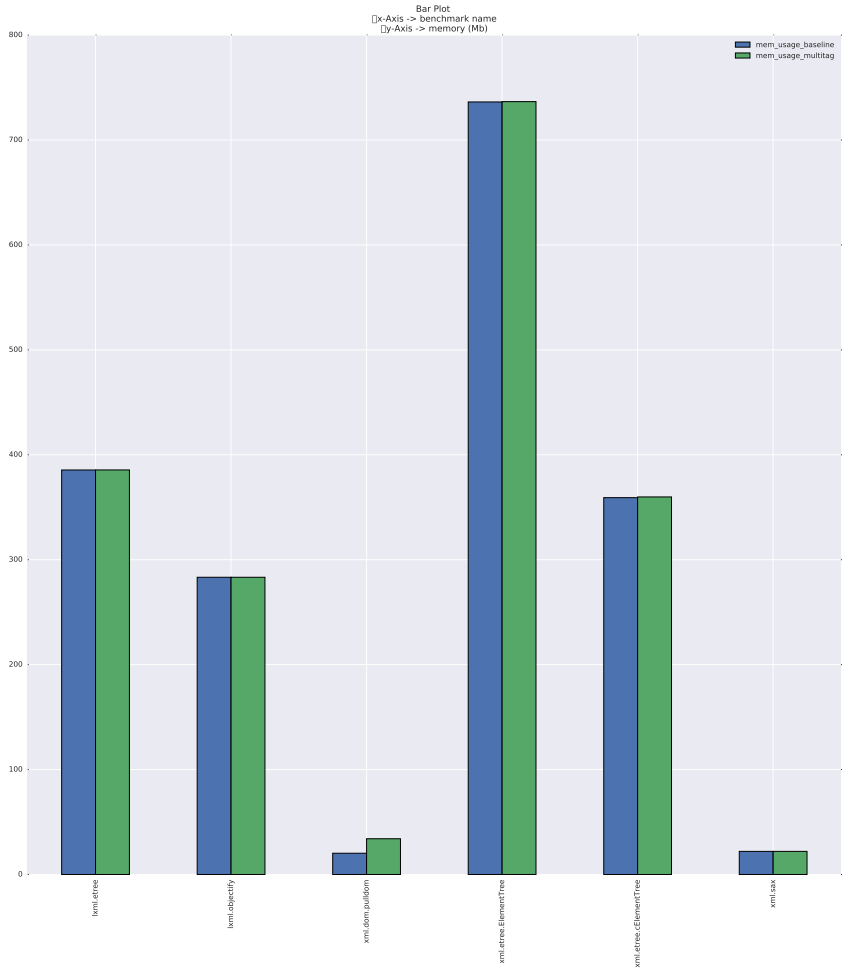


Abbildung A.9.: RAM-Benchmark: Alle Frameworks mit 32MB

Zeit/RAM-Benchmark: Alle Frameworks mit 32MB

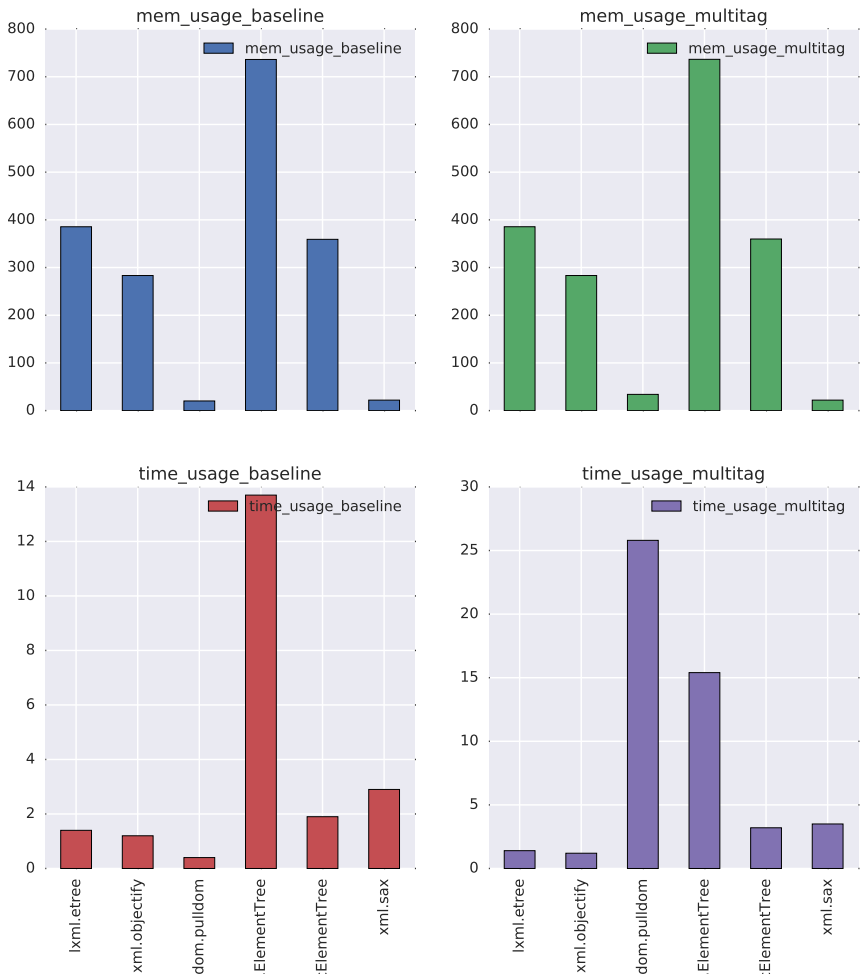


Abbildung A.10.: Zeit/RAM-Benchmark: Alle Frameworks mit 32MB

Zeit-Benchmark: Alle Frameworks mit 32MB

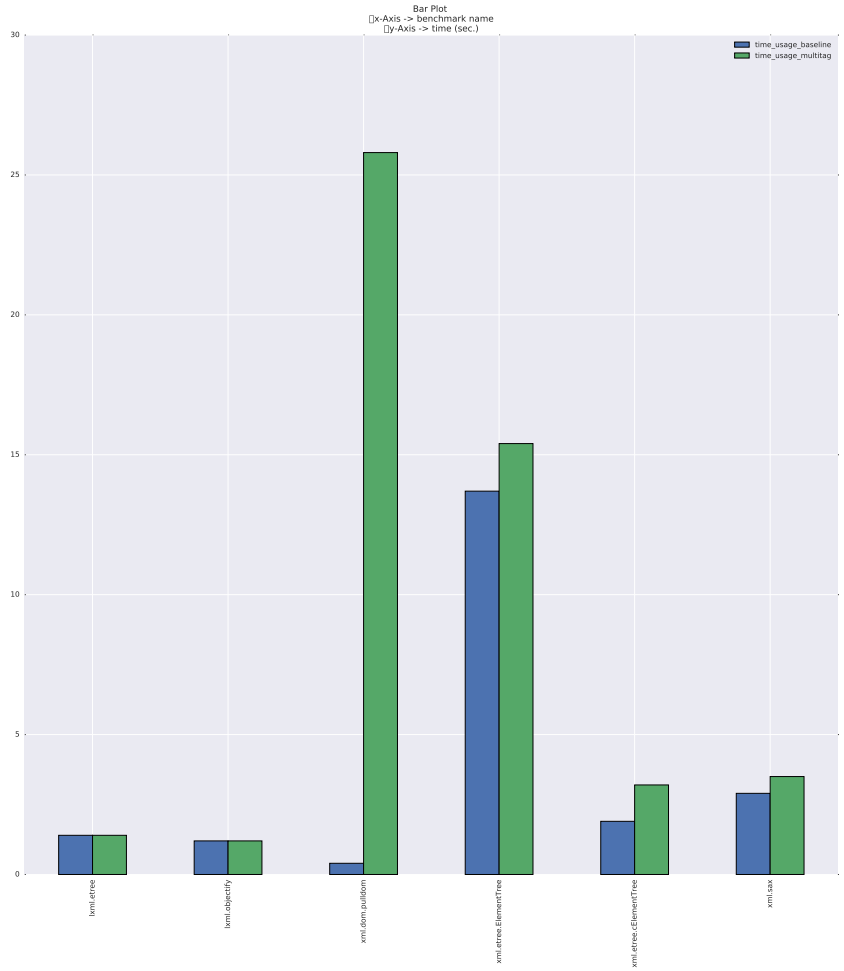


Abbildung A.11.: Zeit-Benchmark: Alle Frameworks mit 32MB

RAM-Benchmark: Alle Frameworks mit 64MB

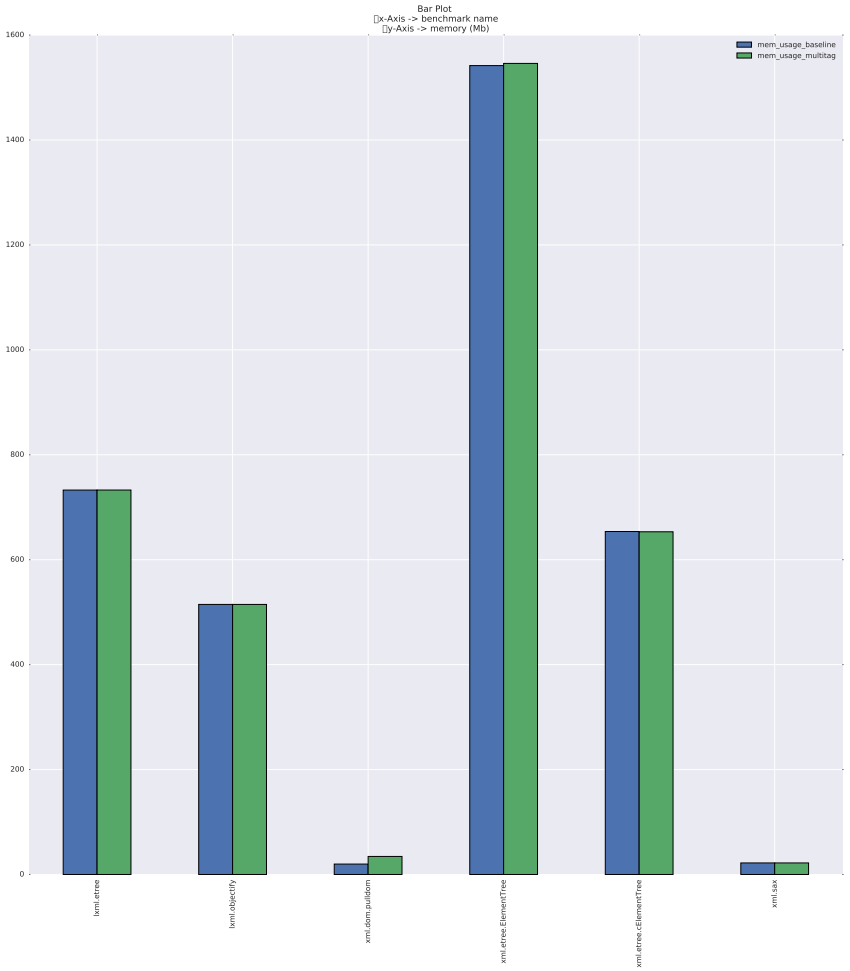


Abbildung A.12.: RAM-Benchmark: Alle Frameworks mit 64MB

Zeit/RAM-Benchmark: Alle Frameworks mit 64MB

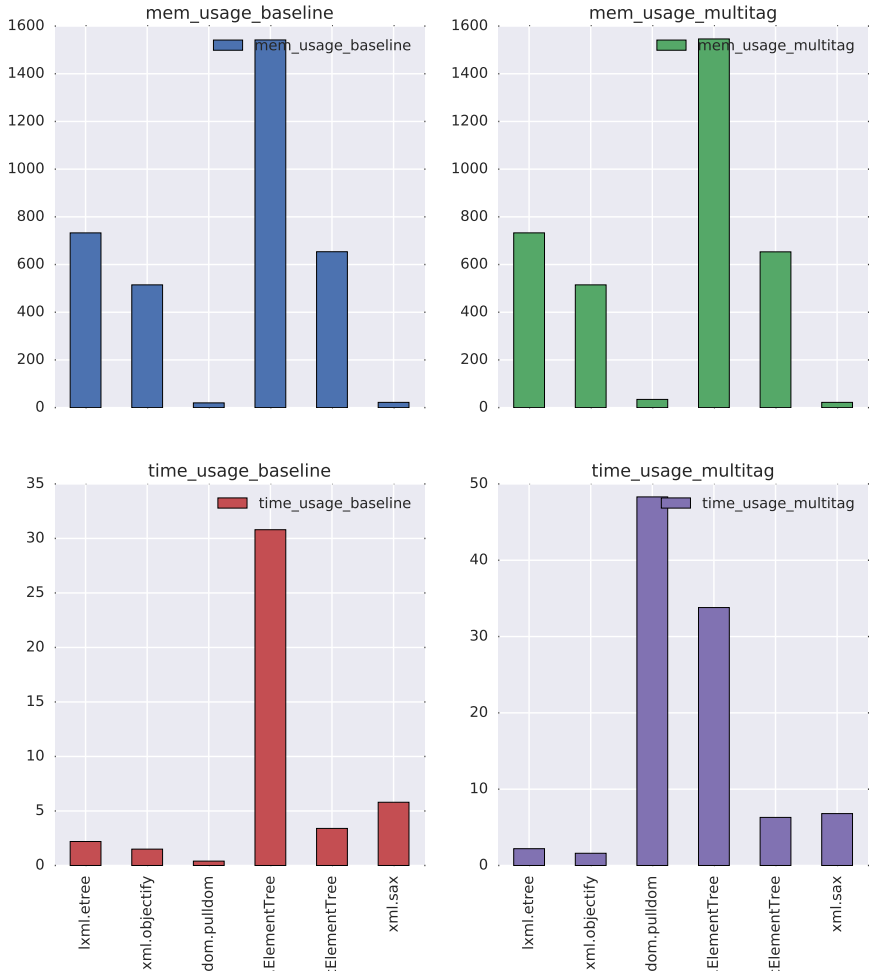


Abbildung A.13.: Zeit/RAM-Benchmark: Alle Frameworks mit 64MB

Zeit-Benchmark: Alle Frameworks mit 64MB

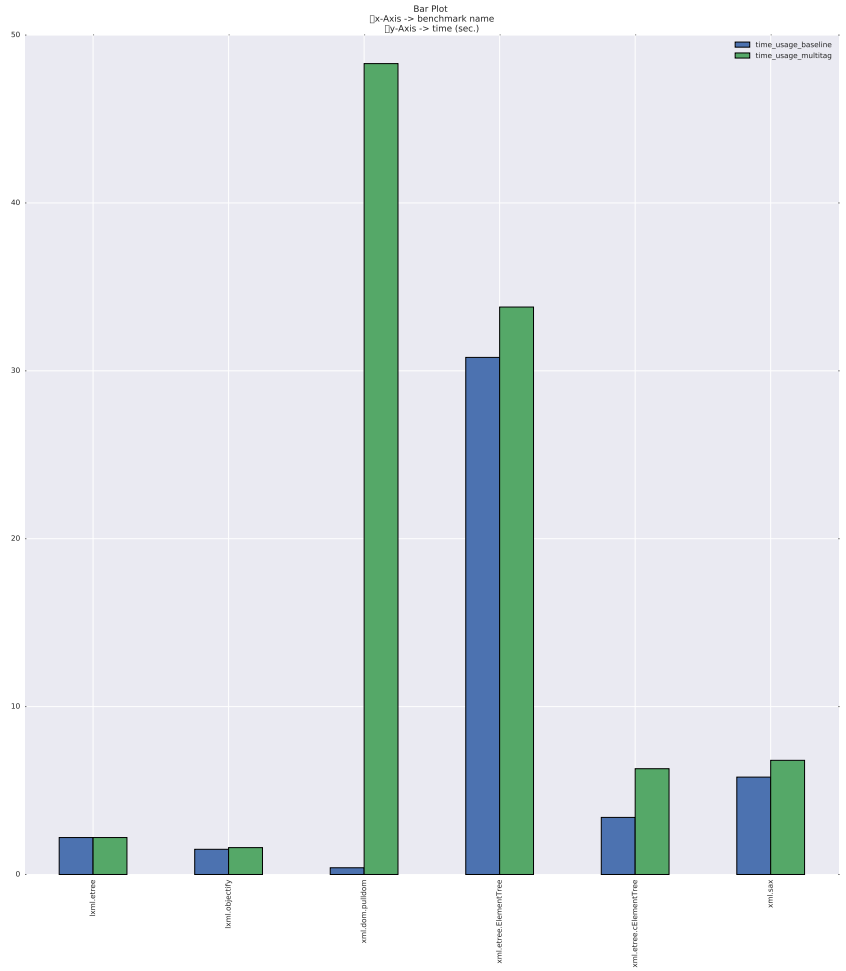


Abbildung A.14.: Zeit-Benchmark: Alle Frameworks mit 64MB

RAM-Baseline-Benchmark: Alle Frameworks mit verschiedenen Dateigrößen

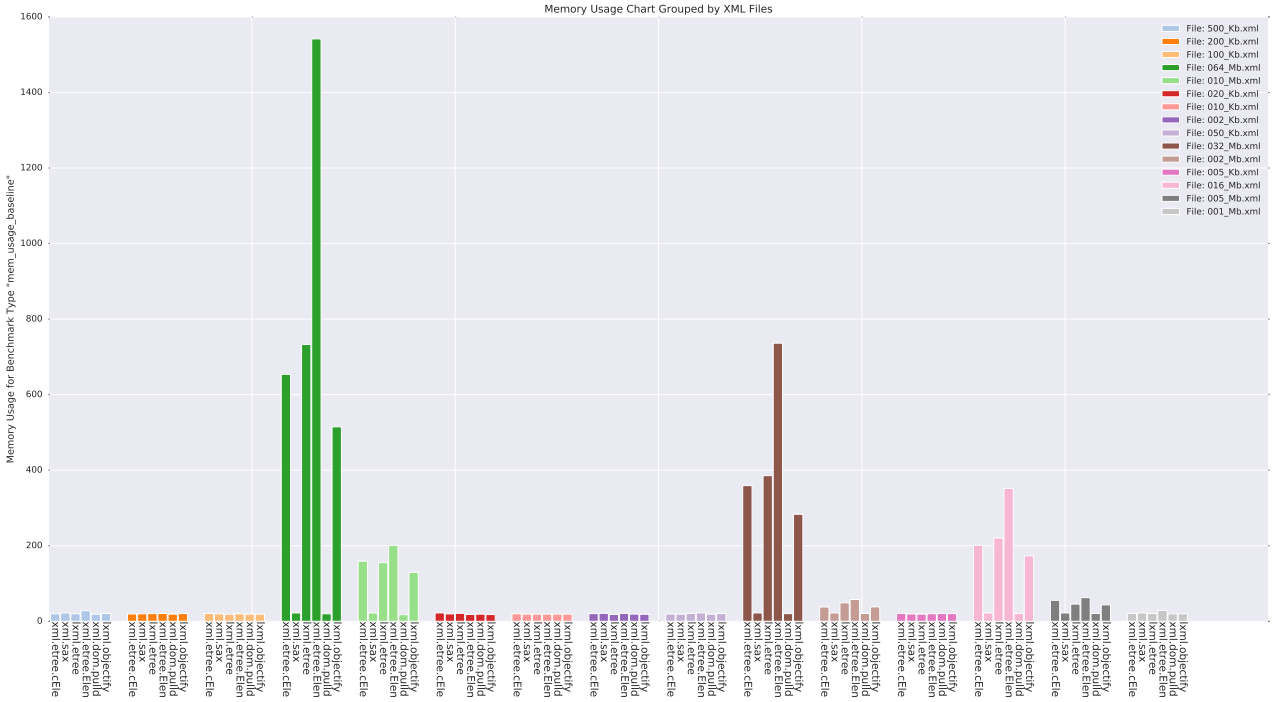


Abbildung A.15.: RAM-Baseline-Benchmark: Alle Frameworks mit verschiedenen Dateigrößen

RAM-Multitag-Benchmark: Alle Frameworks mit verschiedenen Dateigrößen

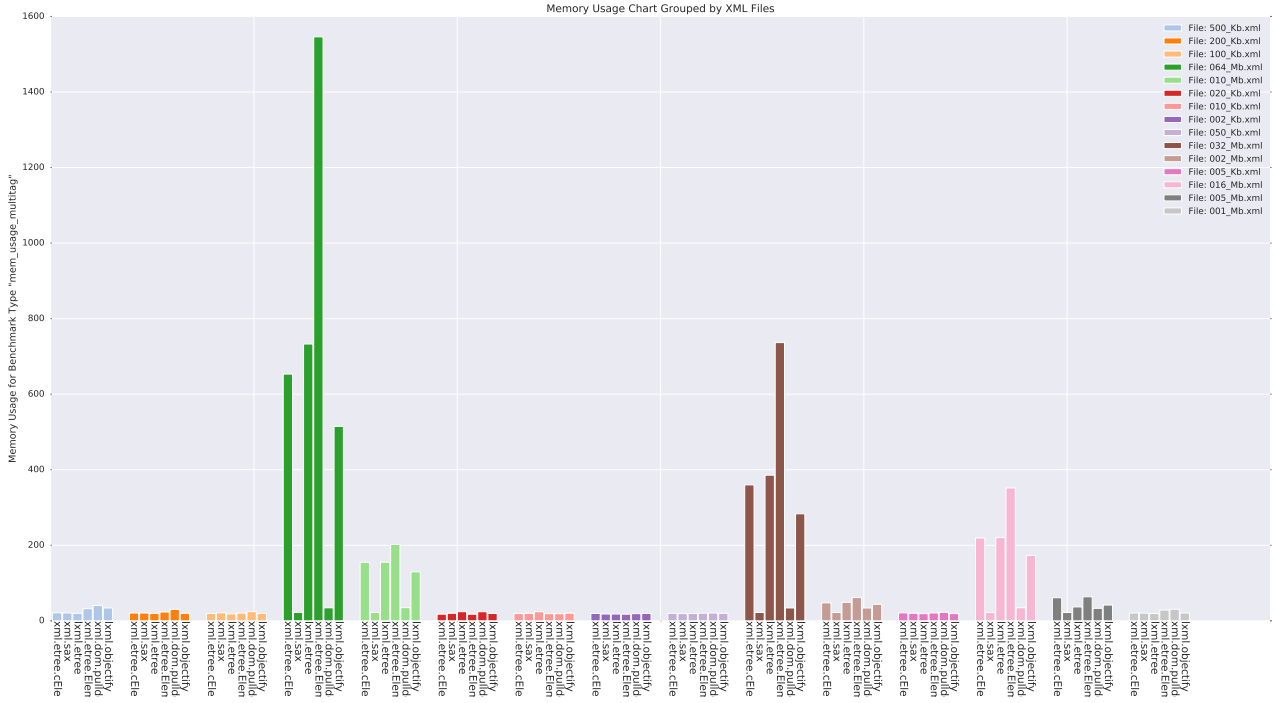


Abbildung A.16.: RAM-Multitag-Benchmark: Alle Frameworks mit verschiedenen Dateigrößen

Zeit-Baseline-Benchmark: Alle Frameworks mit verschiedenen Dateigrößen

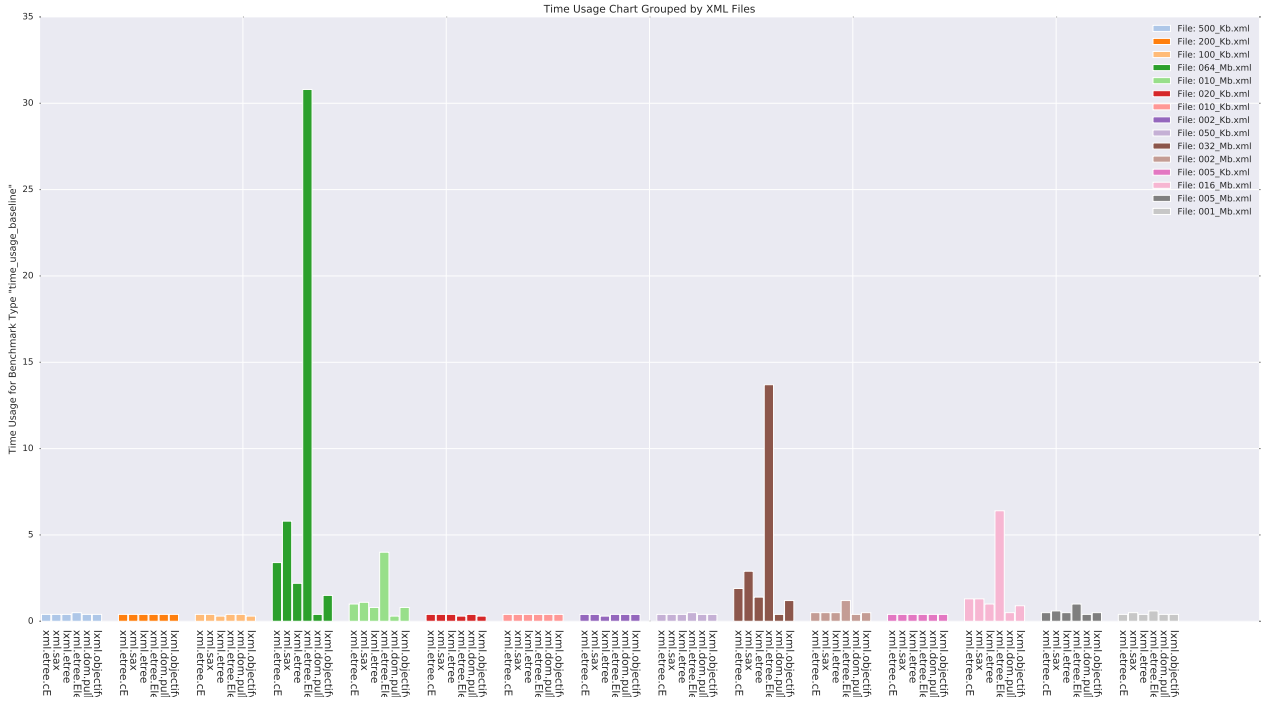


Abbildung A.17.: Zeit-Baseline-Benchmark: Alle Frameworks mit verschiedenen Dateigrößen

Zeit-Multitag-Benchmark: Alle Frameworks mit verschiedenen Dateigrößen

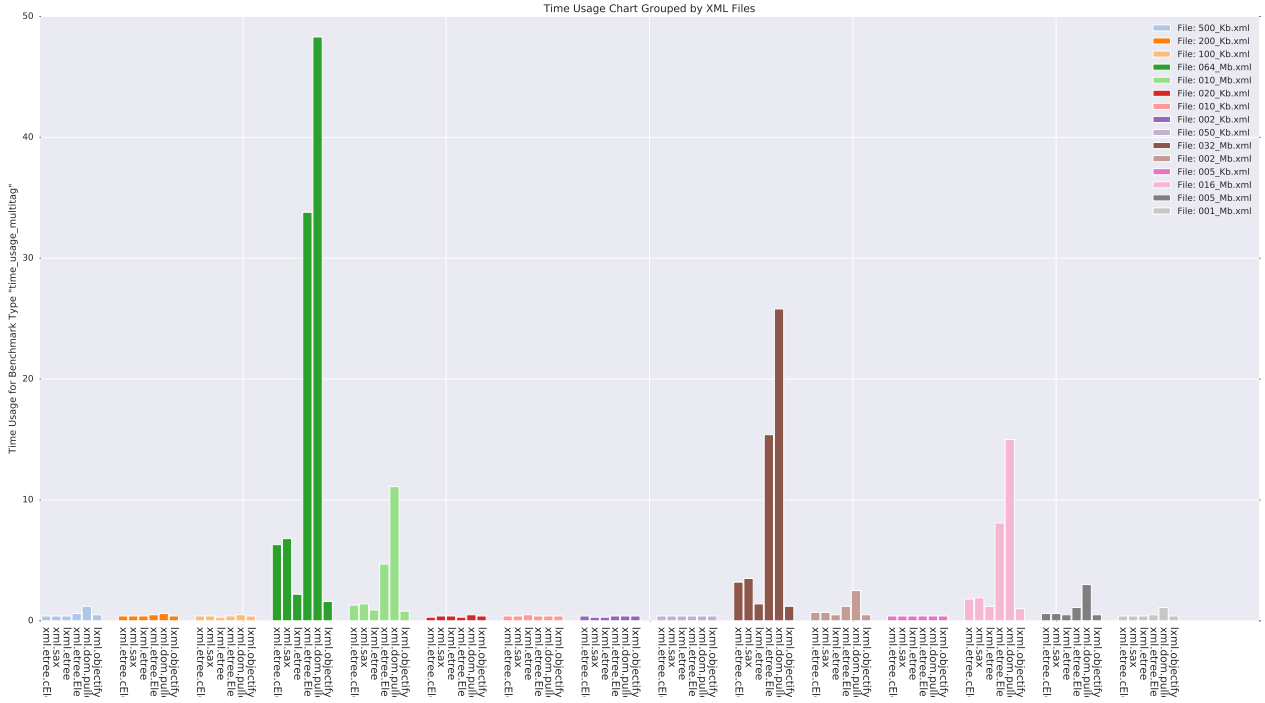


Abbildung A.18.: Zeit-Multitag-Benchmark: Alle Frameworks mit verschiedenen Dateigrößen

A.4. Blog

Beitrag vom 01.10.2015

Wir sind die PG DASH an der Carl von Ossietzky Universität Oldenburg.

Unsere Projektgruppe (PG) nennt sich *DASH - Data AnalyticS with Hadoop*, weil wir uns ein Jahr mit Apache Hadoop auseinandersetzen werden. Dabei arbeiten wir mit der Firma CEWE Stiftung & Co. KGaA zusammen, welche u.a. für ihre Foto-Dienstleistungen bekannt ist.

Beitrag vom 04.10.2015

Das erste Projektgruppentreffen war uni-intern mit den Teilnehmern der PG und den betreuenden Dozenten Andreas Solsbach und Viktor Dmitriyev. Jeder hat sich kurz vorgestellt und es ist Organisatorisches geklärt worden.

Zum zweiten Treffen haben wir dann versucht, einen gemeinsamen wöchentlichen Jour-Fix Termin zu finden. Hierzu haben wir eine Doodle-Umfrage erstellt. Es ist jedoch sehr schwierig mit 12 Leuten + 2 Dozenten sich auf eine Zeit zu einigen. Zudem muss dann zu dieser Zeit auch noch ein Raum an der Uni frei sein. Wir sind leider nicht drum herum gekommen, dass einige Teilnehmer Abstriche machen mussten. Weil wir Studenten noch alle nebenbei arbeiten, haben wir geschaut, welche Zeiten uns noch bleiben, zu denen keiner arbeiten muss. Dann haben wir auch noch den Wunsch von unseren Dozenten berücksichtigt, dass der Termin bitte irgendwann zwischen 8 und 16 Uhr stattfinden soll. Letztendlich haben wir uns einigen können. Dennoch können nun ein paar Studenten dafür andere Vorlesungen oder sogar Tutorien nicht besuchen.

Beitrag vom 13.10.2015

In der zweiten Woche, waren wir bei der Firma CEWE Stiftung & Co. KGaA. Hier wurden uns unsere Ansprechpartner vorgestellt sowie die Fotostationen, auch Digi-Foto-Maker (kurz: DFM) genannt. Die DFM sind die Kiosk-Geräte von der CEWE Stiftung & Co. KGaA, die bei den Handelspartnern wie z.B. Müller aufgestellt sind. An diesen Geräten können Fotobestellungen aufgegeben oder Fotoprodukte sofort vor Ort ausgedruckt werden. Bei der Bedienung eines DFM werden das Bedienverhalten sowie technische Daten des DFM wie beispielsweise die Temperatur des Druckers in einer XML-Datei abgespeichert.

Unsere Aufgabe für das kommende Jahr ist es, mit Hilfe von Apache Hadoop diese XML-Dateien auszuwerten zu können. Um das realisieren zu können, müssen wir uns zunächst mit Apache Hadoop auseinandersetzen und ein solches Sys-

tem aufsetzen. Neben dieser Einführung in die Thematik hat die CEWE Stiftung & Co. KGaA uns auch schon erste Anforderungen und mögliche Analyse-Themen genannt, was wir im Laufe des Projektes umsetzen sollten. Dies war eine „Near Time“ Analyse, ein Alarm-System um Ausfälle und Störzeiten zu erkennen und mittels einer detaillierten Auswertung des Bedienverhaltens der Bestellsoftware, diese weiter optimieren zu können.

Beitrag vom 18.10.2015

In den ersten Wochen sind wir dabei, die genauen Anforderungen zu erheben. Das erweist sich gar nicht als so einfach. Zu Beginn haben wir nur ein grobes Ziel erhalten. Ansonsten lässt unser Auftraggeber CEWE Stiftung & Co. KGaA uns überaus frei arbeiten.

Um zu bestimmen, wo die Reise hingehen soll, werden wir konkrete Ziele benennen. Hierzu haben wir uns zunächst mögliche Anwendungsfälle überlegt. Außerdem haben wir uns Gedanken zu den Anforderungen sowohl aus technischer Sicht an dem System als auch aus Anwender-Sicht gemacht.

Für die nächste Woche planen wir einen Termin mit CEWE Stiftung & Co. KGaA. Dann wollen wir unsere bisherigen Ergebnisse vorstellen und gemeinsam mit unseren Auftraggebern besprechen, welche Anwendungsfälle unserer Longlist nach Möglichkeit realisiert werden sollen. Zudem sind bei unseren bisherigen Überlegungen einige Fragen aufgekommen, die wir noch beantwortet haben müssen.

Beitrag vom 25.10.2015

Neben der Beantwortung der Frage „Wohin geht die Reise?“, haben wir uns auch schon erste Gedanken dazu gemacht, wie wir innerhalb des nächsten Jahres eigentlich dahin kommen möchten. Hierzu hat sich die PG auf einige organisatorische Punkte geeinigt. Teilweise waren diese Einigungen nicht immer ganz eindeutig und es wurde viel hin und her diskutiert.

Wo wir uns sehr schnell einig waren, war die Vorgehensweise nach Scrum mit Orientierung an CRISP DM. In beiden Fällen werden wir das Vorgehen individuell auf unsere Bedürfnisse anpassen. In Bezug auf Scrum haben wir uns derzeit auf zweiwöchige Sprints geeinigt. Unser wöchentliches Meeting zählt dabei als Weekly-Scrum anstelle eines Daily-Scrums.

Als einen weiteren organisatorischen Punkt haben wir uns für die Verwendung von Jira und Confluence entschieden. Dabei dient Jira zur Aufgabenverwaltung und Confluence für die Dokumentation. Ebenfalls setzen wir ein GIT auf, wobei hier noch ein paar Unklarheiten herrschen, wozu dies gut sein soll, wenn wir doch

auch das Confluence haben. Immerhin brauchen wir nicht in beiden Systemen parallel dokumentieren.

Als kleine Auflockerung zwischendurch haben wir einen Fotoshooting-Termin gemacht. Im Januar folgt noch ein Fotoshooting, wo wir dann auch mal ein Bild mit den Betreuern sowohl von der Universität Oldenburg als auch von CEWE Stiftung & Co. KGaA machen.

Nach einigem hin und her und vielen kreativen Phasen konnten wir uns mittlerweile auch für unseren PG-Namen „DASH“ (Data Analytics with Hadoop) und unser Logo entscheiden.

Beitrag vom 17.11.2015

Nachdem die PG DASH sich einige Wochen detailliert mit möglichen Use-/Business-Cases auseinandergesetzt hat, wurden die konkreten Ziele mit den CEWE Stiftung & Co. KGaA-Betreuern abgestimmt. Dabei wurden folgende Ziele definiert:

Alarmsystem Dieser Begriff hat für einige Verwirrung gesorgt, weswegen eine genaue Definition in unserem Glossar dringend erforderlich war. Es ist nämlich kein reiner Ticketing-Dienst gemeint, der eine Nachricht ausgibt, wenn eine Warnung bzw. ein Alarm vorliegen. Hiermit verstehen wir auch die Logik, die benötigt wird, um eine Warnung bzw. den Alarm überhaupt erst erkennen zu können.

Operationales Dashboard Dies dient vor allem auch zum Einstieg der PG in die konkreten XML-Daten. Das Dashboard dient zur Übersicht einiger Kennzahlen wie bspw. bestellte Produkte pro Tag.

Clickstream-Analyse Unter Clickstream verstehen wir den anonymisiert protokollierten Verlauf der Klicks, die in der Bestellsoftware am Digi-Foto-Maker vorgenommen werden. Mit der Analyse dieser Daten kann die Benutzerfreundlichkeit der Digi-Foto-Maker verbessert werden. Es ist außerdem erkennbar, ob bestimmte Wege immer zu Abbrüchen führen oder ob bestimmte Kombinationen häufig getätigt werden.

Der Schwerpunkt unserer Arbeit soll eher in der Entwicklung eines ETL-Prozesses und in der Datenverwaltung in Apache Hadoop liegen. CEWE Stiftung & Co. KGaA ist es weniger wichtig, welche Daten wir genau analysieren oder wie wir die Ergebnisse optisch aufbereiten.

Für das bessere Verständnis der XML-Daten ist ein Workshop in Zusammenarbeit mit CEWE Stiftung & Co. KGaA für den Januar geplant. Außerdem erhalten wir Zugang zu einem Digi-Foto-Maker, an dem wir selbst einige Testfälle erstellen

können. Außerdem hat CEWE Stiftung & Co. KGaA uns eingeladen, bei dem sogenannten „CEWE Analytics Forum“ einen Vortrag zu halten. Dies ist eine CEWE Stiftung & Co. KGaA-interne Veranstaltung, bei der unser Projekt sogar dem Vorstand vorgestellt werden würde.

Beitrag vom 10.12.2015

Als ersten Schritt in Hadoop wollen wir uns für die benötigten Komponenten entscheiden. Weil Hadoop sehr umfangreich ist und über viele verschiedene aber ähnliche Komponenten verfügt, müssen wir einige testen und näher untersuchen. Hierbei zählt vor allem eine wissenschaftliche Herangehensweise, um eine Entscheidung vornehmen zu können. Hierfür hat sich die Projektgruppe in zwei Teilgruppen aufgeteilt. Die erste Teilgruppe beschäftigt sich mit Komponenten für die Schnittstellen-Extraktion und Laden der Daten. Die zweite Teilgruppe befasst sich mit Komponenten für Transformationsstrategien. Somit wurde der ELT-Prozess aufgeteilt.

Sobald wir die Komponenten ausgewählt haben, wollen wir in die nächste Phase übergehen. Das heißt, hier wird sich dann mit dem Thema der Analytik beschäftigt. Dies bezieht Data Mining und Maschinenlernen mit ein.

Es ist geplant, die finale Auswahl der zu installierenden Komponenten mit CEWE Stiftung & Co. KGaA abzustimmen. Obwohl wir zunächst alles Uni-intern installieren, ist zum Frühjahr 2016 ein Umzug der Architektur in die Systemlandschaft von CEWE Stiftung & Co. KGaA geplant.

Jahresrückblick 2015

Im Oktober 2015 begann die Projektgruppe. Als erstes war das große Kennenlernen mit den anderen Projektgruppenteilnehmern und den betreuenden Dozenten angesagt. Danach lernten wir unsere Ansprechpartner bei der Firma CEWE Stiftung & Co. KGaA kennen und uns wurden die DigiFotoMaker (kurz: DFM) vorgestellt. Die DFM sind Kiosk-Geräte der CEWE Stiftung & Co. KGaA, die bei ihren Handelspartnern wie beispielsweise Müller Großhandels Ltd. & Co. KG aufgestellt sind. An diesen Geräten können Fotoprodukte bestellt und direkt vor Ort ausgedruckt werden. Bei der Bedienung eines DFM werden neben technischen Daten auch Informationen zum Bedienverhalten in einer XML-Datei gesichert. Unsere Aufgabe ist es, diese XML-Datei mit Hilfe von Apache Hadoop auswerten zu können.

Bevor wir an die eigentliche Bearbeitung unseres Projekts gehen konnten, mussten wir uns einigen, wie wir uns innerhalb des nächsten Jahres organisieren wollen. Wir einigten uns sehr schnell bei dem Verfahren zur Softwareentwicklung auf

SCRUM, weil wir das alle kennen, mit dem Ablauf vertraut sind und uns damit wohl fühlen. Als Tool zur besseren Organisation wählten wir Jira mit dem dazugehörigen Dokumentations-Tool Confluence. Zusätzlich verwenden wir GIT zum Dateiaustausch. Außerdem wurden einige Rollen verteilt, so haben wir nun zwei Projektmanager, zwei Serverbeauftragte, einen Dokumentenverantwortlichen und einen Webseitenverantwortlichen.

Nachdem das Organisatorische geklärt war, wurden die Anforderungen erarbeitet und im Detail definiert. Dabei stellten wir drei Kern-Anforderungen heraus, die hier kurz erwähnt werden.

- „Near Time“ Analyse wichtiger Kennzahlen
- Alarm-System, welches Ausfälle und Störzeiten erkennt
- Auswertung des Bedienverhaltens zur Optimierung der Bestellsoftware

Als nächsten Schritt evaluierten wir, welche Komponenten des Apache Hadoop Ecosystems wir verwenden werden. Dazu teilte sich die PG in zwei Teilgruppen auf. Die erste Teilgruppe untersuchte Komponenten für die Schnittstellen-Extraktion und das Laden der Daten in das Hadoop Distributed File System (HDFS). Die zweite Teilgruppe beschäftigte sich mit Komponenten für das Transformieren der XML-Dateien. Somit wurde der klassische ELT-Prozess aufgeteilt. Am letzten Termin vor den Weihnachtsferien stellten die einzelnen Teilgruppen den Zwischenstand ihrer Ergebnisse vor.

Jahresausblick 2016

Nachdem in 2015 die Projektgruppe mit der Einarbeitung in die Thematik begonnen hatte, die Anforderungen erhoben wurden und das Jahr mit der Evaluierung der Komponenten von Apache Hadoop endete, geht es in 2016 mit der Entwicklung eines ersten Prototypen weiter.

Insgesamt teilt sich das Jahr 2016 in Hinblick auf die Projektgruppe in zwei Zeiträume. Der erste Zeitraum geht bis April und umfasst die Entwicklung eines ersten Prototypen (Alpha-Version). Der zweite Zeitraum ist ab April bis September und behandelt die Entwicklung des zweiten Prototypen (Beta-Version) als Ergebnis der Projektgruppe. Der große Hauptunterschied zwischen den beiden Versionen ist, dass die Alpha-Version auf Servern innerhalb der Carl von Ossietzky Universität Oldenburg läuft und die Beta-Version in die Systemumgebung von der CEWE Stiftung & Co. KGaA integriert wird.

In der Alpha-Version haben wir noch die Möglichkeit, alles auszuprobieren und zu testen. Wir können sämtliche Komponenten installieren und von verschiedensten Gesichtspunkten evaluieren. Und genau das wird auch noch viel geschehen.

Im April zieht dann unser System von den Servern der Carl von Ossietzky Universität Oldenburg zu den Servern der CEWE Stiftung & Co. KGaA. Hier findet dann die Beta-Version statt. Die Komponenten, welche wir letztendlich verwenden wollen, sollten somit bis April feststehen, weil in der Beta-Version keine Komponenten getestet werden sollen.

Wir haben geplant uns ab März mit der Analyse der Daten auseinanderzusetzen. Hierbei unterscheiden wir zwei Formen der Analyse. Zum einen Analytik an sich und zum anderen Data Mining. Unter Analytik verstehen wir dabei eine reine Ermittlung der Kennzahlen. Mit Data Mining meinen wir sämtliche Analyse-Themen, die komplexer sind.

Um jedoch die Daten überhaupt analysieren zu können, müssen wir die XML-Daten extrahieren, in das Hadoop Distributed File System (HDFS) laden und die Daten transformieren. Mit dieser Thematik haben wir bereits im Dezember 2015 begonnen und führen sie noch im Januar und Februar 2016 weiter.

Beitrag vom 13.01.2016

Nach den Weihnachtsferien untersuchen wir weiter, welche der verschiedenen Komponenten des Apache Hadoop Ecosystems am besten für unser Vorhaben geeignet sind. Dabei betrachten wir folgende Komponenten im Detail:

Für das Extrahieren und Laden der Daten werden die Tools Apache Kafka, Elasticsearch, Logstash, Fluentd, Apache Storm und Apache Flume untersucht. Bei Apache Storm sind wir auf Konfigurationsprobleme im Bereich Apache ZooKeeper gestoßen. Die Konfiguration von Apache Storm und Nimbus ist erfolgreich verlaufen. Elasticsearch ist für unsere Anforderungen nicht geeignet. Die Tools Logstash, Apache Flume und Fluentd haben relativ ähnliche Leistungsmerkmale, weswegen hier nun der Installations- und Konfigurationsaufwand sowie die Kompatibilität mit anderen Tools verglichen werden soll.

Zum Transformieren und Parsen der Daten werden die Tools Apache Hive, Apache Pig und Apache Spark untersucht. Apache Spark wird dabei mit den verschiedenen Programmiersprachen Java, Scala und Python betrachtet. Dabei hat sich bisher gezeigt, dass Apache Pig für unser Anliegen ungeeignet ist, weil sich hier keine einzelnen Attribute aus der XML-Datei auslesen lassen. Mit Apache Hive und Apache Spark lassen sich die Daten für unsere Anliegen parsen. Da Apache Spark jedoch einfacher zu konfigurieren und zu verwenden ist, wird sich hierfür entschieden.

Neben der Komponenten-Auswahl beschäftigt sich ein Teil unserer Gruppe dieser Tage intensiv mit den XML-Dateien. Ziel ist es eine Dokumentation zum Verständnis der XML-Dateien zu erstellen. Derzeit wissen wir noch nicht, wofür jedes

einzelne Element und jedes Attribut steht bzw. welche Informationen sich dahinter verbergen. Um dies in Zukunft sagen zu können, wird eine Dokumentation erstellt und mit den Mitarbeiter des Praxispartners besprochen. Dies ist eine notwendige Grundlage für die zukünftige Analyse der Daten.

Neben diesen Tätigkeiten ist uns aufgefallen, dass unsere bisherige Organisation einiges an Verbesserungspotential aufweist. So wird bisher noch nicht in vollem Umfang mit JIRA gearbeitet, was sich dadurch zeigt, dass die Aufgaben nicht immer richtig zugewiesen werden und die Aufgaben im System allgemein wenig aktuell gehalten werden. Oftmals hat die Person, die für die Entwicklung zuständig war, die Ergebnisse auch selbst getestet. Dadurch war das bisherige Testen der Ergebnisse nicht wirklich gegeben. Jetzt wird direkt nach Abschluss einer Aufgabe diese einer anderen Person, die sich mit der Thematik der Aufgabe ebenfalls auskennt, zum Testen zugewiesen. Somit erhält der Bearbeiter der Aufgabe mehr Feedback. Wir wollen dadurch bessere Ergebnisse erzielen. Damit das Tool wirklich unterstützend wirkt, müssen wir mehr darauf achten, die Aufgaben aktuell zu halten und den richtigen Personen zuzuweisen. Nur dann kann es auch einen positiven Einfluss auf unsere Arbeitsergebnisse haben.

Beitrag vom 30.01.2016

Die zwei Serverbeauftragten aus unserer Gruppe installieren das Hadoop-Ecosystems auf den Servern der Carl von Ossietzky Universität Oldenburg für unsere Alpha-Version.

Für das Extrahieren und Laden der Daten werden die Tools Apache Kafka, Apache Flume, Logstash und Fluentd in Hinblick auf ihren Installations- und Konfigurationsaufwand sowie ihre Kompatibilität mit anderen Tools untersucht. Mittlerweile zeigt sich, dass Fluentd und Logstash für unser Projekt nicht geeignet sind. Es können Daten vom FTP-Server mit Apache Flume ins HDFS übertragen werden. Jedoch gibt es hier einige Probleme bei der Konfiguration von Apache Flume. Apache Kafka wird weiterhin untersucht. Einige Teammitglieder haben ein neues Tool namens Gobblin gefunden und untersuchen dieses. Als Backup-Lösung hat ein Teammitglied auch noch ein Script selbst geschrieben, welches problemlos die XML-Dateien vom FTP-Server zum HDFS übertragen kann.

Es wird beschlossen, zum Parsen der XML-Dateien einen selbstgeschriebenen Parser zu verwenden. Hierbei wird die Programmiersprache Python genutzt. Neben dem Programmieren des Parsers wird bereits eine relationale Datenstruktur entworfen. Diese soll für den Aufbau von Hive-Tabellen genutzt werden. Es ist angedacht, dass der Parser entsprechende csv-Dateien erstellt, die in die Hive-Tabellen gelesen werden.

Von unseren Betreuern kommt der Hinweis, dass Apache Hadoop mit vielen kleinen Dateien nicht so performant arbeitet wie mit größeren Dateien. Somit werden wir versuchen, einige kleine Dateien zu einer großen zusammenzufassen. Zudem wird der Tipp gegeben zum Parsen der XML-Datei die Anwendungen Hadoop Streaming und MapReduce zu nutzen. Auch das untersuchen wir.

Außerdem ist uns aufgefallen, dass wir zwar nach SCRUM arbeiten wollen, bisher aber noch keinen vernünftigen Sprint-Review oder eine Retrospektive bei den bisherigen Sprints gemacht haben. Unsere Sprints haben eine Laufzeit von zwei Wochen und wir sind bereits im vierten Sprint. Dies wollen wir bei dem Abschluss des jetzigen Sprints ändern.

Beitrag vom 12.02.2016

Nach etwa der Hälfte der Projektgruppen-Zeit finden erste Feedback-Gespräche statt. Jedes Teammitglied erhält ein Einzelgespräch mit den Betreuern um einerseits die aktuelle Note bzw. Tendenz zu erfahren und andererseits etwaige Probleme oder Besonderheiten zu erwähnen.

Da wir beim letzten Mal festgestellt hatten, dass der Umgang mit JIRA noch nicht perfekt bei uns funktioniert, findet nun ein Frühjahrsputz in unserem JIRA-System statt. Einige Teammitglieder übernehmen diese Aufgabe. Dies beinhaltet eine Überarbeitung der Epics (Themengebiete im JIRA) sowie eine neue Namensgebung der verschiedenen Status, damit genau zwischen „in Test“ und „fertig“ unterschieden wird. „Fertig“ ist eine Aufgabe erst dann, wenn sie getestet worden ist.

Hadoop Streaming wird untersucht. Hiermit ist es möglich, Python Skripte auf dem Cluster zu verteilen. Als Alternative wird das manuelle Verteilen der Skripte untersucht. Der Parser an sich wird weiterentwickelt. Zudem werden Hive-Tabellen erstellt. Das Tool Gobblin für das Extrahieren und Laden der Daten wird untersucht. Da dieses Tool noch relativ neu bzw. in der Entwicklung ist, gibt es hier einige Probleme.

Es wird ein erstes Social Event für Ende des Monats geplant. Voraussichtlich werden wir das in Zukunft regelmäßig machen.

Beitrag vom 29.02.2016

Das erste Statusmeeting in 2016 mit der CEWE Stiftung & Co. KGaA findet statt. Es wird der aktuelle Stand der beiden Teilgruppen sowie die bisherige Komponentenauswahl vorgestellt. Die CEWE Stiftung & Co. KGaA nennt uns das Sizing der Hadoop-Landschaft für die Beta-Version. Zudem wird geplant, dass die Projektgruppe im Juni eine Präsentation bei einer CEWE-internen Veranstaltung, dem

sogenannten Analytik-Forum hält.

Das Tool Gobblin wird weiterhin untersucht. Als Alternative zum Hadoop Streaming wird Apache Spark untersucht. Dabei kann der Parser in Apache Spark aufgerufen werden. Mit dem Python-Parser lassen sich die bisherigen XML-Dateien parsen. Dabei werden die Ergebnisse in csv-Dateien nach dem entworfenen Datenbankmodell gesichert. Jedoch ist die Implementierung bisher noch statisch. Spätestens in der Beta-Version soll hier eine (teilweise) dynamische Lösung vorliegen. Der Import der CSV-Dateien in die Hive-Tabellen funktioniert. Nun wird geprüft, ob das Workflow-Tool Apache Oozie geeignet ist um zu prüfen, ob neue CSV-Dateien vorhanden sind. Es ist geplant, ein Logging-System einzurichten. Dies hilft bei Fehlern zu ermitteln, in welcher Komponente bzw. bei welchem Schritt der Fehler aufgetreten ist.

Es werden Programmierrichtlinien erstellt. Obwohl wir bereits mit der Programmierung begonnen haben, ist es noch nicht zu spät hierfür. Noch ist der produzierte Quellcode überschaubar und kann leicht überarbeitet werden.

Damit am Ende des Projekts keine reine Doku-Phase entsteht, werden die bisherigen Ergebnisse bereits in einem Dokument für die Abschlussdokumentation zusammengetragen.

Jedes Teammitglied fängt an, sich mit den Themen Data Mining und Visualisierung bzw. Dashboard auseinanderzusetzen, damit die kommenden Aufgaben entsprechend verstanden und verteilt werden können. Es werden demnächst zwei neue Teilgruppen gebildet. Die eine wird sich dann mit dem Thema Data Mining und die andere mit den Themen Visualisierung und Dashboard beschäftigen.

Das erste Social Event der Projektgruppe fand in der 3Raumwohnung statt. Dort haben wir Tischkicker gespielt, uns unterhalten und einfach Spaß gehabt.

Beitrag vom 18.03.2016

Neben den bestehenden beiden Teilgruppen für den ELT-Prozess wurden jetzt zwei neue Teilgruppen ins Leben gerufen, um offene Tätigkeiten innerhalb der PG zu erledigen:

- Teilgruppe Dashboard und Visualisierung
- Teilgruppe Data Mining

Somit existieren aktuell vier Teilgruppen à drei Personen in der PG.

Für das Laden der XML-Daten in das HDFS konnte keine passende Anwendung identifiziert werden, sodass die Entscheidung getroffen wurde, diese Tätigkeit selbst zu implementieren: „Agent for XML Transportation“ (AXT). Leider stellte sich erst im Nachhinein bei der Evaluierung einzelner Funktionalitäten anstatt zu

einem früheren Zeitraum heraus, dass kein Tool existiert, welches die Anforderungen erfüllt und die Entwicklung in Eigenregie erfolgen muss.

Unabhängig von der Implementierung ist ein Ziel der PG der Wissenserwerb mit den eingesetzten Methoden und Tools, das sich beim Abschluss des siebten Sprints zeigte. In diesem wurde festgestellt, dass Confluence die Möglichkeit der Erstellung einer Retrospektive hat, die ab sofort zur Verbesserung zukünftiger Sprints eingesetzt wird.

Die aktuelle Planung beinhaltet den Meilenstein 07.04.2016: ELT-Prozess Durchlauf auf der Alpha Version. Hierzu müssen unsere selbst entwickelten Tools wie z.B. AXT oder der Parser bis dahin fertig und lauffähig sein. Außerdem müssen alle beteiligten Komponenten über ein Workflowmanagementtool miteinander verbunden werden.

Beitrag vom 04.04.2016

Die Teilgruppe „Dashboard und Visualisierung“ hat sich in den letzten zwei Wochen mit der Evaluierung von Tools für die Softwareauswahl beschäftigt. Die Entscheidung, welches Tool zur Visualisierung und für unser Dashboard verwendet werden soll, fiel auf Apache Zeppelin. Es erfüllt die gestellten Anforderungen und erlaubt eine einfache Handhabung.

Die weiteren Teilgruppen sind mit der Einhaltung der Fristen für den Meilenstein am 07.04.2016 beschäftigt, der den kompletten Durchlauf des ELT-Prozesses beinhaltet (siehe Grafik A.19 in vier Schritten).

1. Im ersten Schritt werden die XML-Daten von dem FTP-Server mittels unserer selbst geschriebenen Anwendung AXT (Agent for XML Transportation) in das HDFS übertragen. Dabei gibt es im HFDS eine baumartige Ordnerstruktur, die sich nach Jahren, Monaten und Tagen untergliedert.
2. Anschließend werden die XML-Daten vorbereitet. D.h. entweder, dass mehrere XML-Dateien zu einer XML-Datei zusammengefasst werden und in den Ordner ready2parse ins HDFS gespeichert werden, oder, dass die Original-XML-Dateien in den Ordner ready2parse kopiert werden. Dieser Schritt geschieht mit unserem eigenen Tool PreX (Prepare XML).
3. Die XML-Dateien aus dem Ordner ready2parse werden geparst. Die Ergebnisse sind dann für den vierten Schritt in dem Ordner parsedData im HDFS zu finden.
4. Abschließend werden im letzten Schritt die Daten mit Hilfe von unserer selbst geschrieben Anwendung HIMP (Hive Importer) in Hive-Tabellen geladen.

Der gesamte Workflow wird mit dem Workflowmanagementsystem Airflow kontrolliert.

Beitrag vom 18.04.2016

Manchmal können im Projektalltag, insbesondere im Rahmen einer studentisch organisierten Gruppe, trotz reiflicher Planung und Abstimmung gesetzte Ziele nicht erreicht werden.

So konnte unser Ziel eines kompletten ELT-Laufs aufgrund von Problemen innerhalb von AXT (Agent for XML Transportation) bis zum gesteckten Meilenstein nicht erfüllt werden. Intensive Recherchen ergaben, dass beim Übertragen von größeren Dateien der FTP Server einen Fehlercode sendete. Dieser Fehler wurde von der verwendeten Methode der Apache commons.net Library nicht korrekt verarbeitet, wodurch es zu einer Art Endlosschleife kam.

Damit die nachfolgenden Meilensteine allerdings nicht beeinträchtigt werden, wird parallel in der Projektgruppe an Anforderungen und Ideen für erste Analysen und Hypothesen geforscht.

Außerdem zeigte sich, dass im Umgang mit GIT zur Verwaltung unserer Daten einzelne Repositories für die verschiedenen Komponenten notwendig sind. Dies ist begründet durch die Anforderungen an die einzelnen Komponenten und deren Wachstum. Es resultiert eine neue Struktur in unserem GIT: AXT (Agent for XML Transportation), Parser, HIMP (Hive Importer; ehemals Watchdog), Airflow, Tests, Analytics, Dokumentation, Sonstiges.

Des Weiteren müssen wir uns in Zukunft angewöhnen, Änderungen im Quellcode zunächst auf einem Branch vorzunehmen. Erst wenn die Änderungen erfolgreich getestet worden sind, können sie auf dem Master-Branch gepushed werden. Dies dient dazu, dass der Master-Branch immer stabil ist und eine lauffähige Version enthält.

Beitrag vom 02.05.2016

Oftmals ist es in der Software-Entwicklung so, dass wenn ein Fehler behoben werden konnte, der nächste auftritt. Ähnliches haben wir bei unserem ELT-Prozess feststellen müssen. Unser AXT (Agent for XML Transportation) läuft jetzt einwandfrei, weil die fehlerhafte Methode durch eine andere Methode ausgetauscht worden ist. Allerdings haben wir nun Schwierigkeiten beim Parsen der Daten. Sowohl bei der Variante mit Hadoop Streaming als auch bei der Alternative mit Spark haben wir Out-of-Memory-Probleme, sodass wir bisher nur kleine Datenmengen parsen können.

Damit dennoch die anderen Ziele erreicht werden können, arbeitet sich ein Teil der Projektgruppe bereits in die XML-Dateien in Hinblick auf Data Understanding ein. Es werden sogar schon erste Analysen an den vorhandenen Daten durchgeführt.

Zudem haben wir nun Testbeauftragte in unserer Projektgruppe definiert, die das Testmanagement übernehmen. Hierzu werden anfangs Testdrehbücher für die einzelnen Komponenten erstellt.

Organisatorisch betrachtet wollen wir uns neu aufstellen. Bisher treffen wir uns jeden Donnerstag für den gesamten Tag, um intensiv in der Gruppe bzw. in Teilgruppen an dem Projekt arbeiten zu können. Diese gemeinsame Zeit ist sehr wichtig, weil viel diskutiert und besprochen wird. Allerdings fehlen uns oftmals 1-2 Stunden, in denen Organisatorisches geklärt wird wie z.B. Weekly Scrum, Sprintabschluss und Sprintplanung. Diesen organisatorischen Teil wollen wir nun auf einen anderen Tag verschieben, damit wir donnerstags noch produktiver sein können.

Bei unserem zweiten Social Event waren wir in der Barcelona Finca.

Beitrag vom 16.05.2016

In unserem ELT-Prozess ist nach AXT (Agent for XML Transportation) die Komponente PreX (Prepare XML) geplant gewesen, um die XML-Dateien zu mergen und dann die gemergten Dateien an den Parser zu übergeben. Nun haben wir allerdings beschlossen, die Funktion von PreX nicht separat in einer Komponente zu implementieren, sondern in AXT aufzunehmen.

Weiterhin besteht noch die Problematik, große Datenmengen mittels Hadoop Streaming und Spark zu parsen. Deswegen werden nun einige Mitglieder aus dem Analysen-Team das Parser-Team unterstützen und diese Arbeit weiter vorantreiben.

Die CEWE Stiftung & Co. KGaA hat uns angeboten, dass wir unsere bisherigen Ergebnisse auf dem CEWE Analytics Forum vorstellen dürfen. Wir wollen dieses Angebot annehmen und arbeiten hierzu eine Präsentation aus, welche weniger auf die technischen Details, sondern mehr auf die Ergebnisse eingeht, die mit unserer Arbeit erzielt werden können.

Zeitgleich mit der intensiven Phase der Entwicklung ist nun der Termin für das Projektgruppen-Boule-Turnier bekannt gegeben worden. Wir nehmen natürlich daran teil. In Vorbereitung zur Stärkung des Teamgeistes haben wir noch ein Fußballspiel gegen die Projektgruppe DOHA organisiert.

Beitrag vom 30.05.2016

Das Problem, welches beim Ausführen unseres XML-Parsers auftritt, ist noch nicht vollständig behoben. Es besteht weiterhin die Problematik, dass XML-Dateien beim Übersetzen etwa die hundertfache Menge der eigenen Dateigröße an Hauptspeicher benötigen. Die Ursache für das Problem konnte inzwischen jedoch identifiziert werden: Die verwendete API xml.minidom ist für große XML-Dateien ungeeignet. Es wird nun nach einer möglichen Alternative gesucht.

Nach ausgiebiger Recherche konnte hierfür kein Benchmark gefunden werden, weswegen mit Unterstützung von unserem Betreuer V. Dmitriyev ein eigener Benchmark durchgeführt wird. Hierfür werden die APIs xml.sax, xml.etree.ElementTree, xml.etree.cElementTree, lxml.objectify, lxml.etree, xml.dom.pulldom und die ursprüngliche API xml.minidom miteinander verglichen. Um die verschiedenen APIs objektiv miteinander vergleichen zu können, werden für jede API minimale Anwendungsszenarien für alle Bibliotheken auf verschieden große XML-Dateien implementiert. Dabei werden Messgrößen wie z.B. der benötigte RAM beim Laden und beim Parsen der XML-Datei als auch jeweils die benötigte Zeit miteinander verglichen.

Es besteht die Möglichkeit, dass die Ergebnisse des Benchmarking ergeben, dass der bisher implementierte XML-Parser der Projektgruppe abgeändert werden sollte.

Die Migration der Alpha Version zur Beta Version ist für den kommenden Monat geplant. Die Alpha Version befindet sich derzeit im Netzwerk der Carl von Ossietzky Universität Oldenburg. Die Beta Version wird im Netzwerk der CEWE Stiftung & Co. KGaA liegen.

Beitrag vom 10.06.2016

Die CEWE Stiftung & Co. KGaA hatte uns zum CEWE Analytics Forum eingeladen, welches am 07. Juni stattgefunden hat. Hierzu haben wir eine Präsentation gezielt für das Publikum bestehend aus Vorstand und Abteilungsleitern erstellt. Die bisherigen Ergebnisse der Projektgruppe sind von dem Publikum positiv aufgenommen worden.

Am 08. Juni und 09. Juni fand an der Carl von Ossietzky Universität Oldenburg eine bundesweite und fächerübergreifende Konferenz für studentische Forschung statt. Die Veranstaltung nennt sich „forschen@studium“. Hier können Studenten ihre Forschungsergebnisse aus Hausarbeiten, Abschlussarbeiten oder Projektarbeiten präsentieren. Weiter Informationen zu der Konferenz sind unter zu finden.

Um an der Konferenz teilnehmen zu können, mussten alle interessierten Teilnehmer ihre Forschung in einem Abstract, welcher den Umfang von einer Seite

hatte, darstellen. Auch wir haben einen Abstract verfasst, welcher angenommen und zusammen mit den anderen angenommen Abstracts in dem Tagungsband der Konferenz „forschen@studium“ veröffentlicht wurde. Unser Abstract ist auf Seite 106 zu finden.[1] Zusätzlich streben wir eine Veröffentlichung in dem studentischen Online-Journal „forsch!“ an.

Für die Konferenz in Oldenburg haben wir unsere Ergebnisse in Form eines Posters im Rahmen einer Postersession vorgestellt. Hierbei konnten wir einige Interessenten über unser Projekt informieren. Dabei haben wir ebenfalls positives Feedback erhalten.

Beitrag vom 17.06.2016

Der ELT-Workflow läuft mittlerweile erfolgreich. Im nächsten Schritt wird der Workflow automatisiert. Der Fokus liegt nun allerdings wieder auf den Analysen der Daten und dem Data Mining. Hier ist in den nächsten Wochen der Aufbau eines Dashboards geplant.

Nachdem die Testdrehbücher für die einzelnen Komponenten verfasst und überprüft worden sind, werden die selbst entwickelten Komponenten nun gründlich auf Herz und Nieren geprüft.

Es ist aufgefallen, dass die Dokumentation in letzter Zeit ein wenig auf der Strecke geblieben ist. Um dies nun wieder zu ändern, hat unser Doku-Beauftragter das Zepter in die Hand genommen und einige Aufgaben bzw. Unterkapitel konkret bestimmten Personen zugeteilt. Die Struktur der Kapitel zur Evaluation der Tools wurde überarbeitet, sodass diese Kapitel ebenfalls angepasst werden müssen. Auch hierzu hat der Doku-Beauftragte gleich Personen benannt, damit sich hierfür jemand verantwortlich sieht.

Beitrag vom 27.06.2016

Am 23. Juni fand das Projektgruppen-Boule-Turnier des Departments für Informatik der Carl von Ossietzky Universität Oldenburg statt. Hierzu haben sich die neun aktuell laufenden Projektgruppen, die wissenschaftlichen Mitarbeiter, die OLDIES sowie die Titelverteidiger im Vahlenhorster Wäldchen zusammengefunden. Insgesamt waren wir 12 Mannschaften.

Obwohl wir uns mehrmals vorgenommen hatten im Vorfeld zu üben, haben wir es nie geschafft. Dennoch haben wir uns relativ gut geschlagen.

Unser Teamname war „Epic Data“. Der wichtigste Gegner von uns war die Mannschaft „WoranHatEsGelegen?“, auch bekannt als die PG DoHa. Beide Pro-

[1] Siehe Webseite: `https://www.uni-oldenburg.de/fileadmin/user_upload/flif/Konferenz2016/Broschur_Forschen_A4_Internet.pdf`

jektgruppen finden im Rahmen der VLBA Abteilung statt. Einer der härtesten Gegner war das Siegerteam des letzten Jahres „Meine Favoriten“.

Die Gruppenphase fing für uns sehr gut an. Wir haben die ersten beiden Spiele gewonnen und die nächsten zwei verloren. Das letzte Spiel der Gruppenphase fand dann gegen „WoranHatEsGelegen?“ statt. Hier haben wir wieder souverän gewonnen. Damit haben wir es insgesamt mit 6 Punkten gerade noch so geschafft, ins Viertelfinale zu kommen.

Im Viertelfinale war unser erster Gegner der Titelverteidiger „Meine Favoriten“. Hier war das Spiel wirklich sehr spannend und der Sieg knapp. Durch „Auswerfen“, vergleichbar mit dem Elfmeter vom Fußball, konnten wir gerade so gewinnen. Im Halbfinale hatten wir wieder ein knappes Spiel gegen „Team ScrVM“, welches ebenfalls durch Auswerfen entschieden wurde. Leider ging der Punkt dieses Mal nicht an uns, sodass wir anschließend gegen die „Taxi Drivers“ um den dritten Platz kämpften.

Bei der Siegerehrung auf dem Sommerfest der Informatik konnten wir dann anschließend unsere Urkunde für den dritten Platz entgegennehmen. Den zweiten Platz hat das „Team ScrVM“ (Projektgruppe RCCARS der Abteilungen Fränzle/Damm) gemacht. Der Pokal ging an die „Powder Rangers“, welche die Gruppe der OLDIES ist.

Weiter Infos zum Turnier sind unter `http://tinyurl.com/jt88adc` zu finden.

Beitrag vom 14.07.2016

Unsere Hardware ist aufgestockt worden. Der Hauptspeicher des Masters wurde von 16 GB auf 32 GB und der Speicher der Slaves jeweils von 8 GB auf 16 GB erhöht.

Mittlerweile haben wir es geschafft, neben dem Standard-Workflow auch einen Repeat-Workflow zu realisieren. Beide sind noch nicht automatisiert. Im Gegensatz zum Standardworkflow werden beim Repeat-Workflow die XML-Daten nicht vom FTP-Server, sondern direkt aus dem HDFS gelesen, geparst und entsprechend weiterverarbeitet.

Das Benchmarking der verschiedenen APIs, die für das Parsen verwendet werden können, hat ergeben, dass xml.sax am sinnvollsten für unser Anliegen ist. Deswegen wird aktuell ein neuer Parser entwickelt, der die API xml.sax verwendet.

Das Analyse-Team hat einen ersten Entwurf des Dashboards fertig gestellt. Hier werden unter anderem folgende Kennzahlen präsentiert Umsatz, Anzahl Verkäufe, Top 5 Produkte, Top 5 Handelspartner. Für die Clickstream-Analyse ist es geplant einerseits die „Trampelpfade“ und andererseits häufige Abbruchstellen zu

identifizieren.

Beitrag vom 30.07.2016

So langsam gehen wir auf den Endspurt zu. Dies fällt besonders daran auf, dass unser Dokumentationsbeauftragter allen auf die Finger klopft, schon einmal mit der Projektdokumentation zu beginnen. Hierzu hat er den roten Faden und die Struktur der Gesamtprojektdokumentation vorgestellt. Anschließend wurden die noch ausstehenden Kapitel Personen zugewiesen, welche dafür Sorge tragen, dass hierzu Inhalte entstehen. Damit wir nicht in Zeitnot geraten, haben wir uns darauf geeinigt, dass zu jedem Kapitel bis spätestens Anfang September der erste Entwurf ausformuliert werden sollte. So ist noch ausreichend Zeit, die Texte zu prüfen und ggf. zu überarbeiten.

Neben der Projektdokumentation ist auch ein Plan für die Migration der entwickelten ELT-Strecke mit allen Einstellungen und Tools von dem Universitäts-Cluster zum CEWE-Cluster mit detaillierten Terminen entstanden. Die Migration sollte diese Woche (KW 30) stattfinden.

Der neue Parser SaPPy (Sax Parser on Python), welcher die API xml.sax verwendet, ist fertig gestellt worden. Die ersten Daten wurden mit ihm geparst. Es folgt noch die Implementierung einiger Nice-to-have-Anforderungen. Währenddessen wird SaPPy schon im Rahmen des Testmanagements überprüft.

Das Analyse-Team überarbeitet das Dashboard. Es werden nun neue Kennzahlen ermittelt, die die CEWE Stiftung & Co. KGaA bisher noch nicht ermitteln konnte, weil ihnen hierzu die Detail-Informationen aus der XML-Datei fehlten.

In der letzten Woche gab es Probleme mit unserem Cluster im Uni-Netzwerk. Die Server waren nicht erreichbar, was die Arbeiten ziemlich erschwert. Dies wirkte sich allerdings positiv auf unsere Dokumentation aus.

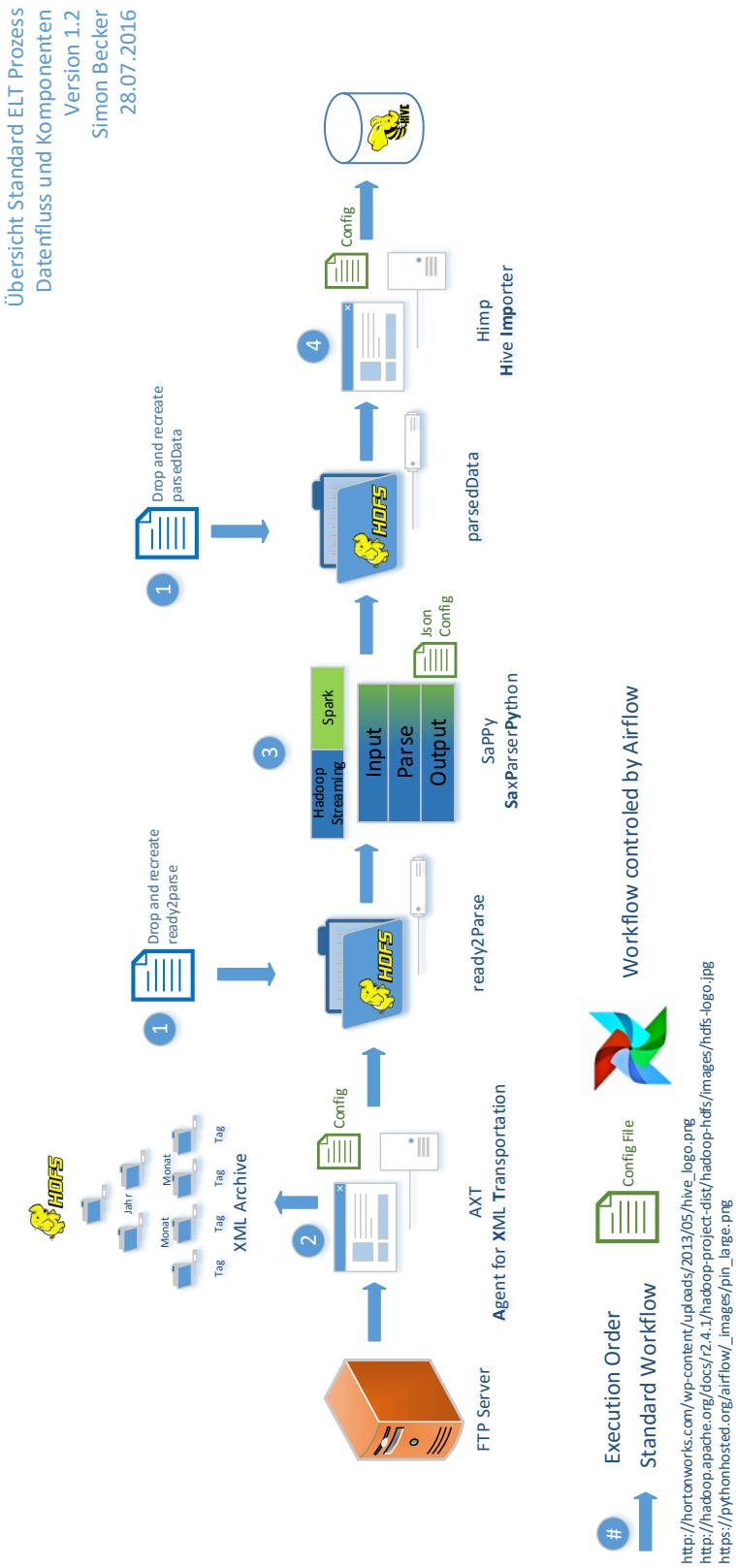


Abbildung A.19.: Standard-ELT-Workflow

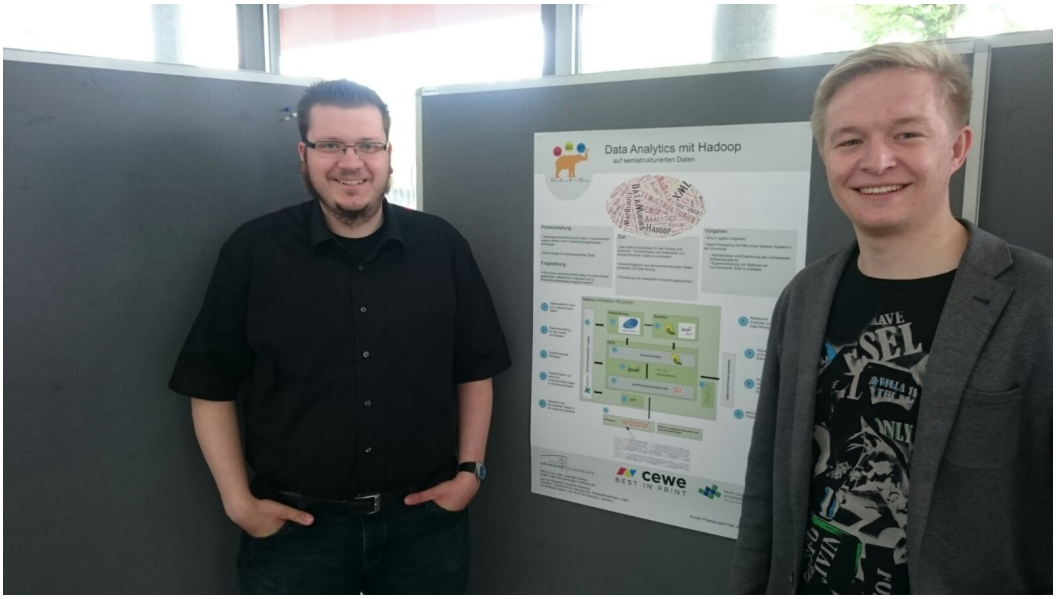


Abbildung A.20.: Quelle: Fotografiert von Simon Becker

Abbildung A.21.: Boule-Foto. Quelle: Fotografiert von Dr. Dietrich Boles (`http://www.boles.de/teaching/pg_fb10/boule/2016/Fotos-Web/index.html`)

Abbildung A.22.: Mannschaft „Epic Data“ von PG DASH. Quelle: PG-DASH

Abbildung A.23.: Unsere beiden besten Spieler (v.l. Felix Kruse, Felix Ivo Schoelzel) beim Auswerfen gegen „Meine Favoriten“. Quelle: PG-DASH

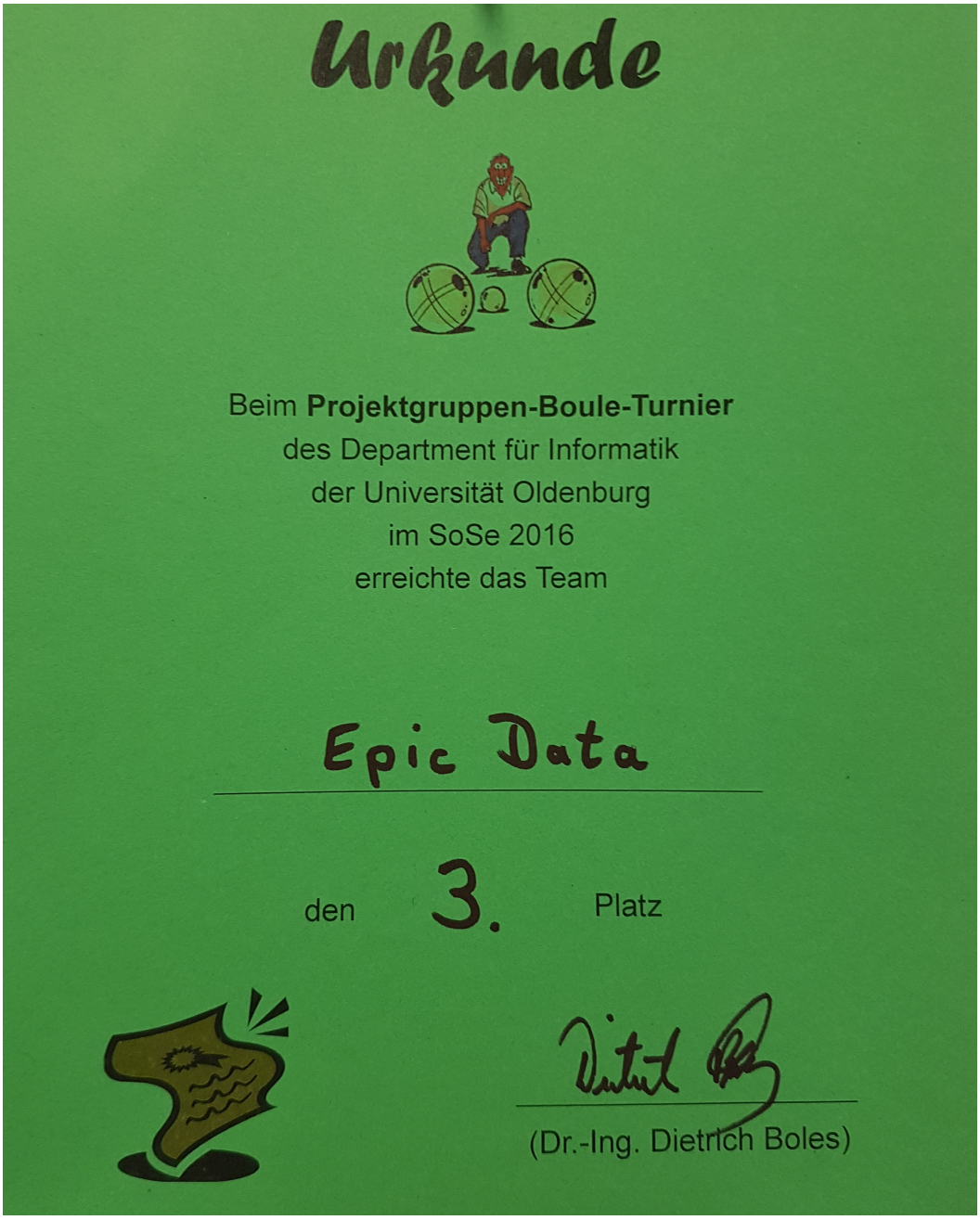


Abbildung A.24.: Quelle: PG-DASH

A.5. Forschen@Studium

Einreichungskennung: 2016-conf1-113

Einsatzmöglichkeit der Big Data Technologie Hadoop Ecosystem in Unternehmen -

am Beispiel von semistrukturierten Daten

Unternehmen besitzen schnell wachsende und vielfältig strukturierte Datenmassen – Big Data (Volume, Velocity and Variety 3Vs). Ziel der Unternehmen ist es aus ihren Datenmassen betriebswirtschaftlich nutzbare Erkenntnisse zu gewinnen. In diesem Kontext sind neue Technologien wie das Hadoop Ecosystem entstanden. [Bit14] S. 12-14

Das Hadoop Ecosystem ist ein skalierbares Servercluster, welches durch verteilte Anfragetechniken, die parallele Verarbeitung und Analyse großer Datenmassen ermöglicht. Zu analysierende Daten sollten strukturiert vorliegen. Folglich sollten die semi- und unstrukturierten Datenmassen für die Analyse strukturiert werden.

Im Rahmen der Projektgruppe „Data Analytics with Hadoop“ (DASH) der Abteilung "Very Large Business Applications" (VLBA) der Carl von Ossietzky Universität Oldenburg, werden die Einsatzmöglichkeiten des Hadoop Ecosystems für die Analyse von XML-Dateien analysiert.

Detaillierter soll das Ziel sein, den Einsatz des Hadoop Ecosystems als Datenspeicher, zur Datenverarbeitung und Analyse von semistrukturierte Dateien zu analysieren. Die XML-Dateien – Log-Dateien von Terminalstationen - werden vom Unternehmenspartner CEWE Stiftung & Co. KGaA bereitgestellt. Zusätzlich soll versucht werden erste nutzbare Erkenntnisse aus diesen XML-Dateien zu gewinnen.

Das Vorgehen der Projektgruppe ist das Prototyping. Für die schnelle Einarbeitung und Evaluierung von diversen Tools des Hadoop Ecosystems wurde von den Projektmitgliedern eine Hadoop Sandbox genutzt. Für den Import der XML-Dateien in das Hadoop System wurden Tools wie bspw. Apache Flume und Gobblin evaluiert. Nachdem die Dateien in das Hadoop System importiert worden sind, sollen diese für die anschließende Analyse strukturiert werden. Für diese Aufgabe wurden Tools wie bspw. Spark und Hadoop Streaming evaluiert.

Herausforderung sind die vielen kleinen XML-Dateien, da Hadoop für das Speichern und Verarbeiten großer Dateien ausgelegt ist [Sei13]. Die Import-Tools sind standardmäßig auf das zeilenweise Auslesen und Verarbeiten Dateien ausgerichtet. Im Kontext von XML-Dateien sollte ein vollständiges Lesen und Verarbeiten einer Datei möglich gemacht werden. Dies kann aktuell mit keiner Standardfunktionalität der Import-Tools abgedeckt werden. Die Tools bieten Möglichkeiten selbstentwickelte Plugins einzubinden. Ein solches wird derzeit entwickelt.

Im Bereich der Datenstrukturierung liegt die Herausforderung darin, gültige XML-Dateien zum Parsen aus dem Hadoop System zu lesen. Eine weitere Schwierigkeit besteht darin, die gültigen XML-Dateien mit der XML-Parser Logik zusammenzubringen, um die Daten zu strukturieren.

Ausblick: Derzeit wird ein Hadoop System aufgesetzt. Die in der Sandbox evaluierten Lösungen sollen im Hadoop System unter möglichst realen Bedingungen getestet werden. Weiterhin sollen Tools in den Bereichen Visualisierung, Datenanalyse und Data Mining aus dem Hadoop Ecosystem evaluiert werden.

[Bit14] BITKOM: Big-Data-Technologien – Wissen für Entscheider: Leitfaden 2014

[Sei13] https://blog.codecentric.de/2013/08/einfuhrung-in-hadoop-die-wichtigsten-komponenten-von-hadoop-teil-3-von-5/ zugegriffen am 22.02.2015

B. Seminararbeiten

Im Rahmen der Projektgruppe wurde zu Beginn durch alle Projektteilnehmer eine Seminararbeit verfasst. Durch die Bearbeitung der verschiedenen Themen sollten die Studierenden einen Einblick in die kommende Aufgabe bekommen. Folgend werden alle Seminararbeiten der Projektdokumentation angehängt.

KDD Process

Seminar paper within the project group DASH

Professor:	Prof. Dr.-Ing. habil. Jorge Marx Gómez
Supervisors:	M. Eng. & Tech. Viktor Dmitriyev Dr.-Ing. Andreas Solsbach
Submitted by:	Sadok Ben Yahya sadok.ben.yahya@uni-oldenburg.de
Submission date:	02.12.2015

Contents

Abbreviations

KDDM	Knowledge Discovery & Data Mining
KDD	Knowledge Discovery in Databases
SE	Software Engineering
KD	Knowledge Discovery
DM	Data Mining
CRISP-DM	CRoss-Industry Standard Process for Data Mining
DASH	Data AnalyticS with Hadoop
CRM	Customer Relationship Management
ISI	Intelligence and Security Informatics
SEMMA	Sample, Explore, Modify, Model, Assess
SIG	Special Interest Group

Table Of Figures

1 Motivation

Managing large datasets can be problematic. Compared to smaller amounts of data, analysis, storage, privacy and interpretation can cause difficulties for today's data leaders. The more data an organization has, the more complex the problems of managing it can become.

The main motivation for this paper is a lack of a comprehensive overview of the existing KDDM models. Furthermore the cooperation with CEWE color as customer led to the existence of the project group Data AnalyticS with Hadoop (DASH) which will try to solve the data scalability issue that face CEWE in dealing with big stream of data each end of the year. Furthermore the project group DASH is to analyse productive data with Apache Hadoop.

2 Introduction

> *"[...] Knowledge Discovery is the most desirable end-product of computing. Finding new phenomena or enhancing our knowledge about them has a greater long-range value than optimizing production processes or inventories, and is second only to task that preserve our world and our environment. It is not surprising that it is also one of the most difficult computing challenges to do well [...]"* [wie96].

The number of KDDM projects has increased enormously over the past few years [MS09]. Due to the rising complexity of these projects, a number of problems emerged such as continues project planning delays, failure to meet user expectations and low productivity [MS09]. The failure rate of KDDM projects varies from around 60% (Gartner,2008) to 70% [Tan08]. This situation is mainly caused by the fact that there is no formal methods and methodologies, support tools or proper development project management [MMMS07].

Which brings us to the 'software crisis' in the late 1960s. The resolution of this crisis saw the emerging of Software Engineering (SE). In the process the software community borrowed ideas from other fields of engineering. Software development improved considerably as process models and methodologies for developing software projects saw the light [MMMS07]. The actual situation of DM and KD projects is not much different from the previous cited example.

The term KDD appeared first time in latest 1980s by Gregory Shapiro [FSM92]. Furthermore Knowledge Discovery (KD) is a creative process which require a number of different skills and knowledge. Besides it is important to understand the overall approach, before one decides to start a KDDM project and contrarily to software engineering there is no standard process to carry out KDDM projects [Cio07]. This means that a Knowledge Discovery & Data Mining project succeeded or failed is dependent on the particular person or team carrying it out and successful practice can not necessarily be repeated across the enterprise [Wir00]. KDDM projects needs a standard approach which will help understand the business problems, the appropriate data that fits the business objectives and in the same time fits the Data Mining (DM) techniques and finale allow one effectively evaluate the result and use it or document it [Wir00]. Therefore this paper focuses on explaining

the different aspects of the KDD process and comparing two of many methodologies as an example [Cio07].

For this purpose, an introduction and definition of KD and the must-know terminology impose itself. In addition an introduction to the KDD and his different domain application is presented. Then this paper explain in detail two KDD process models, which are considered as the pillars of the KDD discipline. To summarize, the contribution of this paper to the project group DASH is discussed.

3 What is KDD?

In this paper a lot of terms are used. Trying to avoid any kind of common confusion and to have a stable bases for further reading, most of the important terms are first explained, with references to definitions published in scientific literature, in the following section.

3.1 Definitions and Terminologies

Data Mining, Knowledge Discovery, Knowledge Discovery & Data Mining are terms used generally to refer to the research results, techniques and tools used in the efficient extraction of unknown *patterns* from a large amount of *data* [MMF10]. The complete process of discovering useful *knowledge* from *data* is known as KDD and DM is a particular step in this process.

Nevertheless the term Data Mining is used in general by many researchers as a synonym of the KDD process [CHS+98, PS00, CCK+00, KM06, MMMS07, MS09], specially in the news world where it refers to the whole KDD process [PS00]. From the beginning, the term Data Mining, which refers to the concept of finding useful patterns in data, was known under many names including information harvesting, information discovery, data pattern processing and knowledge extraction [FPsS96].

Knowledge Discovery in Databases defines the non-trivial process of identifying valid, novel, potentially useful and ultimately understandable pattern in data[FPsS96]. This term appeared at the first KDD Workshop in 1989 [FSM92].

Here *Data* are a set of facts, and *pattern* is an expression in some language describing a subset of the data or a model applicable to the subset. Extracting a pattern means, in general, making any high-level description of a set of data. The discovered pattern should be valid on new data with some degree of certainty and novel and potentially useful. Finally, the patterns should be understandable, if not immediately then after some post-processing [MMF10].

By *nontrivial* it is meant that some search or inference is involved [FPsS96]. Another way in understanding the concept of non-triviality is that a system, when performing a discovery process, must have some degree of autonomy in processing the data and evaluating its results [FSM92].

KDD is a *process* means that it consist of a number of steps that are required for his proper achievement. These steps comprise learning the application domain, creating a target data set, data cleaning and preprocessing and a few that are explained in further section.

To be used, the discovered pattern should be valid on new data with some degree of predefined certainty, for example, estimated prediction accuracy on new data [FPsS96]. An important notion called interestingness, which is a concept that allow the assessment of the knowledge generated during a KDD process, provides an overall measure of pattern values englobing simplicity, novelty and usefulness and simplicity [PsM94]. A pattern that is interesting and certain enough (both conditions are according to the user's criteria) is called knowledge [FSM92].

3.2 KDD domain applications

KDD technologies have made a huge impact on every aspect of our lives whether we realize it or not. Many new methods and techniques are developed in response to the demand for knowledge discovery applications in various domains. In this section a few examples of the domains in which KDD have made significant impacts are reviewed.

3.2.1 Business and Marketing

Business is a well-suited domain for Knowledge Discovery & Data Mining. Around the world, companies are gathering large volumes of data about their customers, sales, transactions, goods transportation and delivery, and customers reviews of products and services. KDD technologies have been playing a critical role in supporting various business functions such Customer Relationship Management (CRM), customer profiling, marketing, supply chain management, inventory control, demand forecasting, and product and service recommendation [GS06].

3.2.2 Finance

KDD have long been used in the financial industry to support decision making in various applications including fraud detection, stock market forecasting, corporate distress and bankruptcy prediction, portfolio management, and financial crime (e.g., money laundering) investigation [ZZ04].

3.2.3 Healthcare and Biomedicine

Due to the "information overload"that poses challenges to the healthcare professionals and medical researchers, it is nearly impossible for anyone to analyze, process, and interpret such large volumes of healthcare professionals to make in-time, correct decisions using manual approaches. KDD came as a solution to these challenges and become increasingly popular in the domain of heathcare and biomedicine because of the availability of large-size databases and data warehouses for clinical records, gene sequences, and medical literature. Data mining technologies enable healthcare professionals and researchers to explore healthcare data for the purpose of diagnosing diseases, developing effective treatment, drugs, nurse care plans, and novel hypotheses [KH05], and detecting infectious disease outbreaks [ZCL+05].

3.2.4 Security and intelligence

For helping intelligence communities and law enforcement agencies combat terrorism and other crimes around the world, a new interdisciplinary field called Intelligence and Security Informatics (ISI) has emerged [CMZ+03]. KDD is a core component in the ISI technology collection and its main goal is to exploit technologies and knowledge from different disciplines to assist crime investigation and help detect and prevent terrorist attacks [CMZ+03].

4 KDDM process models and methodologies

In this section a review of KDDM process models and methodologies is provided. The figure 4.1 shows the evolution of 14 KDD process models and methodologies. It is clear (from the figure 4.1) that KDD is the initial approach and CRISP-DM is the central approach. Furthermore it is shown that most of the approaches are based on them. That is why, for the purpose of the DASH project, this paper is concentrating only on this two approaches, mainly the KDD process model [FPsS96] as the scientific approach and the CRISP-DM [CCK+00] process model as the industrial approach.

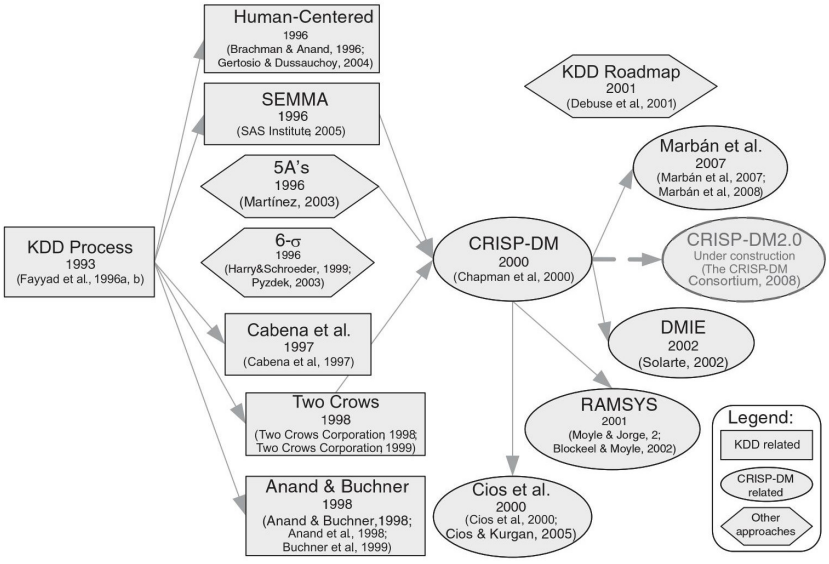


Figure 4.1: Evolution of KDDM process models and methodologies[1]

The two main process models, KDD and CRISP-DM, are described in this chapter in depth and their contributions and disadvantages are shown.

[0]Source : [MMF10]

4.1 KDD process model

As introduced before, KDD refers to the overall process of discovering useful Knowledge from data. This process is iterative. It includes the evaluation and possible interpretation of the patterns, after series of preprocessing, sampling and mining of the data with the primary goal of making a decision based on the result of what qualifies as knowledge. The KDD process is detailed in figure 4.2 from the data viewpoint.

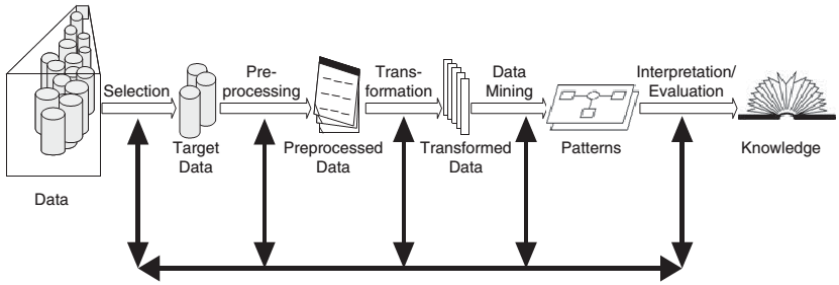


Figure 4.2: Overview of the steps constituting the Knowledge Discovery in Databases process[2]

[2] The Fayad et al. nine-step KDD process model is also interactive and detailed from practical viewpoint as follows :

1. *Developing and understanding the application domain.* It includes learning the relevant prior knowledge and the goals of the end user of the discovered knowledge.

2. *Creating a target data set.* It includes selecting a subset of variables (attributes) and data samples (examples) that will be used to perform discovery tasks. This step usually includes querying the existing data to select the desired subset.

3. *Data cleaning and preprocessing.* This step consists of removing outliers, dealing with noise and missing values in the data, and accounting for time sequence information and known changes, as well as deciding data base management system issues, such as data types, schema and mapping of missing and unknown values.

4. *Data reduction and projection.* This step consists of finding useful attributes by applying dimension reduction and transformation methods, and finding invariant representation of the data.

[2]Source: [FPsSW96]

5. *Choosing the data mining task.* It includes matching the goals defined in Step 1 with a particular DM method, such as classification, regression, clustering, etc.

6. *Choosing the data mining algorithm.* The data miner selects methods to search for patterns in the data and decides which models and parameters of the methods used may be appropriate.

7. *Data mining.* This step generates patterns in a particular representational form, such as classification rules, decision trees, regression models, trends, etc.

8. *Interpreting mined patterns.* Here the analyst interprets the discovered patterns and, when needed, returning to any of the previous steps. He performs also visualization of the extracted patterns and models, and visualization of the data based on the extracted models, as well as removing redundant or irrelevant patterns and translating the useful ones into terms understandable by users.

9. *Consolidating discovered knowledge.* The final step consists of incorporating the discovered knowledge into the performance system, and documenting and reporting it to the interested parties. This step may also include checking and resolving potential conflicts with previously believed knowledge.

KDD process from Fayad et al. explains precisely what to do in a KDDM project but lacks the documentation don't explain how to do every phase. It do not treat the project management and quality aspect.

4.2 CRISP-DM process model

4.2.1 Overview

CRISP-DM is build on previous experiences in the interdisciplinary field of KDD and previous attempts to define KDD processes. Figure 4.1 shows clearly that, with the KDD process from Fayyad et al. [FPsS96], CRISP-DM is a corner stone in the KDD methodologies. This section gives an introduction of the CRISP-DM process model and then dive a little bit in the different phases of the CRISP-DM process model. More detailed information can be found in the CRISP-DM guide [CCK+00].

CRISP-DM is the most used methodology for developing KDDM projects [KdN02, KdN04, KdN14]. Nevertheless CRISP-DM is not a lone runner due to rivalry

with other, in-house methodologies like Sample, Explore, Modify, Model, Assess (SEMMA) (see Figure 4.3).

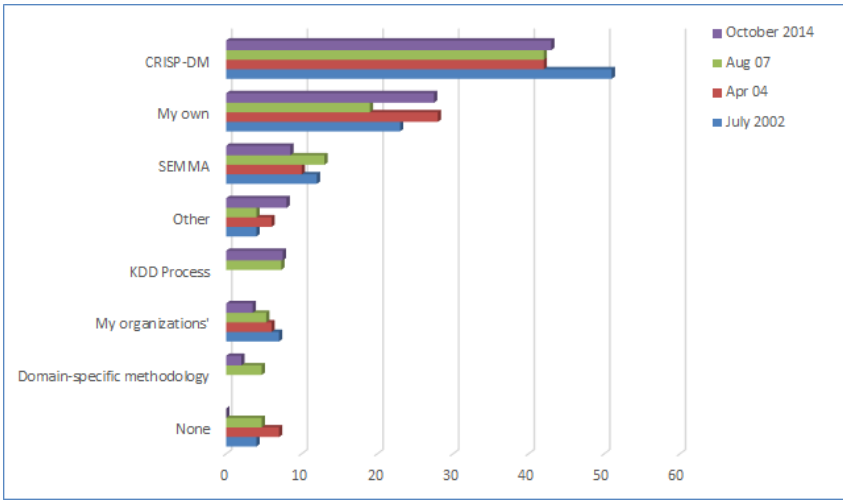


Figure 4.3: The most used KDD methodologies[3]

The inconvenience is that CRISP-DM does not include project management activities such as quality management or change management [MMMS07].On the other hand, the slight increase of the use of SEMMA is due to its data mining support tool Enterprise Miner,developed by SAS and based in SEMMA methodology, which increased in use.

CRoss-Industry Standard Process for Data Mining was first proposed in early 1996 by a consortium of four companies: SPSS (a provider of commercial DM solutions), NCR (a database provider), Daimler Chrysler, and OHRA (an insurance company). The last two companies served as sources of data and case studies. The model was officially released (version 1.0) in 2000 [CCK+00]. The CRISP-DM Special Interest Group (SIG) was created with the goal of supporting the model. One important factor of the success of CRISP-DM is the fact that it is industry-, tool- and application-neutral.

The CRISP-DM methodology is described as a hierarchical process model, englobing four levels of abstraction (from general to specific): phases, general tasks, specialized tasks and process instances. [3]

At the top level, CRISP-DM is organized into a small number of phases. Each phase consists of several second-level generic tasks. This second level is called generic, because it is meant to be general enough to cover all possible data mining situations. The generic tasks are designed to be as complete and stable as possible.

The third level, there is the specialized task level which meant to describe how

[3]Source: [KdN02, KdN04, KdN14]

actions in the generic tasks should be performed in specific situations. In practice, many of the tasks can be performed in a different order and it will often be necessary to backtrack to previous tasks and repeat certain actions.

The fourth level, the process instance level, is a record of actions, decisions, and results of an actual data mining engagement.

Horizontally, the CRISP-DM methodology distinguishes between the reference model and the user guide. The reference model presents a quick overview of phases, tasks, and their outputs and describes what to do in a KDDM project. The user guide gives more detailed tips and hints for each phase and each task within a phase and depicts how to do a KDDM project [Wir00].

4.2.2 The CRISP-DM reference model

The life cycle of a data mining project according to CRISP-DM consists of six phases. The sequence of the phases is not strict. Moving back and forth between different phases is always required. It depends on the outcome of each phase, which phase or which particular task of a phase, that has to be performed next.

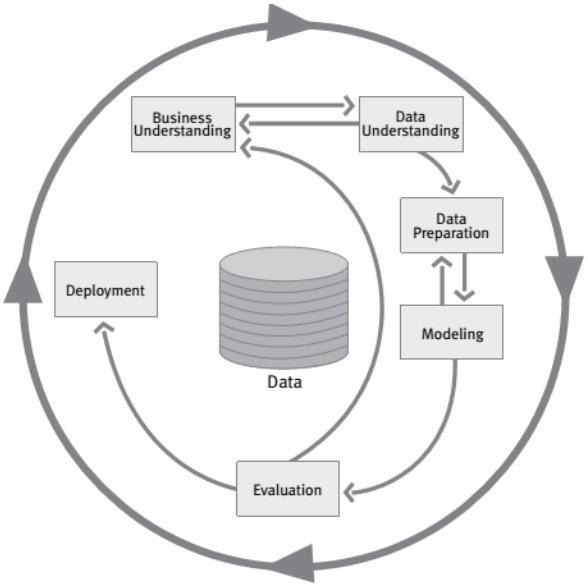


Figure 4.4: Phases of the CRISP-DM reference model [4]

Each phase is outlined as follows [CCK+00] :

1. *Business understanding.* This step focuses on the understanding of objectives and requirements from a business perspective. It also converts these into a DM problem definition, and designs a preliminary project plan to achieve the objectives. It is further broken into several substeps, namely:

[4]Source: [CCK+00]

a) determination of business objectives

b) assessment of the situation

c) determination of DM goals

d) generation of a project plan

2. *Data understanding.* This step starts with initial data collection and familiarization with the data. Specific aims include identification of data quality problems, initial insights into the data, and detection of interesting data subsets. Data understanding is further broken down into:

   a) collection of initial data

   b) description of data

   c) exploration of data

   d) verification of data quality

3. *Data preparation.* This step covers all activities needed to construct the final dataset, which constitutes the data that will be fed into DM tool(s) in the next step. It includes Table, record, and attribute selection; data cleaning; construction of new attributes; and transformation of data. It is divided into :

   a) selection of data

   b) cleansing of data

   c) construction of data

   d) integration of data

   e) formatting of data substeps

4. *Modeling.* At this point, various modeling techniques are selected and applied. Modeling usually involves the use of several methods for the same DM problem type and the calibration of their parameters to optimal values. Since some methods may require a specific format for input data, often reiteration into the previous step is necessary. This step is subdivided into:

   a) selection of modeling technique(s)

   b) generation of test design

   c) creation of models

   d) assessment of generated models

5. *Evaluation.* After one or more models have been built that have high quality from a data analysis perspective, the model is evaluated from a business objective perspective. A review of the steps executed to construct the model is also performed. A key objective is to determine whether any important business issues have not been sufficiently considered. At the end of this phase, a decision about the use of the DM results should be reached. The key substeps in this step include:
   a) evaluation of the results
   b) process review
   c) determination of the next step
6. *Deployment.* Now the discovered knowledge must be organized and presented in a way that the customer can use. Depending on the requirements, this step can be as simple as generating a report or as complex as implementing a repeatable KDD process. This step is further divided into:
   a) plan deployment
   b) plan monitoring and maintenance
   c) generation of final report
   d) review of the process substeps

5 Conclusion

Currently, the most widely used data mining model is CRISP-DM [CCK+00], considered as the de facto standard. CRISP-DM does not cover many tasks related to project management, organization and quality in the way required by the increasing complexity of the more recent data mining and knowledge discovery projects. These projects not only involve examining huge volumes of data, but also managing and organizing big interdisciplinary human teams. In the case of the project group DASH, a revised version of CRISP-DM is needed as many of the generic and specialized tasks apply only in real industrial project with big interdisciplinary teams.

Bibliography

[CCK+00] CHAPMAN, Pete ; CLINTON, Julian ; KERBER, Randy ; KHABAZA, Thomas ; REINARTZ, Thomas ; SHEARER, Colin ; WIRTH, Rudiger: CRISP-DM 1.0 Step-by-step data mining guide / The CRISP-DM consortium. Version: August 2000. `https://www.the-modeling-agency.com/crisp-dm.pdf`. 2000. – Forschungsbericht

[CHS+98] CABENA, Peter ; HADJINIAN, Pablo ; STADLER, Rolf ; VERHEES, Jaap ; ZANASI, Alessandro: *Discovering Data Mining: From Concept to Implementation*. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 1998. – ISBN 0–13–743980–6

[Cio07] *Kapitel* The Knowledge Discovery Process. In: CIOS, Krzysztof: *Data mining a knowledge discovery approach*. New York New York : Springer, 2007. – ISBN 978–0–387–36795–8

[CMZ+03] CHEN, Hsinchun (Hrsg.) ; MIRANDA, Richard (Hrsg.) ; ZENG, Daniel D. (Hrsg.) ; DEMCHAK, Chris C. (Hrsg.) ; SCHROEDER, Jennifer (Hrsg.) ; MADHUSUDAN, Therani (Hrsg.): *Intelligence and Security Informatics, First NSF/NIJ Symposium, ISI 2003, Tucson, AZ, USA, June 2-3, 2003, Proceedings*. Bd. *2665*. Springer, 2003 (Lecture Notes in Computer Science). – ISBN 3–540–40189–X

[FPsS96] FAYYAD, Usama ; PIATETSKY-SHAPIRO, Gregory ; SMYTH, Padhraic: From Data Mining to Knowledge Discovery in Databases. In: *AI Magazine* 17 (1996), S. 37–54

[FPsSW96] FAYYAD, Usama ; PIATETSKY-SHAPIRO, Gregory ; SMYTH, Padhraic ; WIDENER, Terry: The KDD Process for Extracting Useful Knowledge from Volumes of Data. In: *Communications of the ACM* 39 (1996), S. 27–34

[FSM92] FRAWLEY, W. J. ; SHAPIRO, Piatetsky G. ; MATHEUS, C. J.: Knowledge discovery in databases - an overview. In: *Ai Magazine* 13 (1992), 57–70. `http://citeseer.ist.psu.edu/frawley92knowledge.html`

[GS06] GHANI, Rayid ; SOARES, Carlos: Data Mining for Business Applications. In: *Proceedings of the KDD Workshop on Data Mining for Business Applications*. New York, NY, USA, 2006

[KdN02] KDNUGGETS.COM: *What main methodology are you using for data mining?* http://www.kdnuggets.com/polls/2002/methodology.htm, July 2002

[KdN04] KDNUGGETS.COM: *Data Mining Methodology*. http://www.kdnuggets.com/polls/2004/data_mining_methodology.htm, April 2004

[KdN14] KDNUGGETS.COM: *What main methodology are you using for your analytics, data mining, or data science projects? Poll*. http://www.kdnuggets.com/polls/2014/analytics-data-mining-data-science-methodology.html, October 2014

[KH05] KOH H.C., Tan. G.: Data mining applications in healthcare. In: *Journal of healthcare information management : JHIM* Bd. 19, 2005, S. 64–72

[KM06] KURGAN, Lukasz A. ; MUSILEK, Petr: A Survey of Knowledge Discovery and Data Mining Process Models. In: *Knowl. Eng. Rev.* 21 (2006), März, Nr. 1, 1–24. http://dx.doi.org/10.1017/S0269888906000737. – DOI 10.1017/S0269888906000737. – ISSN 0269–8889

[MMF10] MARISCAL, Gonzalo ; MARBÁN, Óscar ; FERNÁNDEZ, Covadonga: A survey of data mining and knowledge discovery process models and methodologies. In: *The Knowledge Engineering Review* 25 (2010), 6, 137–166. http://dx.doi.org/10.1017/S0269888910000032. – DOI 10.1017/S0269888910000032. – ISSN 1469–8005

[MMMS07] MARBÁN, Óscar ; MARISCAL, Gonzalo ; MENASALVAS, Ernestina ; SEGOVIA, Javier: An Engineering Approach to Data Mining Projects. In: *Proceedings of the 8th International Conference on Intelligent Data Engineering and Automated Learning*. Berlin, Heidelberg : Springer-Verlag, 2007 (IDEAL'07). – ISBN 3–540–77225–1, 978–3–540–77225–5, 578–588

[MS09] *Kapitel* A Data Mining & Knowledge Discovery Process Model. In: MARBÁN, Gonzalo M. ; SEGOVIA, Javier: *Data Mining and Knowledge Discovery in Real Life Applications*. I-Tech Education and Publishing, 2009, 17

[PS00] PIATETSKY-SHAPIRO, Gregory: Knowledge Discovery in Databases: 10 Years After. In: *SIGKDD Explor. Newsl.* 1 (2000), Januar, Nr. 2, 59–61. http://dx.doi.org/10.1145/846183.846197. – DOI 10.1145/846183.846197. – ISSN 1931–0145

[PsM94] PIATETSKY-SHAPIRO, Gregory ; MATHEUS, Christopher J.: The interestingness of deviations. In: *Proceedings of the AAAI-94 Workshop on Knowledge Discovery in Databases, Seattle, Washington, pp. 25–36.*, 1994

[Tan08] TANIAR, David: *Data mining and knowledge discovery technologies*. Hershey : IGI Pub, 2008. – ISBN 9781599049601

[wie96] On the Barriers and Future of Knowledge Discovery. In: FAYYAD, Usama M. (Hrsg.) ; PIATETSKY-SHAPIRO, Gregory (Hrsg.) ; UTHURUSAMY, Ramasamy (Hrsg.): *Advances in Knowledge Discovery and Data Mining* -. New. AAAI Press, 1996. – ISBN 978–0–262–56097–9

[Wir00] WIRTH, Rüdiger: CRISP-DM: Towards a standard process model for data mining. In: *Proceedings of the Fourth International Conference on the Practical Application of Knowledge Discovery and Data Mining*, 2000, S. 29–39

[ZCL+05] ZENG, Daniel ; CHEN, Hsinchun ; LYNCH, Cecil ; EIDSON, Millicent ; GOTHAM, Ivan: Infectious Diseaxe Informatics and Outbreak detection. Version: 2005. http://dx.doi.org/10.1007/0-387-25739-X_13. In: CHEN, Hsinchun (Hrsg.) ; FULLER, SherrilynneS. (Hrsg.) ; FRIEDMAN, Carol (Hrsg.) ; HERSH, William (Hrsg.): *Medical Informatics* Bd. 8. Springer US, 2005. – ISBN 978–0–387–24381–8, 359-395

[ZZ04] ZHANG, Dongsong ; ZHOU, Lina: Discovering golden nuggets: data mining in financial application. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C* 34 (2004), Nr. 4, 513-522. http://dblp.uni-trier.de/db/journals/tsmc/tsmcc34.html#ZhangZ04a

Abschließende Erklärung

Ich versichere hiermit, dass ich meine Seminararbeit (Titel der Arbeit)... selbständig und ohne fremde Hilfe angefertigt habe, und dass ich alle von anderen Autoren wörtlich übernommenen Stellen wie auch die sich an die Gedankengänge anderer Autoren eng anlegenden Ausführungen meiner Arbeit besonders gekennzeichnet und die Quellen zitiert habe.

Oldenburg, den September 26, 2016 Sadok Ben Yahya

Projektgruppe Apache Hadoop

BAND 2

Projektdokumentation

Projektgruppe Apache Hadoop

Projektdokumentation

Themenersteller:	Prof. Dr.-Ing. habil. Jorge Marx Gómez
Betreuer:	M. Eng. & Tech. Viktor Dmitriyev Dr.-Ing. Andreas Solsbach
Abgabetermin:	30. September 2016

Inhaltsverzeichnis

Data Mining Methoden

Seminararbeit

Themenersteller:	Prof. Dr.-Ing. habil. Jorge Marx Gómez
Betreuer:	M. Eng. & Tech. Viktor Dmitriyev Dr.-Ing. Andreas Solsbach M.Sc. Ola Mustafa
Vorgelegt von:	Simon Becker simon.becker1@uni-oldenburg.de
Abgabetermin:	2. Dezember 2015

Inhaltsverzeichnis

Abkürzungen

CLARANS Clustering Large Applications based on RANdomized Search

CRISP-DM Cross Industry Standard Process for Data Mining

KDD Knowledge Discovery in Databases

kNN k-Nearest Neighbour

PAM Partitioning Around Medoids

Abbildungen

Tabellen

1 Einleitung

Unternehmen und Institutionen sammeln immer mehr Daten. Dies kann freiwillig geschehen, um weitere Information zu sammeln, oder aufgrund rechtlicher Bestimmungen. Banken beispielsweise speichern Daten über die Transaktionen ihrer Kunden, Wetterdienste sammeln verschiedenste Messwerte an vielen Orten und Händler untersuchen das Kaufverhalten ihrer Kunden. Somit verdoppelt sich die Menge an verfügbaren Daten immer schneller. (Vgl. [LC14, S.1])

Eine Studie der EMC Corporation prognostiziert einen Anstieg des Volumens der jährlich weltweit generierten digitalen Datenmengen von 8.591 Exabyte in 2015 auf über 40.000 Exabyte in 2020 ([Cor12]).

Die Menge an Daten ist in vielen Unternehmen so groß, dass das Treffen von Entscheidungen durch einfaches anschauen und interpretieren der Daten aus dem Bauch heraus nicht sinnvoll sein kann. Auch die klassische Statistik stößt mit ihren Mitteln an Grenzen. An dieser Stelle bietet das Data Mining Verfahren zur Auswertung der Daten und Generierung von Wissen an. Gegenstand des Data Minings ist es, nach bisher unbekannten Mustern in Daten zu suchen und neue Zusammenhänge zu identifizieren. Das Ziel dabei ist es, nützliches Wissen zu generieren. Dieses neue Wissen kann genutzt werden, um Prozesse zu optimieren, Marketingkampagnen Zielgruppengerechter zu steuern oder Kundenverhalten vorauszusagen, um so einen Wettbewerbsvorteil gegenüber Mitbewerbern zu erlangen. (Vgl. [LC14, S.1-2] und [Run10, S.2-3])

In dieser Ausarbeitung sollen Methoden des Data Mining vorgestellt werden. Dabei sollen die vorgestellten Methoden mit realen Anwendungsfällen des Fotoentwicklungsunternehmen CEWE verknüpft werden. CEWE speichert Informationen über die Nutzung ihrer Digi-Foto-Maker, ihren Fotoautomaten, an denen die Kunden selber Bilder drucken und Artikel bestellen können.

Mögliche Anwendungsfälle sind:

Beispiel 1.0.1: *Warenkorbanalyse*
Durch die Analyse des Kaufverhaltens der Kunden soll herausgefunden werden, welche Produkte gerne zusammen gekauft werden. Durch optimierte Werbung soll der Absatz dieser Produktkombination gesteigert werden.

Beispiel 1.0.2: *Clickstream Analyse*

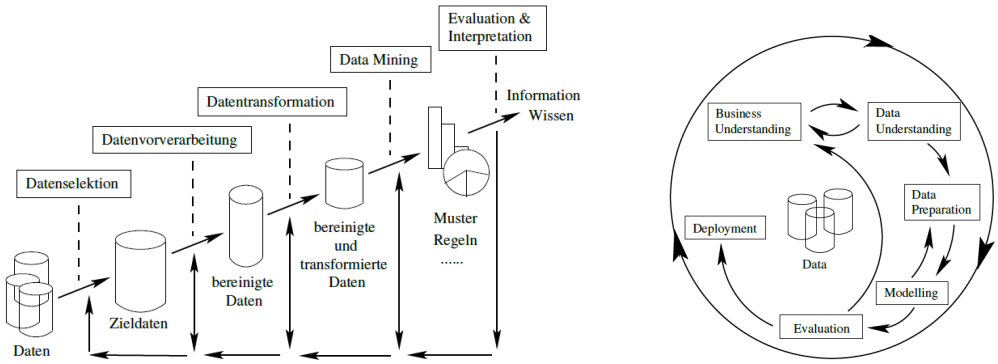
Durch Erforschung des Verhalten der Kunden am Automaten durch Auswertung der gespeicherten Clickstream Daten soll die Software verbessert werden.

Beispiel 1.0.3:*Predictive Analytics*
Anhand der Analyse von Sensordaten der Automaten sollen Ausfälle und Störungen vorausgesagt werden, um so präventiv eingreifen, oder schneller auf Fehler reagieren zu können.

Beispiel 1.0.4:*Klassifikation von Kunden*
Die Kunden von CEWE können in verschiedene Gruppen eingeordnet werden. Beispielsweise in technisch affine Kunden, die die Digi Foto Maker intuitiv und sicher bedienen und in technisch unerfahrenen Kunden, die Schwierigkeiten haben, sich am Automaten zurechtzufinden. Mit den Informationen, wann welche Zielgruppe welche Automaten benutzt, kann die Werbung für bestimmte Produkte Zielgruppengerecht gesteuert werden oder an bestimmten Stellen Hilfe angeboten werden.

1.1 Data Mining Prozess

Um Data Mining Verfahren einzusetzen gibt es verschiedene Vorgehensmodelle. Die beiden bekanntesten werden in Abbildung 1.1a bzw. 1.1b dargestellt und in diesem Abschnitt beschrieben.



(a) Data Mining Prozess ([LC14], S.5 und [Fay96], S.10) (b) CRISP Modell ([LC14], S.5)

Abbildung 1.1: Vergleich des Data Mining Prozesses nach Fayyad mit dem CRISP Modell

Das Verfahren auf der linken Seite der Abbildung wird als Knowledge Discovery in Databases (KDD) bezeichnet und wurde von Fayyad entwickelt und besteht aus folgenden Phasen:

Selektion In Unternehmen liegen Daten an vielen verschiedenen Orten in unterschiedlichster Form vor. Aus diesen Unmengen an Daten müssen zunächst

die interessanten Daten ausgewählt und geeignete Merkmale zur Analyse bestimmt werden. Nach der Durchführung der Datensammlung steht ein Satz an Rohdaten zur Verfügung.

Datenvorverarbeitung Die Rohdaten beinhalten oftmals nicht konsistente oder fehlerhafte Informationen. In der Datenvorverarbeitung werden die Daten bereinigt. Es werden Fehler und widersprüchliche Werte korrigiert und fehlende Informationen ergänzt.

Transformation Während der Transformation werden die Daten in adäquate Datenformate gebracht, die die entsprechenden Auswertungen erfordern. Beispielsweise kann es erforderlich sein metrische Daten in Intervalle einzusortieren.

Data Mining In diesem Abschnitt geschieht das eigentliche Data Mining. Dabei wird ein Modell entwickelt, wie beispielsweise die Erstellung eines Entscheidungsbaumes. Anschließend werden die Parameter für das Modell gewählt und das Ergebnis getestet. Dieses Vorgehen kann mit angepassten Parametern mehrmals wiederholt werden.

Interpretation und Evaluation In der anschließenden Phase werden die Ergebnisse aus dem Data Mining interpretiert. Dabei wird geprüft, ob aus der Analyse neue Erkenntnisse generiert werden konnten.

In der Abbildung ist erkennbar, dass jederzeit in eine vorherige Phase zurückgesprungen werden kann, um das Data Mining Verfahren zu optimieren. Auf der rechten Seite der Abbildung 1.1b ist das CRISP Modell visualisiert. CRISP-DM steht für Cross Industry Standard Process for Data Mining. Das Ziel dieses Vorgehens ist es, einen Data Mining Prozess zu definieren und das Modell in einem zyklischen verfahren zu validieren und zu optimieren. Im Unterschied zum KDD-Prozess enthält dieses Vorgehen drei zusätzliche Phasen. Vor der Sammlung der Daten fordert das CRISP Modell die Aufgabe, die gestellt wurde zu verstehen (Business Understanding). Dazu gehört auch das Verstehen der Unternehmensziele und die Erstellung eines Projektplans. Beim Verständnis der Daten (Data Understanding) geht es darum, die vorliegenden Daten zu beschreiben und zu untersuchen. Abschließend enthält das CRISP Modell zusätzlich die Phase Deployment. In dieser Phase wird der Einsatz des erforschten Modells im Unternehmen geplant und das Data Mining Projekt rückblickend reflektiert. (Vgl. [LC14], S.4-11)

1.2 Einordnung und Abgrenzung

Diese Ausarbeitung fokussiert sich auf das eigentliche Data Mining und dessen Methoden. Nachdem die Datenbestände gesammelt und aufbereitet wurden, können die Analysen des eigentlichen Data Minings durchgeführt werden. Dazu wird zunächst festgelegt, welche Data Mining Aufgabe (bspw. Klassifizierung oder Cluster-Bildung) vorliegt. Danach wird eine passende Data Mining Methode ausgewählt. Anschließend werden die Parameter der Methode entsprechend der Problemstellung konfiguriert. Mit dieser Parametrisierung wird schließlich nach interessanten Mustern in den Daten gesucht. Die ausgewählte Aufgabe und die Methode zusammen mit der Parametrisierung bilden ein Data Mining Modell. Dieses Modell wird evaluiert und bei Bedarf mehrmals mit anderen Parametern angepasst und erneut angewendet. (Vgl. [LC14], S.10)

Im folgenden werden nun zunächst in Kapitel 2 die Aufgaben des Data Minings vorgestellt und anschließend in Kapitel 3 die dazu passenden Methoden beschrieben. In diesen Kapiteln wird der Bezug zu dem oben aufgeführten Beispielen von CEWE hergestellt, um den Nutzen in der Praxis aufzuzeigen.

2 Anwendungsklassen des Data Minings

In diesem Kapitel werden zunächst die wichtigsten Aufgaben des Data Minings vorgestellt. Im folgenden Kapitel werden anschließend einige Methoden und Algorithmen zu den Aufgaben erläutert.

2.1 Klassifikation

Die Klassifikation ist eine der bekanntesten Anwendungen des Data Minings. Bei der Klassifikation geht es darum, Datensätze einer bestehenden Klasse zuzuordnen. In Beispiel 1.0.4 auf Seite 344 wurde bereits ein Beispiel für einen Anwendungsfall bei CEWE genannt. Dabei geht es darum, Kunden anhand ihrer Affinität zur Technik in unterschiedliche Gruppen einzuordnen. Auf Basis einer Trainingsdatenmenge, in der Datensätze mit bekannten Klassen vorhanden sind, wird ein Modell zur Klassifikation von Datensätzen mit unbekannten Klassen entwickelt. Das Modell wird an einer Menge an Testdaten evaluiert und falls notwendig angepasst. Für die Kontrolle der Richtigkeit des Modells muss die Klasse der Testdatensätze ebenfalls bekannt sein. Wenn die Fehlerrate des Modells einen akzeptablen Bereich erreicht hat, kann das Modell auf Datensätze mit unbekannten Klassen angewendet werden. Während der Lernphase gilt es zu erkennen, welche der gegebenen Attribute einen entscheidenden Einfluss auf die Klasseneinordnung haben. Das Ergebnis der Klassifikation muss im Übrigen keine eindeutige Klasse als Ergebnis aufweisen. Oft ist es interessant die Wahrscheinlichkeit für die Zuordnung zu der jeweiligen Klasse zu erhalten. Da für den Lernprozess sowohl die Klassen der Trainingsmenge, als auch der Testmenge bekannt sein muss, wird dieses Vorgehen als überwachtes Lernen bezeichnet. (Vgl. [LC14], S.59-61)

2.2 Clustering

Ähnlich wie das in Abschnitt 2.1 beschriebene Verfahren der Klassifikation fasst auch das Clustering Daten zu Gruppen zusammen. Im Gegensatz zur Klassifikation sind die Klassen jedoch zu Beginn unbekannt. Es handelt sich folglich um eine Art des unüberwachten Lernens. Das Clustering gruppiert Daten anhand ihrer Ähnlichkeit zu anderen Datensätzen in sogenannte Cluster. Jedes Datum ist

den Daten des eigenen Clusters ähnlicher als der Daten eines beliebigen anderen Clusters.

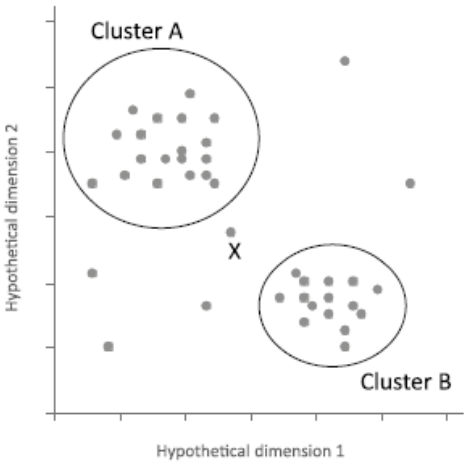


Abbildung 2.1: Darstellung von Daten und möglichen Clustern (s. [MJ14], S.89)

In Abbildung 2.1 sind Daten mit einer fiktiven zweidimensionalen Ausprägung in Form eines Streudiagramms dargestellt. Die Ähnlichkeit zwischen den Datensätzen wird durch den Abstand im Diagramm dargestellt. Die Kreise zeigen eine mögliche Einteilung der Daten in zwei verschiedene Cluster A und B. Dargestellt ist hier nur eine mögliche Einteilung in Gruppen. Bei dem angewendeten Modell wurde das mit einem X markierten Datum in keines der Cluster einsortiert. Bei einer anderen Parametrisierung könnte X jedoch in Cluster A oder in Cluster B aufgenommen werden. Das Clustering Verfahren ist eine sehr flexible Möglichkeit Daten zusammenzufassen. Durch das Clustering können aber auch sogenannte Ausreißer erkannt werden, also Datensätze, die keinerlei Ähnlichkeit zu den anderen Sätzen aufweisen. In der Abbildung sind mehrere dieser Ausreißer zu erkennen. (Vgl. [Mya07], S.110 - S.111)

2.3 Numerische Vorhersage

Das numerische Verfahren wird genutzt, um numerische Werte zu prognostizieren. Thematisch fällt dieses Vorgehen zusammen mit der Klassifikation in eine Kategorie. Der Unterschied besteht jedoch darin, dass die Klassifikation einen diskreten Wert aus einer endlichen Lösungsmenge vorhersagt, während die numerische Vorhersage Werte aus einem unendlichen numerischen Wertebereich vorhersagt. Der Grundgedanke hinter dem numerischen Verfahren nutzt die Approximation einer Funktion anhand von Beispielen. Als Ausgangsbasis ist nur eine Instanzenmenge E einer Funktion $y = f(x)$ bekannt. Dabei sind die Beispieldatensätze

$X = x_1, \ldots, x_n$ sowie deren Zielwerte $Y = y_1, \ldots, y_n$ bekannt. Auf Basis dieser Daten soll eine Funktion $f'(x) = h(x)$ ermittelt werden, die die Zusammenhänge zwischen den gegebenen Werten und ihren Zielwerten möglichst genau wiedergibt. Mit Hilfe dieser Funktion können dann auch bisher unbekannte Zielwerte prognostiziert werden. In der Praxis werden numerische Vorhersagen unter anderem bei der Prognose von Temperaturen, Verkaufszahlen oder Produktabsätzen verwendet. (Vgl. [LC14], S.61 - S.62)

2.4 Assoziationsanalyse

Die Assoziationsanalyse hat ihren Ursprung in der Warenkorbanalyse. Ziel einer solchen Analyse ist es, das Kaufverhalten anhand von Beziehungen zwischen den gekauften Waren zu analysieren. Dabei wird untersucht welche Produkte gemeinsam mit einem anderen Produkt gekauft wird. In Beispiel 1.0.1 auf Seite 343 wurde bereits ein Beispiel im Unternehmensumfeld von CEWE angeführt. Die Aussage aus diesem Verfahren wird wie folgt dokumentiert:

Wer Produkt A kauft, kauft häufig auch Produkt B. $A \rightarrow B$

Um den Verkauf von Produkt B zu steigern und somit den Gewinn zu erhöhen, kann Produkt B gezielt in Kombination mit Produkt A beworben werden. Die Assoziationsanalyse ist nun eine verallgemeinerte Form der Warenkorbanalyse. Der Zielbereich ist nicht mehr nur der Warenkorb eines Kunden, sondern der Versuch in einer Datenbank Bereiche zu finden, in denen mit hoher Wahrscheinlichkeit Werte gemeinsam auftreten. Für beide Verfahrensarten gilt, dass die Vorhersage nicht immer zu 100% zutreffen muss. Vielmehr wird zu jeder Regel eine Wahrscheinlichkeit ($supp$) ermittelt, zu der die Regel Erfolg hat. Zusätzlich wird die Konfidenz ($conf$) errechnet. Die Konfidenz beschreibt das Vertrauen in die Regel. (Vgl. [LC14], S.7 - S.9)

2.5 Text Mining

Im Gegensatz zu den bisher vorgestellten Data Mining Verfahren untersucht das Text Mining unstrukturierte Daten. Dies sind Bücher, Artikel, oder sonstige Texte, die beispielsweise auf ihren Inhalt oder ihre Güte hin untersucht und klassifiziert werden sollen. Ein mögliches Vorgehen des Text Minings ist die Extraktion von Schlüsselbegriffen. Dazu muss der Text zunächst aufbereitet werden. Die Füllwörter im Text werden eliminiert. Die übrigen Begriffe werden auf Abkürzungen und Synonyme untersucht und vereinheitlicht. Auf Basis des aufbereiteten Bestands können dann andere Data Mining Verfahren angewendet werden. Beispielsweise

ist auch die Häufigkeitsverteilung der Wörter zur Einordnung in einen bestimmten Fachbereich interessant. (Vgl. [LC14], S.65 - S.66)

2.6 Web Mining

Das Web Mining befasst sich mit der Analyse von Informationen rund um das Internet. In Abbildung 2.2 sind die einzelnen Themengebiete des Web Minings mit ihren Unterkategorien dargestellt. Grundsätzlich wird zwischen dem Web Con-

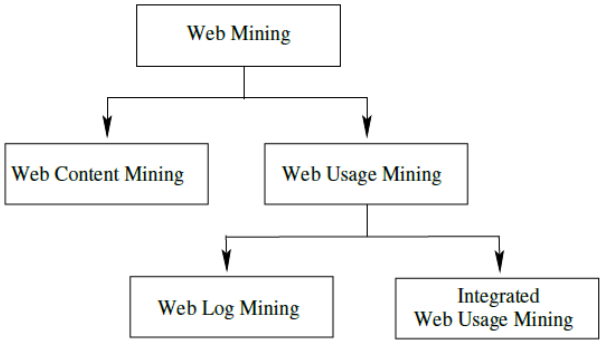


Abbildung 2.2: Themengebiete des Web Minings in der Übersicht (s. [LC14], S.67)

tent Mining und dem Web Usage Mining differenziert. Beim Web Content Mining werden die Inhalte des Internets untersucht. Dazu gehören einfache Texte, sowie multimediale Daten. Auch die Verlinkung zwischen den verschiedenen Webseiten wird analysiert. Das Web Usage Mining fokussiert sich dagegen auf die Analyse des Verhaltens der Nutzer im Internet. Dazu werden beispielsweise die Log Dateien der Webserver untersucht. Dieses Teilgebiet des Web Usage Mining wird als Web Log Mining bezeichnet. Der Begriff Integrated Web Usage Mining umfasst zusätzlich weitere Datenbestände neben den Log Dateien als Datenquelle.

Da Unternehmen ihre Geschäftsfelder immer mehr auf das Internet ausbreiten wird insbesondere das Web Log Mining weiter an Bedeutung gewinnen. Den Betreiber einer Webseite interessiert nicht nur, wer seine Seite besucht, sondern auch wer etwas kauft und vor allem an welchen Stellen die Seite verlassen wird. Aus den Analysen lassen sich Potentiale erkennen, um durch Optimierung der Webseite weitere Kunden anzulocken und mehr Kunden dazu zu bringen die angeschauten Produkte auch zu kaufen. (Vgl. [LC14], S.66 - S.67)

3 Data Mining Methoden

Nachdem in Kapitel 2 die Aufgaben des Data Minings vorgestellt wurden, werden in diesem Kapitel einzelne Verfahren und Methoden zur Anwendung der Aufgaben vorgestellt.

3.1 k Nearest Neighbour

Das k-Nearest Neighbours Verfahren (kNN) ist ein Data Mining Verfahren, dass zur Klassifikation genutzt wird. Das Verfahren ist ein vergleichsweise einfaches Verfahren um unbekannte Klassen von Objekten vorherzusagen. Im Gegensatz zu anderen Verfahren wird bei diesem Vorgehen kein mathematisches Modell entworfen, sondern die Vorhersage anhand der Berechnung von Ähnlichkeiten zu benachbarten Objekten ermittelt. Grundsätzlich werden die k ähnlichsten Objekte, der bereits bekannten Menge ermittelt und anhand deren Klassenzuordnung die neue Klasse des Objektes bestimmt. In einer Lernphase wird anhand einer Trainingsmenge die optimale Anzahl an zu betrachtenden Nachbarn k ermittelt. (Vgl. [Mya07], S.176 - S.178)

Ist k gefunden ist die Vorhersage der neuen Klasse trivial. Die Ähnlichkeit der beschreibenden Attribute eines neuen Objektes zu allen bereits bekannten Objekten wird ermittelt. Die k ähnlichsten Objekte - die k-nächsten Nachbarn - werden für die Vorhersage der neuen Klasse herangezogen. Dazu werden die Klassen der k Objekte gezählt und die häufigste wird als Vorhersage identifiziert. (Vgl. [LC14], S. 83)

Das Beispiel in Abbildung 3.1 soll das Vorgehen des kNN veranschaulichen. Der Graph zeigt bereits bekannt Objekte mit einer zweidimensionalen Ausprägung, die jeweils auf den Achsen abgetragen sind. Das unbekannte Objekt wird durch das Symbol x dargestellt. Die bekannten Objekte fallen in zwei verschiedene Klassen, die durch das Symbol $\oplus$ und ■ dargestellt werden. Der Abstand zwischen den Objekten sei proportional zur Distanz im Koordinatensystem. Die Kreise symbolisieren unterschiedlich gewählte k. Der durchgezogene Kreis umfasst die $k = 5$ nächsten Nachbarn. Für den gestrichelten Kreis gilt $k = 1$. Wird der äußere Kreis betrachtet, so ergibt sich für die Vorhersage die Klasse $\oplus$, da diese viermal vorkommt, während die Klasse ■ nur einmal vertreten ist. Dass die Wahl

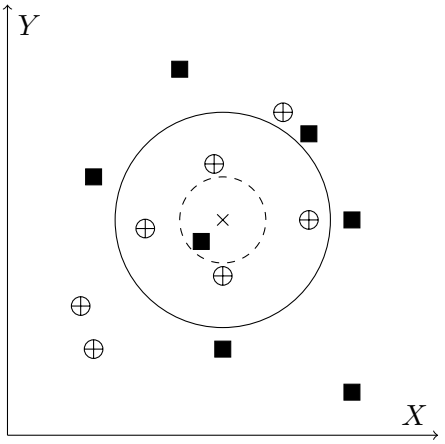


Abbildung 3.1: Visualisierung eines kNN Beispiels

eines passenden k wichtig ist zeigt die Betrachtung des inneren Kreises. Wird $k = 1$ gewählt so ergibt sich für das neue Objekt x die Klasse ■.

3.2 Entscheidungsbäume

Mit Hilfe von Entscheidungsbäumen lassen sich Objekte in Klassen einordnen. Es handelt sich folglich um eine Methode der Klassifizierung. Entscheidungsbäume sind Repräsentationsformen, bei der die Ergebnisse einer Bedingung verzweigt dargestellt werden. Aus diesen Verzweigungen können weitere Verzweigungen entstehen. Folgt man dem Baum vom Wurzelelement zum Blatt ist dies eine gute Visualisierung des Weges zur Klasse. (Vgl. [LC14], S.71-72)

Zur Veranschaulichung des Prinzips der Entscheidungsbäume wird ein Beispiel aus [LC14] herangezogen. In dem Beispiel geht es um die Bedingungen, die gegeben sein müssen, damit ein Golfspiel stattfindet. Die Abbildung 3.1 zeigt die Beispieldatenmenge.

Aus dieser Beispielmenge E kann nun ein Entscheidungsbaum erzeugt werden. Ein möglicher Baum zu den Beispieldaten ist in Abbildung 3.2 dargestellt. Ausgehend von der Attributmenge A wird zunächst ein Attribut $a \in A$ als Wurzel ausgewählt. In diesem Beispiel sei dieses Attribut *outlook*. Sei weiterhin w_a die Menge aller Werte, die a annehmen kann. In diesem Beispiel also *sunny, overcast und rainy*. Für jede Ausprägung $w \in w_a$ wird die Menge $E' \subseteq E$ gebildet, für die gilt $\forall e \in E' : w_a(e) = w$. Für jede dieser Ausprägungen wird eine Kante an die Wurzel gelegt. Gilt $E' = \emptyset$ wird die Kante mit NIL beendet. Besitzen *alle* Objekte aus E' die *gleiche* Klasse, wird die Kante mit einem Blatt abgeschlossen und mit entsprechender Klasse markiert. Im Beispiel bei der Ausprägung *overcast*. In den anderen Fällen wird ein neuer Entscheidungsbaum erzeugt, dessen Wurzel als Knoten an

#	outlook	temperature	humidity	windy	play
1	sunny	hot	high	false	no
2	sunny	hot	high	true	no
3	overcast	hot	high	false	yes
4	rainy	mild	high	flase	yes
5	rainy	cool	normal	false	yes
6	rainy	cool	normal	true	no
7	overcast	cool	normal	true	yes
8	sunny	mild	high	false	no
9	sunny	cool	normal	false	yes
10	rainy	mild	normal	false	yes
11	sunny	mild	normal	true	yes
12	overcast	mild	high	true	yes
13	overcast	hot	normal	false	yes
14	rainy	mild	high	true	no

Tabelle 3.1: Entscheidungsbaum: Beispiel Golfspiel

das Ende der Kante gehängt wird. Der neue Teilbaum wird ausgehend der Attributmenge $A' = A \setminus \{a\}$ und der Beispielmenge E' erzeugt.

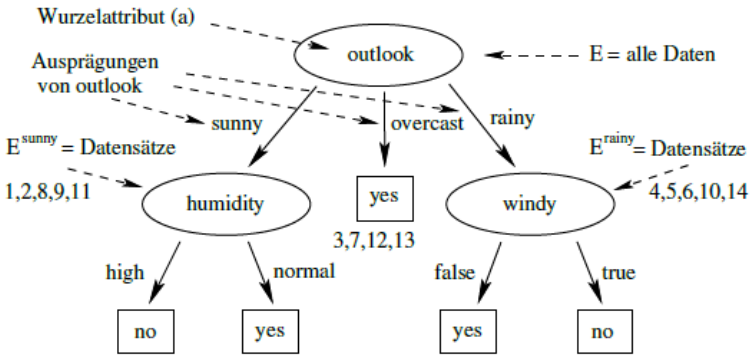


Abbildung 3.2: Entscheidungsbaum Golf Beispiel aus ([LC14],S.94)

Der in der Abbildung gezeigte Aufbau des Baumes ist nicht die einzige Möglichkeit der Strukturierung. Die Knoten des Entscheidungsbaumes könnten beliebig angeordnet werden. Die Frage ist nun, welcher Baum ist der beste Baum zur Darstellung des Problems? Es sollte ein möglichst kompakter Baum angestrebt werden, der nach wenigen Fragen zum Ergebnis, also zur Klasse, führt. Bei kom-

plexen Konstellationen ist der Aufwand alle mögliche Bäume zu generieren und diese zu vergleichen viel zu groß.

Die Entscheidung, wie der nächste Knoten ausgewählt wird, kann manuell durch den Anwender erfolgen. Dies wäre aber nur bei sehr kleinen Attributmengen zielführend. Eine weitere Überlegung ist es die Knoten zufällig zu wählen. Der Nachteil hier ist jedoch, das Bäume mit sehr langen Pfaden entstehen können, die den ganzen Baum unübersichtlich machen. Folglich muss es einen Weg geben den besten Knoten zu berechnen. Es haben sich insbesondere zwei Algorithmen für diese Art der Berechnung etabliert. Der ID3-Algorithmus und sein Nachfolger der C4.5-Algorithmus. Der ID3 definiert den besten Baum als den einfachsten, also kompaktesten Baum. Der Algorithmus ermittelt für jedes mögliche Attribut als nächsten Knoten den Informationsgewinn und wählt dasjenige Attribut mit dem höchsten Informationsgewinn aus. Zur Ermittlung des Informationsgehaltes sei hier auf die Literatur verwiesen - zum Beispiel [LC14]. Den Nachteil, dass der ID3 nicht mit numerischen Attributen umgehen kann löst sein Nachfolger der C4.5, indem dieser die numerischen Attribute in Intervalle und somit in Werte einer ordinalen Skala einteilt. (Vgl. [LC14], S.94-108)

Ein bekanntes Problem beim Entscheidungsbaumlernen ist das Overfitting. Als Overfitting bezeichnet man die Situation, wenn der Baum die Trainingsdaten auswendig lernt. Dann klassifiziert der Baum alle Objekte der Trainingsmenge korrekt, es kann jedoch sein, dass der Baum auf Testdaten nicht gut funktioniert. Um dies zu verhindern wird versucht die Bäume künstlich zu kürzen indem man fordert, dass jeder Unterbaum nur eine minimale Anzahl an Elementen besitzt oder einzelne Unterbäume eines kompletten Baumes nach der Generierung durch ein Blatt ersetzt. Beide Verfahren gehören zum sogenannten Pruning. (Vgl. [LC14], S.108-109)

3.3 A-Priori

Zu den wichtigsten Verfahren der Assoziationsanalyse zählt der A-Priori-Algorithmus. Ein beliebtes Beispiel der Assoziationsanalyse ist die Warenkorbanalyse. A-Priori hat das Ziel, sogenannte Frequent Itemsets aus einer Menge von Items zu finden. Items sind dabei beliebige Objekte zum Beispiel die Artikel eines Warenkorbs. Frequent Itemsets sind eine Menge von Items, deren relative Häufigkeit (Support) einen festgelegten Schwellwert überschreitet. Wenn der Support mit 2% festgelegt wird, kommt ein Frequent Itemset im Beispiel des Warenkorbs in mindestens 2% aller Einkäufe vor. Aus den Frequent Itemsets lassen sich Aussagen wie 'Wer A und B kauft, kauft auch C' ableiten. (Vgl. [LC14], S. 175-176)

Zum weiteren Verständnis werden an dieser Stelle die Begriffe Support und Konfidenz definiert.(Vgl. [LC14],S.74-75)

Support Der Support eines Itemsets ist das Verhältnis der Anzahl der Transaktion, in denen das Itemset vorkommt, zu der Anzahl aller Transaktion. Der Support einer Assoziationsregel ist gleich dem Support der Vereinigung von Prämisse und Konsequent der Regel: $supp(A \rightarrow B) = P(A \cup B)$.

Konfidenz Die Konfidenz einer Assoziationsregel bildet das Verhältnis der Anzahl der Transaktionen, die sowohl in der Prämisse als auch in der Konsequenz vorkommen zu der Anzahl an Transaktionen, die nur in der Prämisse enthalten sind: $conf(A \rightarrow B) = \frac{supp(A \rightarrow B)}{supp(A)}$. Die Konfidenz sagt aus, wie oft die Regel wirklich zutrifft in Relation zur Anzahl wie oft sie hätte zutreffen müssen.

Der A-Priori-Algorithmus arbeitet in zwei Schritten. Im ersten Schritt werden alle Frequent Itemsets mit einem ausreichenden Support gesucht, um im zweiten Schritt aus den gefundenen Mengen Assoziationsregeln zu generieren.

Die Suche nach Frequent Itemsets beginnt mit einelementigen Sets. Im nächsten Lauf werden darauf aufbauend zweielementige Itemsets gebildet, anschließend dreielementige und so weiter bis keine neuen Itemsets mehr gefunden werden. Für jedes gebildete Set wird der Support berechnet und das Element verworfen, wenn der Support nicht ausreichend ist. Zur Verbesserung der Komplexität und damit der Laufzeit wird die Monotonieeigenschaft der Itemsets genutzt: $A \subset B \rightarrow supp(A) \geq supp(B)$. Diese besagt, dass jede nichtleere Teilmenge eines Frequent Itemsets wiederum ein Frequent Itemset sein muss. Das bedeutet, dass der Support nicht kleiner werden kann, wenn ein Item aus einem Frequent Itemset entfernt wird und andersherum der Support nicht größer werden kann, wenn ein Item zu einem Frequent Itemset hinzugefügt wird. Das hat den Vorteil, dass bei der Erzeugung von n-elementigen Frequent Itemsets nur jene betrachtet werden müssen, die aus zwei (n-1)-elementigen Frequent Itemsets entstanden sind. (Vgl. [LC14], S.176-177)

Nachdem im ersten Schritt die Frequent Itemsets erzeugt wurden, werden im zweiten Schritt die Assoziationsregeln gebildet. Dazu wird jedes Frequent Itemset in zwei nichtleere Teilmengen A und $X \setminus A$ zerlegt. A wird dann zur Prämisse und X zur Konsequenz der Regel. Beispielsweise kann aus dem Frequent Itemset $X = a, b, c, d$ die Regel $(a, b) \rightarrow (c, d)$ gebildet werden. Sind alle möglichen Regeln gebildet, so muss geprüft werden, ob sie die zuvor festgelegte minimale Konfidenz erfüllen. Alle Regeln, die den minimalen Support und die minimale Konfidenz erfüllen bilden das Ergebnis des A-Priori-Algorithmus. (Vgl. [LC14], S.179-180)

3.4 Lineare Regression

Ziel einer linearen Regression ist es, einen Trend oder durchschnittlichen Zusammenhang von numerischen Attributen zu ermitteln. Voraussetzung dabei ist, dass ein linearer Zusammenhang zwischen den x und y Werten besteht. Der Zusammenhang der Datensätze einer Beispielmenge $E \subset X \times Y$ soll durch die Regressionsfunktion $\hat{y} = f(x)$ beschrieben werden. Ziel ist es, den quadratischen Fehler zwischen den tatsächlichen Zielwerten und dem berechneten Wert zu minimieren: $error = \sum(y_i - \hat{y_i})^2 \rightarrow min$

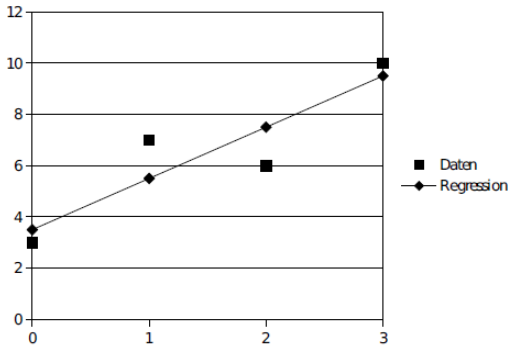


Abbildung 3.3: Beispiel einer linearen Regression aus ([LC14],S.63)

Abbildung 3.3 zeigt ein Beispiel einer linearen Regression für die Werte *(2,3)*, *(3,7)*, *(4,6)*, *(5,10)*.

3.5 k-means

Die k-means Methode ist das klassische Verfahren zur Clusterbildung (s. [WFH11], S.139 und [HKP12], S.402). Das Verfahren hat die Vorteile, dass es einfach zu implementieren ist und vergleichsweise wenige Iterationen benötigt, um die Cluster zu bilden. Daher ist es auch so populär. (Vgl. [LC14], S. 141) Dagegen werden jedoch auch Nachteile genannt. Der k-means benötigt eine fest vorgegebene Anzahl an Clustern, was seine Flexibilität einschränkt. Außerdem ist der Einfluss von Ausreißern sehr hoch, da alle Eingabedaten betrachtet werden. Im Gegensatz zu anderen Clustering Verfahren bildet der k-means keine Hierarchie der Cluster ab. (Vgl. [Mya07], S.121-122).

Im Folgenden wird nun beschrieben, wie das k-menas Verfahren funktioniert. Grundsätzlich partitioniert der k-menas Algorithmus n Objekte der Eingabemenge in k Cluster, deren Anzahl zu Beginn festgelegt wird. Die Ähnlichkeit der Objekte innerhalb eines Clusters ist dabei sehr hoch, während die Ähnlichkeit zwischen Objekten aus verschiedenen Clustern gering ist. (Vgl. [HKP12], S.402) Die

Cluster sind dabei disjunkt. Das heißt, dass ein Objekt nur einem Cluster zugewiesen wird und jedes Cluster mindestens ein Objekt beinhaltet (s. [LC14], S.136).

Das Verfahren des k-means ist ein Iteratives Verfahren, in dem die Zentren der Cluster immer wieder neu berechnet werden und die Cluster dementsprechend angepasst werden, bis schließlich die finalen Cluster ermittelt wurden. Am Anfang werden zufällig k Punkte bestimmt. Diese Punkte bilden die Centroiden der Cluster. Jedes Objekt der Eingabemenge wird nun dem Cluster zugewiesen, dem es am ähnlichsten bzw. am nächsten ist. Dieses Maß kann beispielsweise durch die Euklidische Distanz ermittelt werden. Ist die Zuweisung abgeschlossen sind initiale Cluster entstanden. Von diesen Clustern werden nun die neuen Centroiden berechnet und die Objekte neu angeordnet. Das ganze Verfahren wird solange wiederholt, bis es keine neuen Zuweisungen mehr gibt und die finalen Cluster feststehen. (Vgl. [HKP12], S.402-403) In Abbildung 3.4 auf Seite 357 sind die Iterationen visualisiert.

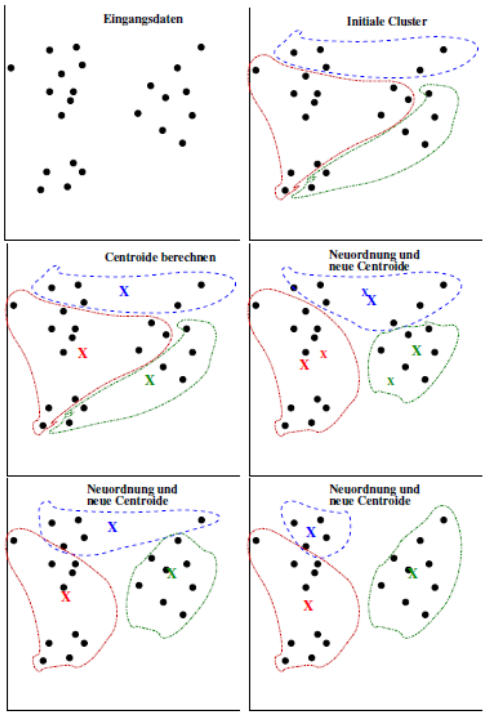


Abbildung 3.4: Ablauf des k-means. (nach [LC14], S.139-140)

3.6 k-medoid

Das k-medoid Verfahren ist dem k-means sehr ähnlich. Der Unterschied ist, dass statt der Betrachtung des Centroids der Medoid gewählt wird. Der Medoid muss -

im Gegensatz zum Centroiden - einen Punkt aus den Eingabedaten repräsentieren.

Bei kardinalen Skalen ist dies das Objekt, das dem Centroiden am nächsten ist. Bei nominalen oder ordinalen Daten ist dies nicht ohne weiteres zu berechnen. Hier wird der 'bessere' Medoid durch Tauschverfahren ermittelt. Der k-medoid ist robuster als der k-means was den Einfluss von Ausreißern in der Datenmenge angeht, da der Medoid immer ein bestehendes Objekt ist. Ein bekannter Algorithmus ist *Partitioning Around Medoids (PAM)*, der eine vollständige Suche in der Datenmenge nach einem Medoid durchführt. Dadurch ist der PAM in der Lage gute Cluster zu finden, jedoch erhöht sich die Laufzeit des Algorithmus mit der Anzahl der Objekte enorm, weshalb der Algorithmus nur für kleinere Datenmengen effektiv anwendbar ist. Abhilfe verschafft hier der *Clustering Large Applications based on RANdomized Search (CLARANS)* Algorithmus. CLARANS durchsucht nicht die gesamte Eingabemenge, sondern berücksichtigt zufallsbasiert nur einen Teil der Menge. Dadurch steigert sich die Effizienz deutlich ohne dabei wesentlich schlechtere Ergebnisse zu erzielen. (Vgl. [LC14], S.148 - 153)

3.7 Hierarchisches Clustering

Hierarchisches Clustering nutzt die Gruppierung von Objekten in eine Baumstruktur. Grundsätzlich wird zwischen zwei Methoden unterschieden: dem Agglomerativen Clustering und dem Divisivem Clustering. Bei dem ersten Verfahren wird nach der Bootom-Up Methode, beim zweiten nach der Top-Down Methode vorgegangen. Ein großer Nachteil an dem Verfahren ist, dass wenn eine Entscheidung zur Teilung oder Verschmelzung einmal getroffen wurde, diese nicht wieder rückgängig gemacht werden kann. Die Qualität des Verfahrens hängt also insbesondere von der ersten Entscheidungen ab. Vorteilhaft beispielsweise gegenüber dem k-menas ist aber, dass die Anzahl an Clustern zuvor nicht festgelegt werden muss. Im folgenden werden nun die beiden Verfahren vorgestellt. (Vgl. [HKP12], S. 408-409)

Agglomeratives Clustering Bei diesem Verfahren wird die Bottom-Up Methode verwendet. Zunächst wird jedes Objekt in sein eigenes Cluster eingeordnet. Im weiteren Verlauf werden die atomaren Cluster schrittweise immer wieder mit anderen Clustern verschmolzen bis eine Abbruchbedingung erreicht wird. Beispielsweise bis eine bestimmte Anzahl an Clustern erreicht wurde, oder bis alle Cluster vereint wurden.

Divisives Clustering Dieses Verfahren baut eine Hierarchie in umgekehrter Reihenfolge auf. Es startet mit einem Cluster, in dem Alle Objekte vereint sind

und teilt dies schrittweise in kleinere Cluster auf, bis jedes Cluster genau ein Objekt beinhaltet, oder eine andere Abbruchbedingung erfüllt ist.

Meistens wird für die Darstellung des hierarchischen Clusterings ein Dendogramm genutzt. Dabei handelt es sich um eine Baumstruktur mit folgenden Eigenschaften (Vgl. [LC14], S.136):

- Die Wurzel repräsentiert die gesamte Datenmenge.
- Ein innerer Knoten repräsentiert die Vereinigung aller darunterliegenden Teilbäume und deren Objekte.
- Die Blätter repräsentieren jeweils ein einzelnes Objekt.

Die Abbildung 3.5 zeigt beispielhaft ein Dendogramm.

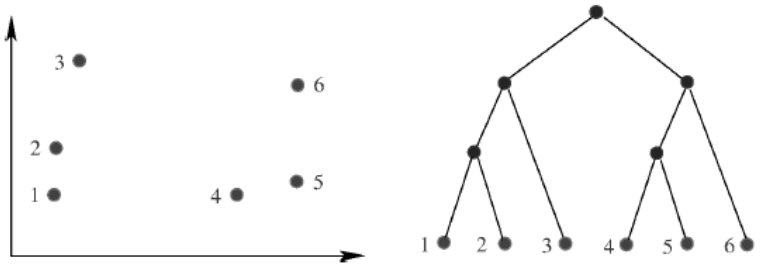


Abbildung 3.5: Hierarchisches Clustering (s.[LC14], S.137)

4 Zusammenfassung

In dieser Seminararbeit wurde zunächst in das Themengebiet des Data Minings eingeführt. Dazu wurden die Vorgehensmodelle KDD und CRISP-DM vorgestellt.Anschließend wurden Aufgaben und Methoden des Data Minings beschrieben. In Abbildung 4.1 sind noch einmal alle vorgestellten Methoden und ihren zugeordneten Anwendungsklassen aufgelistet.

	Klassifikation	**Clustering**	**Assoziationsanalyse**	**Num. Vorhersage**
k Nearest Neighbour	X			
Entscheidungsbäume	X			
A-Priori			X	
Lineare Regression				X
k-means		X		
k-medoid		X		
Hier. Clustering		X		

Tabelle 4.1: Data Mining Methoden und ihre Anwendungsklassen

Die in dieser Arbeit vorgestellten Algorithmen und Methoden sind eine Auswahl aus vielen weiteren Verfahren, die dem Themengebiet des Data Minings zuzuordnen sind. Die in der Tabelle gezeigte Zuordnung der Methoden zu den Anwendungsklassen basiert auf den Hauptanwendungsfällen der Methoden. In abgewandelter Form oder in Kombination mit anderen Methoden ist der Einsatz einer Methode auch in anderen Anwendungsgebieten möglich.

Literatur

[Cor12] CORPORATION, EMC ; CORPORATION, EMC (Hrsg.): *Prognose zum Volumen der jährlich generierten digitalen Datenmenge weltweit in den Jahren 2005 bis 2020 (in Exabyte).* `http://de.statista.com/statistik/daten/studie/267974/umfrage/prognose-zum-weltweit-generierten-datenvolumen/`. Version: 2012

[Fay96] FAYYAD, Usama M.: *Advances in knowledge discovery and data mining*. Menlo Park and Calif : AAAI Press and MIT Press, 1996. – ISBN 0–262–56097–6

[HKP12] HAN, Jiawei ; KAMBER, Micheline ; PEI, Jian: *Data mining: Concepts and techniques, third edition*. 3rd ed. Waltham and Mass : Morgan Kaufmann Publishers, 2012 (The Morgan Kaufmann series in data management systems). – ISBN 978–0–12–381479–1

[LC14] LÄMMEL, Uwe ; CLEVE, Jürgen: *Data Mining*. [S.l : s.n.], 2014. – ISBN 9783486720341

[MJ14] MYATT, Glenn J. ; JOHNSON, Wayne P.: *Making sense of data I: A practical guide to exploratory data analysis and data mining*. Second edition. 2014. – ISBN 9781118407417

[Mya07] MYATT, Glenn J.: *Making sense of data*. Bd. / Glenn J. Myatt ; [1]: *A practical guide to exploratory data analysis and data mining*. Hoboken and NJ : Wiley, 2007. – ISBN 978–0–470–07471–8

[Run10] RUNKLER, Thomas A.: *Data-Mining: Methoden und Algorithmen intelligenter Datenanalyse ; mit 7 Tabellen*. 1. Aufl. Wiesbaden : Vieweg + Teubner, 2010 (Studium). – ISBN 978–3–8348–0858–5

[WFH11] WITTEN, I. H. ; FRANK, Eibe ; HALL, Mark A.: *Data mining: Practical machine learning tools and techniques*. 3rd ed. Burlington and MA : Morgan Kaufmann, 2011 ([Morgan Kaufmann series in data management systems]). – ISBN 978–0–12–374856–0

Abschließende Erklärung

Ich versichere hiermit, dass ich meine Seminararbeit Data Mining Methoden selbständig und ohne fremde Hilfe angefertigt habe, und dass ich alle von anderen Autoren wörtlich übernommenen Stellen wie auch die sich an die Gedankengänge anderer Autoren eng anlegenden Ausführungen meiner Arbeit besonders gekennzeichnet und die Quellen zitiert habe.

Oldenburg, den 26. September 2016 Simon Becker

CARL VON OSSIETZKY universität OLDENBURG

VERY LARGE BUSINESS APPLICATIONS
Carl von Ossietzky Universität Oldenburg

Was ist Business Intelligence?

Seminararbeit

Betreuer: M. Eng. & Tech. Viktor Dmitriyev
Dr.-Ing. Andreas Solsbach
M.Sc. Ola Mustafa

Vorgelegt von: Muhammed Altuntas
muhammed.altuntas@uni-oldenburg.de

Abgabetermin: 02. Dezember 2015

in Zusammenarbeit mit

cewe
BEST IN PRINT

Inhaltsverzeichnis

Abkürzungsverzeichnis

BI	Business Intelligence
BW	Business Warehouse
CRM	Customer Relation Management
DSS	Decision Support System
DWH	Data Warehouse
DWS	Data-Warehouse-System
EIS	Executive Information System
ETL	Extraktion Transformation Laden
EUS	Entscheidungs-unterstützungssystem
FIS	Führung-sinformationssystem
IIA	International Institute for Analyticsl
SQL	Structured Query Language
IBM	International Business Machines Corporation
IT	Informations-technologie
MIS	Management Information System
MSS	Management Support System
MUS	Management-unter-stützungs-system
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
SAP	Software Anwendungen und Produkte

Abbildungen

1 Einleitung

Diese Seminararbeit wird im Rahmen der diesjährigen Projektgruppe Hadoop verfasst und soll mit der Fragestellung „Was ist Business Intelligence?“ sowohl einen allgemeinen Überblick verschaffen, als auch einen Ausblick über zukünftige Trends im Bereich Business Intelligence, kurz BI, vorstellen.

1.1 Motivation

IT-basierte Managementunterstützung ist im Informationszeitalter einer der wichtigsten Wettbewerbsfaktoren für Unternehmen. In ihrem Werk stellen Gómez, Rautenstrauch und Cissek Unternehmen vor, die BI-Lösungen für die Unterstützung der Wettbewerbsituation am Markt anbieten (vgl. [GRC09, S. 7f.]). Es wird als Pflicht gesehen, als Führungskraft im Unternehmen sich mit dem Thema BI auseinanderzusetzen. Vor allem Entscheidungsträger, als auch alle Teilnehmer im Unternehmen, die an BI Prozessen beteiligt sind, sollten diesem Thema vertraut sein. (vgl. [GRC09, S. 8f.])

Richtige Entscheidungen in wichtigen Anwendungsgebieten verschaffen Unternehmen einen Wettbewerbsvorteil gegenüber anderen Unternehmen auf dem Markt. Entscheidungsrelevante Daten sind im Informationszeitalter immer schwieriger zu analysieren, da im Laufe der Zeit immer mehr unstrukturierte oder semistrukturierte Daten generiert werden. (vgl. [ML13, S. 1f.])

Berichte über Prognosen nehmen immer mehr an Bedeutung zu. Unternehmen verlegen daher ihren Schwerpunkt von der ehemals dominierenden Analyse historischer Daten in prognoseorientierte Analysen. Daher sind nicht nur Reports und Analysen für Fakten und Geschäftsprozesse wichtig, sondern auch für Analysen auf Prognosen anhand von mathematischen Modellen. (vgl. [o.V10])

Für Unternehmen ist die Analyse sowohl von strukturierten Daten als auch von unstrukturierten oder semi-strukturierte Daten für die Zukunft besonders wichtig. International Institute for Analytics (IIA) beschreibt konzeptionell beide Themengebiete als Analytics 1.0 und Analytics 2.0 und führt diese in ein neues Konzept zusammen und bezeichnet dieses als Analytics 3.0. Das Konzept unterstützt sowohl die Analyse von strukturierten als auch von unstrukturierten bzw. semistrukturierten Daten. (vgl. [Dav13])

1.2 Zielsetzung und Struktur der Arbeit

Das übergeordnete Ziel dieser Seminararbeit ist es ein Grundverständnis im Bereich Business Intelligence aufzubauen und sowohl Marktführer als auch zukünftige Trends der Branche vorzustellen.

Einführend in Kapitel 2 wird das Thema Business Intelligence und die damit verbundene Data Warehouse-Architektur vorgestellt, um ein Grundverständnis zu vermitteln. Anschließend wird in Kapitel 3 eine Marktanalyse von Gartner vorgestellt, in der Unternehmen nach bestimmten Eigenschaften in einer Matrix aufgestellt werden, um die Marktführerposition des jeweiligen Unternehmens hervorzuheben. Weiterhin werden in Kapitel 4 Zukunftstrends auf dem BI-Markt erläutert und BI-Lösungen von Unternehmen mit Einbindung dieser Trends vorgestellt. Abschließend gibt es in Kapitel 5 eine Zusammenfassung der Ergebnisse.

2 Grundlagen

In dem Grundlagenkapitel wird zunächst in Abschnitt 2.1 eine Einfährung über BI gegeben und der historische Hintergrund kurz beleuchtet. Anschließend geht es in Abschnitt 2.2 um die Definition des BI-Begriffs. Weiterhin geht es in Abschnitt 2.3 um das Data Warehouse und die Datenbeschaffung anhand des ETL-Prozesses (s. Abschnitt 2.4). Abschnitt 2.5 stellt OLTP und OLAP, grundlegend vor und schließt somit das Grundlagenkapitel.

2.1 Einführung und Historie von Business Intelligence

In der betrieblichen Praxis existiert der Begriff Business Intelligence bereits seit den 90er Jahren. Die ersten Versuche, Führungskräfte in ihren Entscheidungen anhand von Informationssystemen zu unterstützen, starteten schon in den 60er Jahren, wodurch auch die historische Entwicklung des Begriffs begann. Der Begriff des Management Support Systems, kurz MSS, etablierte sich im Laufe der 80er Jahre und wird im Deutschen als Managementunterstützungsysteme (MUS) bezeichnet (vgl. [KBM10, S. 1]). Laut Hahne werden „unter dem Begriff Management Support Systeme (MSS) [...] alle Varianten der elektronischen Unterstützung betrieblicher Entscheidungsträger bei der Abwicklung ihrer Aufgaben [...]“ zusammengefasst (s. [Hah05, S. 5]). Zu den MSS gehören unter anderem Managementinformationssysteme (MIS), Entscheidungsunterstützungssysteme (EUS) und Führungsinformationssysteme (FIS) und gelten als informationstechnisch unterstützende Systeme.

Zum ersten Mal kam das MIS in den 60er Jahren zum Einsatz und diente zur Informationsversorgung des Managements. Die mangelnde Möglichkeit Entscheidungen anhand von starren Berichten, sowie durch das reine Berichtswesen zu treffen, führte dazu, dass sich EUS etablierten. Die Entscheidungsqualität des Managements sollte dadurch verbessert werden, indem zur Entscheidungsunterstützung Methoden und Modelle angeboten werden konnten. Die Entscheidungen über bestimmte Sachzusammenhänge konnten durch Tabellenkalkulationsprogramme verbessert werden, wobei auch in diesem Fall Systemschwächen auftraten, die durch das FIS aufgebaut wurden. Einer dieser Schwächen der EUS war die Erkennung besonderer Warnsignale und Situationen. Für die Etablierung von FIS

zur Mitte der 80er Jahren war vor allem das Generieren von Warnmeldungen und die innovativen Auswertungsmöglichkeiten von Datenbeständen ausschlaggebend. Durch das FIS wurde das obere Management adressiert, die durch bessere Möglichkeiten der Entwicklung graphischer Benutzeroberflächen eine verbesserte Grundlage zur Entscheidungsunterstützung nutzen konnten. (vgl. [Hah05, S. 6f.])

2.2 Definition: Business Intelligence

Diese Thematik wird in [KBM10, S. 3] aufgegriffen und in Anlehnung zu Chamoni und Gluchowski werden verschiedene Definitionsansätze zum Begriff Business Intelligence vorgestellt.

1. Definition: Enges BI-Verständnis

Unter Business Intelligence im engeren Sinne wird verstanden, dass hierzu wenige Applikationen, wie zum Beispiel MIS und EIS, unmittelbar zur Entscheidungsunterstützung beitragen.

2. Definition: Analyseorientiertes BI-Verständnis

Im analyseorientiertem Sinne beinhaltet Business Intelligence sämtliche Anwendungen, zu denen der Entscheider oder der Entscheidungsmitwirkende einen direkten Zugriff hat, um in der Systemoberfläche mit bestimmten Funktionalitäten Analysen unter anderem im Bereich Data Mining und Text Mining und weiteren analyseorientierten Methoden durchführen kann.

3. Definition: Weites BI-Verständnis

Das weite BI-Verständnis umfasst alle direkten und indirekten Anwendungen, die zur Entscheidungsunterstützung beitragen. Insbesondere geht es hier um Reporting-Funktionalitäten, Datenaufbereitung und Datenspeicherung.

Durch die breiten Definitionsansätze von Kemper, Baars und Mehanna, wird ein basales Verständnis von BI vermittelt, jedoch sind die Erläuterungen der genannten Autoren nicht frei von Kritik. So wirken „[...] viele Ansätze nicht trennscharf, weisen zum Teil einen hohen Grad an Beliebigkeit auf oder lassen Abgrenzungen zu bestehenden Ansätzen vermissen.“ (s. [KBM10, S. 5]).

In Abbildung 2.1 ist eine mögliche Strukturierung der Definitionsansätze zu erkennen. Durch die Spezifizierung der Achsen in der Abbildung ist es Chamoni und Gluchowski gelungen die Definitionsansätze anhand von Technik und Anwendung auf der horizontalen Achsen und Prozessschwerpunkten der Datenbereitstellung und der Datenauswertung auf der vertikalen Achse überschaubar darzustellen und voneinander zu trennen.

Ziel dieses Abschnittes ist es nicht eine allgemeingültige Definition vorzustellen, sondern vielmehr anhand der Defintionsansätze das Verständnis für BI im engeren, analyseorientierten sowie weiten Sinne zu schaffen.

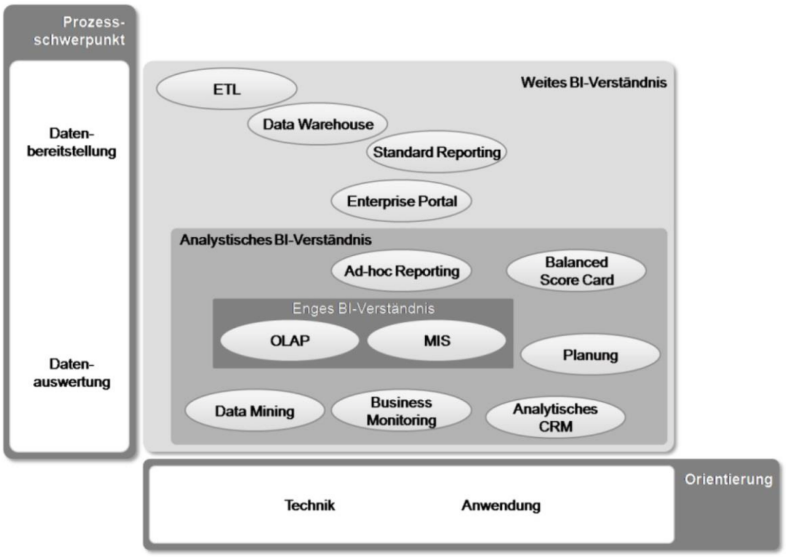


Abbildung 2.1: BI-Verständnis (s. [GRC09, S. 13])

Bedingt durch das wachsende Marktumfeld und die Entwicklung des globalen Datenvolumens entwickelten sich im Laufe der Jahre weitere Anforderungen des Managements an die MSS. Da Datenbestände der Systeme isoliert behandelt werden und dadurch die, „Struktur als zu inflexibel zur Anpassung an die zunehmend schneller wachsenden betrieblichen Anforderungen von Fach- und Führungskräften galten“ (s. [Hah05, S. 6f.]), können herkömmliche Systeme der MSS diesen Anforderungen nicht mehr gerecht werden. Dieselbe Meinung wird auch durch Kemper, Baars und Mehanna in ([KBM10]) vertreten, die daher Business Intelligence als „einen integrierten, unternehmensspezifischen, IT-basierten Gesamtansatz zu betrieblichen Entscheidungsunterstützung“ (s. [KBM10, S. 8f.]) bezeichnen.

2.3 Data-Warehouse-System

Mit der sich ständig weiterentwickelnden Informationstechnologie und insbesondere im Zuge des Informationszeitalters werden immer größere Datenbestände in Unternehmen gespeichert. Es ist nicht von Innovation zu sprechen, dass die Speicherung und Verwaltung von Daten in Informationssystemen erfolgen. Die Gewinnung von Informationen kann entweder aus Unternehmensdaten erfolgen oder durch den externen Zukauf. Interessant wird es dann, wenn Daten aus heterogenen Quellen in einem Informationssystem zusammengeführt werden. Indirekte Faktoren wie Informationsqualität und -quantität können auch zum Unternehmenserfolg beitragen, da sie die Geschäftsprozessabwicklung fördern. Aus

diesem Grund „[...] wird Information immer häufiger als vierte Säule zu den drei konventionellen Produktionsfaktoren Boden, Arbeit und Kapital hinzugezählt." (s. [GRC09, S. 60]).

Die größte Motivation für ein Data Warehouse ist es, dass in Unternehmen große Datenbestände vorhanden sind mit der Absicht entscheidungsrelevante Informationen aus diesen zu gewinnen. Seit der 90er Jahren nahm der Begriff Data Warehouse, kurz DWH, an Popularität zu. Der Begriff wurde vielseitig in Beiträgen und Zeitungsartikeln verwendet, woraufhin die Notwendigkeit bestand, eine Vereinheitlichung einzuführen. „Dualismus von Informatik und Betriebswirtschaft" (s. [BG13, S. 5]), so wird es von Bauer und Günzel definiert. Sowohl technisch, als auch betriebswirtschaftlich wird für den Begriff DWH ein Verständnis geschaffen. Aus technischer Sicht beschreibt ein DWH eine Datenbank, die aus unterschiedlichen Systemen Daten integriert und diese für die betriebswirtschaftliche Sicht für Analyse- und Auswertungsmöglichkeiten dem Anwender zur Verfügung stellt.

Von Inmon wurde einer der ersten Definitionen im Bereich DWH aufgestellt: „A data warehouse is a subject-oriented, integrated, non-volatile and time variant collection of data in support of management's decisions." (s. [Inm05, S.29]). Mit „subject-oriented" ist gemeint, dass der Fokus nicht mehr auf Datenpflege bestimmter Fachbereiche liegt, sondern vielmehr in deren Auswertung. Unter „integrated" ist zu verstehen, dass Daten zur weiteren Verarbeitung aus unterschiedlichen Quellen integriert und dabei non-volatile", das heißt nicht flüchtig, im Data Warehouse gespeichert werden. Des Weiteren ist hierbei die Entfernung oder Änderung der integrierten Daten nicht vorgesehen. Das führt dazu, dass Vergleiche der Daten über die Zeit („time variance") angestellt werden können. (vgl. [BG13, S. 7f.])

Diese Definition erläutert im Folgenden vier Eigenschaften des DWHs (vgl. [KBM10, S. 19-21]):

Themenorientierung: Die Unterstützung der Geschäftsprozesse in Unternehmen erfolgt durch die Datenhaltung aus operativen Systemen. Hierbei soll das Management durch das DWH in seinen Entscheidungen unmittelbar mit Informationen zu Kerngebieten unterstützt werden.

Integration und einheitliche Datenbasis: Im DWH wird aus allen Daten, die durch die Integration aus unterschiedlichen Quellen gesammelt werden, eine einheitliche Datenbasis geschaffen, die inhaltlich keine Redundanzen, Inkonsistenzen und semantische Widersprüche aufweist.

Zeitraumbezug: Das DWH ermöglicht Analysen und Auswertungen von bestimmten Zeiträumen. Im Data Warehouse können Daten im Gegensatz zu operativen Systemen historisch abgelegt werden und repräsentieren einen Zeitraum. In operativen Systemen können Daten transaktionsorientiert und zeitpunktbezogen

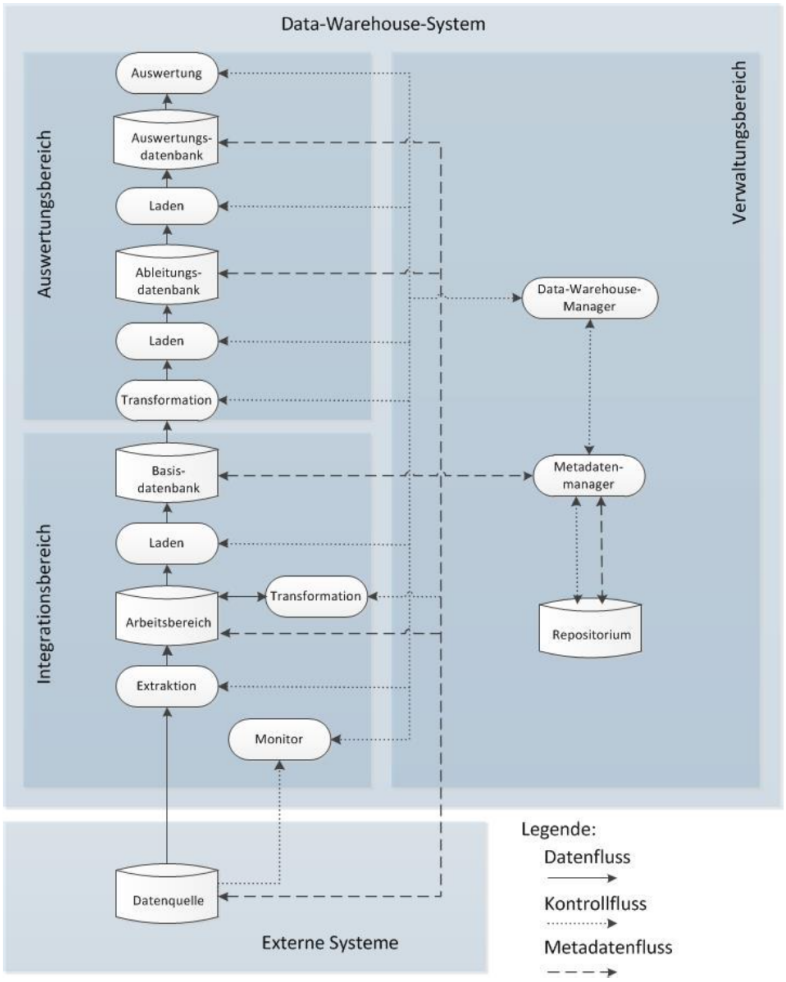


Abbildung 2.2: Data-Warehouse-System (vgl. [BG13, S. 42])

abgelegt werden.

Dauerhafte Datenablage: Das DWH zeichnet sich unter anderem dadurch aus, dass ein historischer Verlauf der Daten vorhanden ist. Im Data Warehouse werden Daten nicht entfernt oder geändert, wohingegen in operativen Systemen historische Daten nicht von Bedeutung sind. Dadurch können aus dem DWH heraus betriebswirtschaftliche Analysen über historische Entwicklungen von Geschäftsprozessen durchgeführt werden.

Bauer und Günzel bewerten die Definition von Inmon als nicht aussagekräftig genug und erweitern diese hinsichtlich verschiedener Gesichtspunkte der physischen Integration und der Analyse: „Ein Data-Warehouse-System ist ein physisches Informationssystem, das eine integrierte Sicht auf beliebige Daten zu Auswertungszwecken ermöglicht“ (s. [BG13, S. 61]). Abbildung 2.2 der Referenzarchitektur eines Data-Warehouse-Systems von Bauer und Günzel veranschaulicht

das Data-Warehouse-System, kurz DWS, mit seinen jeweiligen Komponenten und Datenflüssen.

Der Bereich der Integration, Auswertung und Verwaltung ist hierbei sehr wichtig. Der Data-Warehouse-Prozess, der „[...] den dynamischen Vorgang, angefangen beim Datenbeschaffungsprozess über das Speichern bis zur Auswertung der Daten, d.h. den Fluss und die Verarbeitung der Daten aus den Datenquellen bis zum Auswertungsergebnis beim Anwender“ (s. [BG13, S. 9]) beschreibt, ist auch ein wichtiger Bestandteil.

Damit enthält die DWS-Referenzarchitektur mit dem Data-Warehouse-Prozess die nicht nur für die Integrations- und Auswertungszwecke notwendigen Komponenten, sondern auch einen weiteren wichtigen Aspekt. Die Legende der Abbildung 2.2 zeigt (Meta)Daten- und Kontrollflüsse, die zwischen den Komponenten zusammenwirken. Des Weiteren beinhaltet die DWS- Referenzarchitektur verschiedene Operanden, die als Datenbehälter wirken und Operatoren, die zur Aufbereitungs- und Auswertungszwecke dienen. Folgende Komponenten des DWS- Referenzarchitektur werden nun näher beschrieben (s. Abbildung 2.2):

Integrationsbereich: Mittels Extraktion werden die Daten von externen Systemen durch die Datenquelle in den Arbeitsbereich kopiert, damit im nächsten Schritt diese transformiert werden können. Durch den Transformationsschritt werden die Ergebnisse langfristig in die Basisdatenbank geladen und abgelegt, daher ist der Arbeitsbereich eine temporäre Datenablage. Die Basisdatenbank bildet die Basis für zukünftige Auswertungen und beinhalten die Daten- und Schemaintegration aus operativen Quellen. Mit Hilfe des Monitors werden relevante Daten aus den Quellen ausgewählt. Folglich wird dieser Bereich als Datenbeschaffungsprozess bezeichnet und ist als Integrationsbereich gekennzeichnet. (vgl. [BG13, S. 40f.])

Auswertungsbereich: Der Auswertungsbereich, der dem Integrationsbereich folgt, legt seinen Fokus mit der Ableitungs- und Auswertungsdatenbank auf die Auswertung der aus der Basisdatenbank transformierten und geladenen Daten. Daher sind die beiden Datenbanken in der Modellierung denormalisiert und auswertungsorientiert. Die aus der Basisdatenbank zur Transformation ausgelegten Daten werden in die Ableitungsdatenbank geladen. Es werden gemeinsame Kennzahlen und Dimensionen gebildet. Für die Auswertungen erhält die Auswertungsdatenbank nur einen Datenausschnitt aus der Ableitungsdatenbank. (vgl. [BG13, S. 41])

Verwaltungsbereich: Für die Verwaltung und Steuerung der gesamten Komponenten des DWS ist der Verwaltungsbereich mit drei Komponenten zuständig. Diese sind das Repositorium, der Data-Warehouse-Manager und Metadatenmanager. Der Metadatenmanager versorgt das Repositorium und weitere Kompo-

nenten mit Metadaten. Das Repositorium sorgt für einen gesamten Überblick über die Inhalte des gesamten Data-Warehouse-Prozess. Die Aufgabe zur Initiierung, Steuerung und Kontrolle der gesamten Data-Warehousing-Prozesse durch Kontrollflüsse übernimmt der Data-Warehouse-Manager. Als zentrale Komponente des DWS ist der Data-Warehouse-Manager somit verantwortlich für die gesamte Steuerung der Komponenten im DWS. Die Meldung über Änderungen von Daten in den Quellen, die für die Auswertungs- und Ableitungsdatenbank wichtig sind, erfolgt durch den Monitor. Extraktoren wählen Daten aus den Datenquellen aus und transportieren diese in den Arbeitsbereich. Durch Transformationsprozesse werden diese Daten im Arbeitsbereich oder auch in der Ableitungsdatenbank vereinheitlicht, bereinigt und integriert. Die Ladekomponenten übernehmen das Laden der transformierten Daten in die Basisdatenbank und im weiteren Schritt in die Auswertungs- und Ableitungsdatenbank. In diesem Bereich können bestimmte Daten zur Auswertung und Präsentation mit Hilfe der Auswertungskomponenten genutzt werden. (vgl. [BG13, S. 43])

2.4 ETL-Prozess

In Abschnitt 2.3 wurde der Datenbeschaffungsprozess in Zusammenhang mit dem DWS erläutert und die einzelnen DWS-Komponenten mit ihren Aufgaben und Zuständigkeiten erklärt. In diesem Abschnitt geht es darum, den grundlegenden Prozess zur Datenbeschaffung anhand des ETL- Prozesses darzustellen.

Extraktion: In der Extraktion geht es darum, dass Daten aus Datenquellen in den Arbeitsbereich übertragen werden. Zum Beispiel können die Daten mit einem zeitstempelbasierten Vorgang aus den Datenquellen extrahiert werden, um jeweils diese je nach Zeitstempel in den Arbeitsbereich zu überführen. Als weitere Aufgabe der Extraktion wird die Steuerung der Auswahl von Quellen gezählt, die in das DWS zu importieren sind. (vgl. [BG13, S. 56])

Transformation: Im Anschluss zum Extraktionsschritt müssen die extrahierten Daten für die nächsten Schritte transformiert werden, um in die Basisdatenbank oder von dieser in die Ableitungsdatenbank geladen zu werden. Im Transformationsschritt werden Aspekte wie Schemaintegration, Datenintegration und Datenbereinigung behandelt. Eine Vereinheitlichung der Daten aus heterogenen Datenquellen findet statt, bevor sie in die Datenbank übertragen werden. Unter anderem zählen dazu die Anpassung von Datentypen, Konvertierung von Codierungen, Vereinheitlichung von Zeichenketten und Vereinheitlichung von Datumsangaben. Die Datenmigration beschäftigt sich genau mit dieser Thematik, wohingegen in der Datenbereinigung es um bestimmte Verfahren und Methoden geht, bei der die Pflege und Korrektur von fehlerhaften, redundanten aber auch veralteten Werten

durchgeführt wird. (vgl. [BG13, S. 57])

Laden: Nach den Transformationsschritten liegen die Daten gereinigt und aufbereitet im Arbeitsbereich zur Speicherung und Auswertung zur Verfügung. Das Laden der Daten in die jeweiligen Datenbanken geschieht mit Hilfe von zwei Komponenten. Auswertungsunabhängige Daten werden aus dem Arbeitsbereich in die Basisdatenbank übertragen, wohingegen auswertungsspezifische Daten aus der Basisdatenbank in die Ableitungsdatenbank (oder von der Ableitungsdatenbank in die Auswertungsdatenbank) geladen werden. (vgl. [BG13, S. 58])

2.5 OLAP und OLTP

Abbildung 2.3 verdeutlicht den Prozess zwischen OLTP, ETL und OLAP. Was OLTP und OLAP bedeutet wird folgend in Form von Definitionen und Erläuterungen diesbezüglich dargestellt.

„**OLTP** (Online Transaction Processing) sind transaktionsorientierte Prozesse, welche die operativen Daten des Unternehmens in (standardisierten) Softwarelösungen bearbeiten und in normalisierten Datenbanken oder flachen Dateien speichern.“ (s. [Gro15, S.115])

OLTP-Abfragen werden als Mikrodaten bezeichnet und könnten interessant als „Punktabfrage“ für Sachbearbeiter dienen, die zu einer gewissen Thematik eine Information brauchen. Als Makrodaten werden OLAP-Abfragen bezeichnet, das hier nicht Einzefälle sondern eine Fallmenge untersucht wird. Mitarbeiter aus dem Management würden sich viel eher dafür interessieren, weil sie in diesem Fall statistische Eigenschaften entdecken können. (s. [ML13, S.15])

„**Online Analytical Processing (OLAP)** ist eine Vorgehensweise, die schnelle und flexible Ad-Hoc-Analysen von multidimensionalen Daten für einen breiten Nutzerkreis ermöglicht“ (s. [ML13, S.50])

Im Jahre 1995 werden fünf Anforderungen von Pendse und Greeth vorgestellt, die ein OLAP System erfüllen soll. Diese fünf Anforderungen wurden mit dem Schlüsselwort **FASMI** (engl. Fast Analysis of Shared Multidimensional Information) benannt. Aus diesen Anfoderungen ableiten muss ein OLAP-System eine gewisse Geschwindigkeit bieten, intuitive Auswertungsmöglichkeiten durchführen, gemeinsame Datennutzung für Mehrbenutzer ermöglichen, die charakteristische Multidimensionalität aufweisen und mit großen Datenmengen gut skalieren können. (vgl. [ML13, S.51])

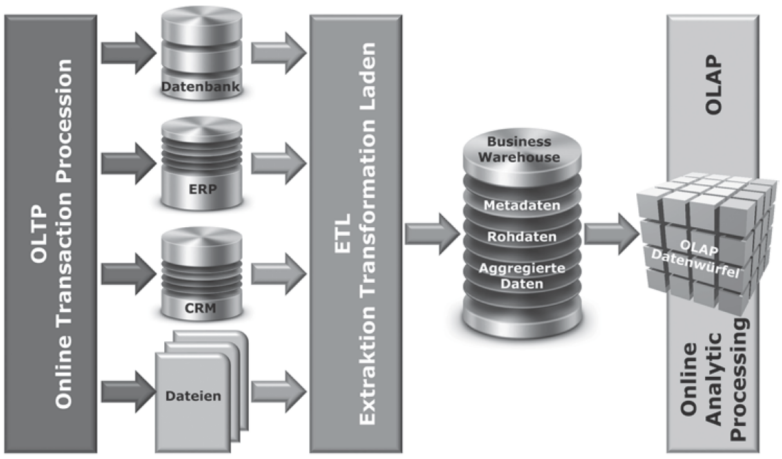


Abbildung 2.3: Business Intelligence OLTP-ETL-OLAP (s. [Gro15, S. 15])

3 Markt für Business Intelligence

Der Markt für Business Intelligence und Analytics Plattformen durchläuft derzeit eine grundlegende Veränderung. Die BI-Investments der letzten zehn Jahre zeigen, wie sehr Unternehmen den Wert auf IT-Projekte mit einem großen Anteil der Reporting-Komponente legen. Während der Bedarf an Business Intelligence in Unternehmen weiterhin ansteigt, entwickeln Unternehmen parallel nutzerspezifische Anforderungen, um den Zugang und Umgang von BI-Anwendungen für jeden Anwender, dezentralisiert und schnell, für den Aufbau von Analysen zur Verfügung zu stellen. (vgl. [Rit15])

In diesem Kapitel wird in Abschnitt 3.1 der eine Forschungsmethode von Garnter und im weiteren Verlauf im Abschnitt 3 BI-Marktführer mit ihren Herausforderungen im Wettbewerb vorgestellt.

3.1 Gartner Magic Quadranten

Der Magic Quadrant von Gartner ist eine Methode, um Unternehmen in einem spezifischen Technologie-Markt zu untersuchen und Bewertungen über diese zu visualisieren. Diese Forschungsmethode bietet sich für Unternehmen als Grundlage an, Entscheidungen über bestimmte unternehmensspezifische Bedürfnisse fällen zu können.

Der Magic Quadrant wird in vier Quadranten aufgeteilt. Die Platzierung der Quadranten können aus der Abbildung 3.1 entnommen werden. Der Magic Quadrant basiert auf zwei Achsen. Auf der X-Achse wird gemessen, wie die finanzwirtschaftliche Lage des Softwareherstellers ist, wie die Reaktion auf Marktänderungen sind und welche Vertriebskanäle und welchen Kundenstamm dieser hat. Die Y-Achse bestimmt die Vollständigkeit der Vision. Dazu zählen vor allem Aspekte der Innovation, Marktstrategie und Entwicklungsvorhersagen über den Markt.

Im Folgenden wird kurz beschrieben welche Bedeutung die einzelnen Quadranten haben (vgl. [Jen08]).

Niche Players: Die Unternehmen in diesem Quadranten konzentrieren sich auf ihren Marktsegment und weisen somit eine geringe Vision gegenüber anderen Unternehmen auf. Hierbei handelt es sich um Unternehmen, die entweder neu

am Markt sind oder sich vor allem auf Funktionalität und eine bestimmte Region konzentrieren.

Visionäre: Unternehmen aus diesem Quadranten verfolgen die Entwicklungen am Markt und versuchen mit Innovationen den Markt gerecht zu werden. Sie besitzen eine Vision, sind allerdings nicht in der Lage diese vollständig umzusetzen.

Challengers: Die Challengers sind Unternehmen, die auch die Fähigkeit dazu haben ihre Visionen umzusetzen. Es fehlt ihnen nur an der Stärke der Vision, um in den Leader-Quadranten zu wechseln. Derartige Unternehmen bewegen sich meistens in reifen Märkten.

Leader: In der Regel gehören zu diesem Quadranten große Unternehmen mit einem großen Kundenstamm, die sich in reifen Märkten bewegen. Sie zeichnen sich dadurch aus, dass sie sowohl eine große Vision, als auch die Fähigkeit dazu haben diese umzusetzen. Durch ihre hohe Anziehungskraft können Leader ihren Marktbereich in eine neue Richtung lenken.

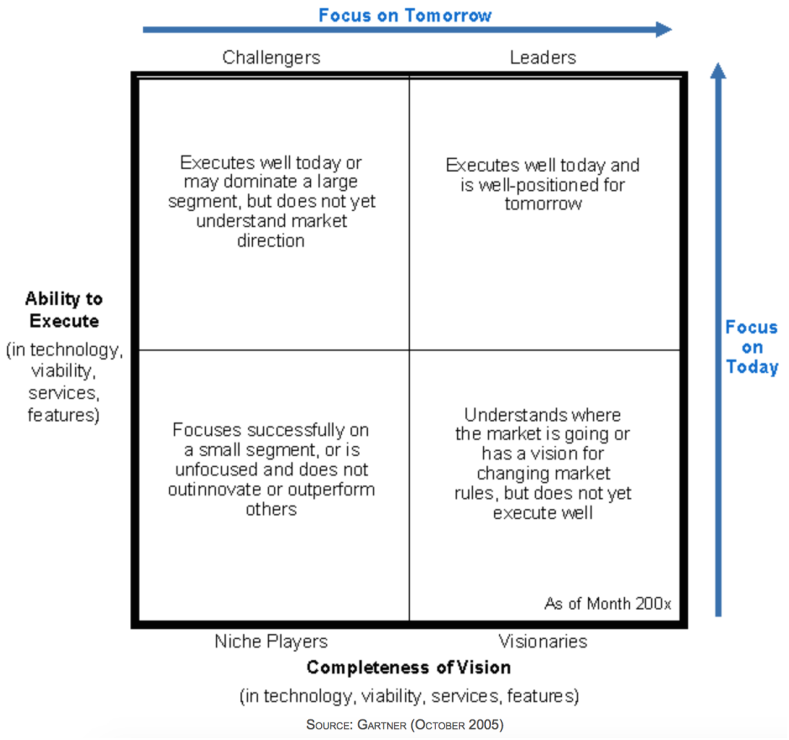


Abbildung 3.1: Gartner Magic Quadrant (s. [Jen08])

3.2 BI-Marktführer

In Abschnitt 3.1 wurde die Forschungsmethode von Gartner theoretisch vorgestellt. In diesem Abschnitt geht es nun um die Ergebnisse der Forschungsmethode

vom Februar 2015. Abbildung 3.2 bildet Unternehmen in den verschiedenen Quadranten ab. In den weiteren Abschnitten werden die Unternehmen SAP (Abschnitt 3.2.1), IBM (Abschnitt 3.2.2) und Microsoft (Abschnitt 3.2.3) in ihren Stärken und Schwächen näher betrachtet.

Abbildung 3.2: Magic Quadrant für BI und Analytics Plattformen (s. [Rit15])

3.2.1 SAP

SAP liefert eine breite Palette an Produkten zur BI und Analytics. Für große IT-unterstützte BI-Einführungen wird SAP BusinessObjects BI-Plattform angeboten und für dezentralisierte Datenerhebungen SAP Lumira bereitgestellt. Darüber hinaus dient zu den bereits genannten Produkten als Ergänzung die SAP Hana mit der In-Memory-Technologie. Unternehmen entscheiden sich häufig für SAP, um unternehmensweit einen BI-Standard einzuführen.

Die Führungseigenschaft von SAP ist in erster Linie auf zwei Aspekte zurückzuführen. SAP investiert in eine zukunftsorientierte Produktausrichtung mit SAP Lumira und ein zahlreiches Produktportfolio, das sich mit dem neuen Release verbessert hat. Darüber hinaus stellte SAP eine Vereinfachungsstrategie für die Komponenten zur Nutzung der BI-Plattform vor. Allerdings erzielt SAP unterdurch-

schnittliche Resonanz von Kundenerfahrungen in fast allen Bereichen, was den Geschäftserfolg für die Zukunft eingrenzen könnte. Um weiterhin als Leader zu bleiben, muss SAP Investments in die Kundenerfahrung setzen, um in Zukunft dynamisch und kundennah agieren zu können. (vgl. [Rit15])

3.2.2 IBM

IBM bietet eine breite Palette an Tools im Bereich Business Intelligence, Performance Management und Funktionen zu fortgeschrittenen Analysen, die durch eine spezielle Service Organisation für jeden Bereich der Industrie implementiert werden können. IBM Cognos ist eine integrierte BI-Plattform mit unter anderem Funktionen zur webbasierten Ad-hoc-Abfragen, Berichterstattung und Alarm- und Warnsysteme. Weiterhin wird OLAP-Funktionalität unterstützt und die Erstellung von Dashboards sowohl webbasiert, als auch mobil ermöglicht.

Mit Watson Analytics baut IBM für die Zukunft eine Vision auf. Herausforderungen im Bereich BI entstehen dennoch, um eine positive Resonanz der Kunden zu erhalten. Ausschlaggebend für die gute Positionierung im Magic Quadranten ist die gute Vetriebsstrategie. IBM muss vor allem auf die Marktdynamik setzen, um die Führungsposition weiterhin zu behalten. (vgl. [Rit15])

3.2.3 Microsoft

Das Produktportfolio von Microsoft für BI und Analytics unterstützt eine breite Masse an zentralisierten und dezentralisierten BI-Anwendungsfällen, sowie Analytics-Verfahren zum Nutzen ihres Kundenstamms. Typischerweise setzen Organisationen SQL Server und SharePoint ein, um für IT-Entwickler das Datenmanagement und die Anforderungen für das Reporting zu unterstützen. Im Bereich der Datenaufbereitung und Analyse werden die Anforderungen durch Business-User mit Hilfe von Komponenten des Power BI gedeckt, wie zum Beispiel Excel 2013 und Office 365.

Die Führungsposition von Microsoft im Magic Quadrant ist in erster Linie durch eine starke Produktvision und ein klares Verständnis für Marktanforderungen ausgezeichnet. Zu den Anforderungen eine Plattform zu schaffen, gehören unter anderem die Datenaufzeichnung- und Aufbereitung, das Datenverständnis und die einfache Nutzung der Daten durch Business-User. Durch das Produktportfolio von Power BI hat Microsoft auf dem BI-Markt für Anregung gesorgt, jedoch noch keine ausschlaggebende Akzeptanz gewonnen. Durch das Office 365 Cloud ist derzeit eine relativ begrenzte Funktionalität gegeben und die Komplexität zur Bereitstellung entsteht für Microsoft eine Barriere, die durch neue Versuche gebrochen werden möchte. Microsoft will durch die weitere Cloud-Nutzung und dem

Ausbau der bereits vorhandenen Produkte wie Excel und SQL Server Analysis Services den Marktanteil steigern, indem der Fokus auf Vertrieb und Marketing des BI-Bereichs gelegt wird und in die Erleichterung der Einführung für Kunden von Microsoft BI investiert wird. (vgl. [Rit15])

4 Zukunftstrends auf dem BI-Markt

In diesem Kapitel werden einige Zukunftstrends im Rahmen der BI-Thematik vorgestellt und auf Unternehmen verwiesen, die einige dieser Trends im Einsatz haben. Im Speziellen geht es in Abschnitt 4.1 um Big Data und den Umgang mit dieser Thematik. Abschnitt 4.2 wird einen kurzen Einblick darüber geben, was Predictive Analytics bedeutet und mit welchen Tools Unternehmen diesen Trend mit in ihr Produktportfolio mit aufnehmen. In Abschnitt 4.3 wird mit der Thematik Cloud BI erläutert, warum Unternehmen in der heutigen Zeit ihre Lösungen in der Cloud anbieten.

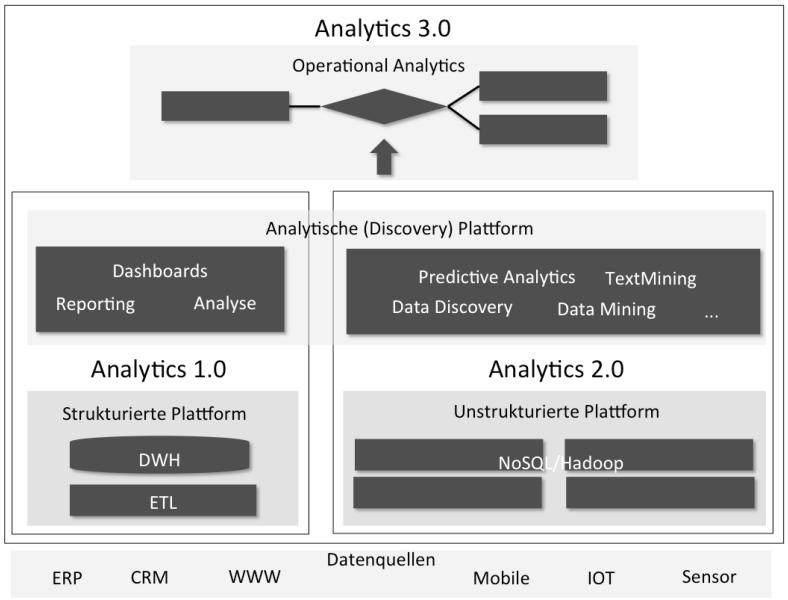


Abbildung 4.1: Analytics 3.0: Eigene Darstellung nach [SEV09])

4.1 Big Data

Im traditionellen BI werden Analysen anhand von strukturierten Daten in einem Data Warehouse für bestimmte Geschäftsszenarien durchgeführt. Dabei werden Reports für bestimmte Fakten aufgestellt oder Gründe für bestimmte Ergebnisse gesucht. Beispielsweise werden Kunden in Gruppen segmentiert und anhand von

bestimmten Paramtern oder Modelle werden diese bestimmten Zielgruppen zugeordnet (vgl. [Gro15, S. 122]). Im Zuge der Digitalisierung und des wachsenden Datenvolumens liegen Daten aus verschiedensten Datenquellen wie unter anderem Sensoren, Internet, Mobile Geräte und IT-Systeme in unstrukturierter Form vor. Big Data nutzt zur Real-Time-Analyse oder Erstellung von Prognosen Data Mining Methoden, um individuellen Ergebnissen Rückschlüsse zu bestimmten Verhaltensweisen zu ziehen (vgl. [KS15, S. 30-33]).

Zur Abgrenzung von Big Data gegenüber BI werden vier Dimensionen (Vś) zur Definition verwendet: Volume (Datenmenge), Velocity (Geschwindigkeit der Daten), Variety (Vielfalt und Komplexität der Daten), Value (ökonomischer Nutzen). (vgl.[Sub13])

Das International Institute for Analytics (IIA) beschreibt das klassische BI als Analytics 1.0 und den Bereich der Big Data Analytics als Anaylitcs 2.0. Im Rahmen dieses Konzepts wird eine Zusammenführung der beiden Analytics-Verfahren (Analytics 3.0, siehe Abbildung 4.1) vorgestellt. (vgl. [SEV09])

In Abbildung 4.2 ist deutlich zu sehen wie in den Jahren 2005 bis 2020 die digitale Datenmenge gestiegen ist. Die Abbildung zeigt uns eine Prognose zum Jahren 2020, wie die globale Datenmenge durch das Web 2.0 weiterhin stark ansteigen wird.

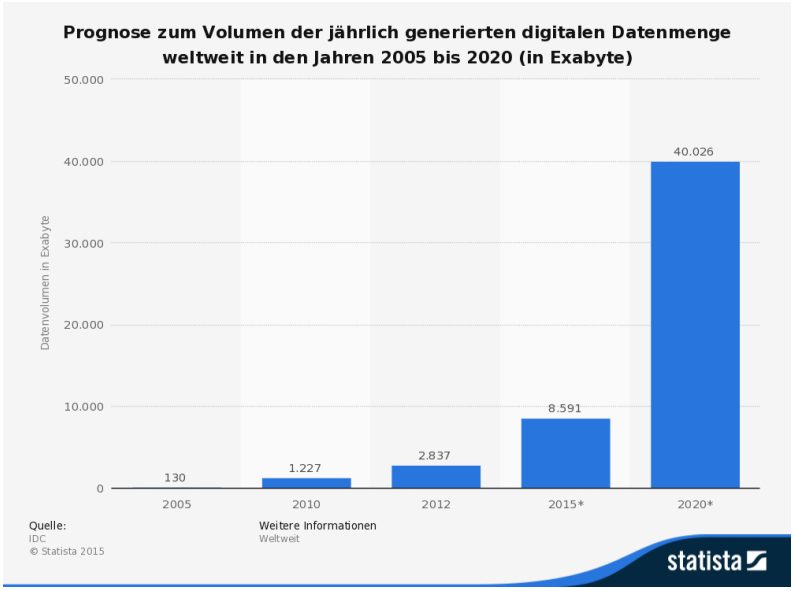


Abbildung 4.2: Prognose zum Volumen der jährlich generierten digitalen Datenmenge (vgl. [Sta15])

Proportional zu den steigenden Datenmengen wird durch das Management auch gesteigerte Anforderungen gestellt, in dem Informationsbedürfnisse umfassend, zeitnah, gezielt und ortstgebunden zu decken sind. Echtzeit-Analysen, Homoge-

nität von Daten und explorative Datenanalyse gehören unter anderem zu den Anforderungen des Managements, um intelligentes Wissen aus den riesigen Datenmengen zu generieren. (vgl. [ML13, S. 2])

Das Themengebiet des Big Data ist für viele Unternehmen aus verschiedensten Branchen sehr interessant. Wie der Abbildung 4.2 zu entnehmen ist, als auch dem Artikel von Chen, Chiang und Storey, handelt es sich um Terrabytes, wenn nicht sogar Exabytes, die meistens durch webbasierten- und mobilen Daten oder auch Sensordaten generiert werden. Im Zuge der Digitalisierung werden in staatlichen Organisationen wie zum Beispiel der Regierung oder auch im E-Commerce bis hin zur Gesundheitsorganisationen immer mehr Daten generiert, die Bedarf an Analysen aufweisen. (vgl. [CCS12, S. 1168])

Zwischen 2012 und 2014 wurden 90% der gesamten generierten Daten auf der Welt gemessen. Diese stammen überwiegend aus unstrukturierten Datenquellen. Durch MapReduce und Hadoop wurden Verfahren entwickelt, um große Datenmengen unter Verwendung von Clustern parallel verarbeiten zu können und aus unstrukturierten bzw. semi-strukturierten Daten strukturierte Daten zur Analyse aufbereiten zu können. (vgl. [Gro15, S. 126])

4.2 Predictive Analytics

Predictive Analytics ist ein zukunftsorientiertes Verfahren, um anhand von bestimmten Algorithmen und Modellen Vorhersagen über beispielsweise Marktentwicklungen oder über bestimmte Kennzahlen basierend auf Geschäftsprozessen treffen zu können. Dabei werden statistische Methoden, Modelle, Data Mining und Machine Learning verwendet, um sowohl aus aktuellen als auch aus historischen Daten Analysen über zukünftige Verhaltensweisen zu erstellen. (vgl. [Jef13])

Unternehmen wie IBM und SAP setzen stark auf die Themen Predictive Analytics und Branchenlösungen. Durch SPSS (Statistical Package for the Social Sciences) bietet IBM in diesem Gebiet Prognosefähigkeit an. Bewegungsdaten unternehmensweiter Standardsoftware wurden als Grundlage genutzt, um bestimmte Sachverhalte abzufragen. Im Bereich Business Intelligence hat SAP das SAP NetWeaver Business Warehouse (BW) im Einsatz. (vgl. [o.V10])

Mit SAP Hana bietet SAP Werkzeuge für Data Mining und Predictive Analytics. Die Erkenntnisse der Analysen aus diesen Gebieten können direkt in Prozesse integriert werden, sodass eine automatische Entscheidungsfindung der Systeme in Echtzeit stattfinden kann. Durch den Zukauf von KXEN bietet SAP unter dem Namen SAP InfiniteInsight eine separate Komponente an, um Anwendungsfälle für Predictive Analytics zu analysieren. (vgl. [Dil14])

Microsoft hingegen bietet im Bereich Predictive Analytics Azure Machine Learning an. Azure Machine Learning ermöglicht den Anwendern Erkenntnisse aus Daten mit Predictive Analytics zu gewinnen. Vorhersagemodelle können mit dem Machine Learn Studio aufgebaut und getestet werden. (vgl. [BFT15, S. 21])

4.3 Cloud BI

Für eine vollständige Nutzung der BI-Funktionen, muss BI flexibel sein und eine gemeinsame Nutzung für Mitarbeiter und Organisationsstrukturen schaffen. Vor allem ist aber auch die Qualität der Daten eine wichtige Grundlage für gute Ergebnisse. Dies ist vor allem im Bereich der Predictive Analytics wichtig, um genaue Prognosen erstellen zu können. Mit dem Wachstum eines Unternehmens wächst auch gleichzeitig die Menge und Komplexität der Daten, die verarbeitet werden müssen. Dabei ist die Aktualität der Software-Technologie ein wichtiger Grundbaustein, um Hindernisse zu vermeiden. Die Integration von BI in eine Cloud-Umgebung löst das Problem, dass veraltete Technologien genutzt werden. Darüber hinaus löst es das Problem der ständig zu erweiternden physischen Speicherkapazitäten, was ein wesentlichen Kostenvorteil mit sich bringt. Des Weiteren kann durch die Cloud-Nutzung Business Intelligence mehreren Personen und Organisationen zugängig gemacht werden, wobei Daten und Analysen auf einem Web-Browser abrufbar sind. (vgl. [Tol14, S. 54-56])

Im Zuge der Digitalisierung entwickeln sich Geschäftsprozesse von Unternehmen immer mehr online, um dadurch auch weiterhin die Kundennähe zu erhalten. Laut einer Studie des Enterprise Managment Associates über Anaylitcs in der Cloud vom Januar 2015, wird BI als primäre Geschäftsmotivation für die Wahl einer Cloud-Option gesehen. Minimierte Hardwarekosten, die Benutzerfreundlichkeit, minimierte Implementierungskosten, Zugang zu mehreren Akteuren und verbesserte Mobilität sind auch für Unternehmen wie SAP, IBM und Microsoft wichtige Faktoren für die Einführung von Cloud-Lösungen. Zur Datenauswertung und -Visualisierungen werden in der Cloud SAP Lumira Cloud, IBM Watson Analytics und Microsoft Power BI bereitgestellt. (vgl.[Dou15])

Auch wenn Cloud BI für viele Unternehmen einen Nutzen bringt, ist es nicht ausgeschlossen, dass Probleme wie Datensicherheit in ernste Betrachtung gezogen werden. BI in die Cloud zu übertragen bedeuetet gleichzeitig hochsensible Kundendaten ins Internet zu übertragen. Ein weiteres Problem ist die Bandbreite, die vor allem bei großen Datenübertragungen für bestimmte Analysen nicht ausreichend sein könnte. BI-Analysen brauchen viele Daten, um komplexe Analysen durchführen zu können. Diese müssen zum Cloud-Provider transferiert werden und könnte bei geringen Bandbreiten zu Problemen führen. (vgl.[Kla13])

5 Zusammenfassung und Ausblick

Im Rahmen dieser Seminararbeit wurde ein Grundverständnis für BI erarbeitet, indem zunächst der historische Hintergrund und anschließend eine Definition und ein Verständnis für den BI-Begriff erläutert wurde. Grundlegend für BI wurde das Data-Warehouse-System, der ETL-Prozess, OLTP und OLAP vorgestellt.

Des Weiteren wurde eine Marktstudie für BI vorgestellt, in dem Software-Hersteller nach bestimmten Kategorien bewertet und als Ergebnis in einer Matrix aufgestellt wurden. Hierbei wurden die Software-Hersteller in vier Quadranten aufgeteilt: Niche Players, Visionäre, Challengers und Leader. Basierend auf dieser Marktanalyse wurden drei BI-Marktführer, SAP, IBM und Microsoft, mit ihren BI-Lösungen und ihren zukünftigen strategischen Ausrichtungen kurz vorgestellt.

Darüber hinaus wurde im Rahmen dieser Seminararbeit Zukunftstrends, die dem BI-Markt eine gewisse Richtung beisteuern, darlgelegt. Unter anderem wurden die Themen Big Data, Predictive Analytics und Cloud BI aufgegriffen und die Stellung der Unternehmen zu den jeweiligen Themengebieten erörtert.

Zusammenfassend ist zu sagen, dass Business Intelligence einer der wichtigsten Wettbewerbsfaktoren für Unternehmen geworden ist. Sei es die Analyse von strukturieren, als auch unstrukturierten oder semi-strukturierten Daten. Jegliche Ausprägung von Business Intelligence, ob Analytics 1.0, Analytics 2.0 oder die Zusammenführung der beiden Blöcke. Business Intelligence ist und bleibt ein wichtiger Bestandteil sowohl für Analysen zur Optimierung von Geschäftsprozessen, als auch zur Erstellung von Prognosen für zukünftige Entscheidungen. Es wurden zwar drei Zukunftstrends vorgestellt, jedoch gibt es vielmehr an Trends (Mobile BI, Social Intelligence, etc.), die dafür sorgen, dass BI-Software-Hersteller den Wettbewerb am Markt weiterhin wahrnehmen. Für die Zukunft ist zu sagen, dass der BI-Markt weiterhin dynamisch bleibt und im weiteren Verlauf des Technologie- und Internetzeitalters auch in der BI-Branche weitere spannende Themenfelder entstehen werden.

Literatur

[BFT15] BARGA, Roger ; FONTAMA, Valentine ; TOK, Wee-Hyong: *Predictive Analytics with Microsoft Azure Machine Learning: Build and Deploy Actionable Solutions in Minutes*. Berkeley C.A. : Apress, 2015

[BG13] BAUER, Andreas ; GUNZEL, Holger: *Data-Warehouse-Systeme. Architektur, Entwicklung, Anwendung.* Heidelberg : dpunkt.verlag, 2013

[CCS12] CHEN, Hsinchun ; CHIANG, Roger H. L. ; STOREY, Veda C.: Business intelligence and analytics: from big data to big impact. In: *MIS Quarterly* 36 (2012), Nr. 4

[Dav13] DAVENPORT, Thomas H.: The rise of analytics 3.0. In: *International Institute for Analytics* 14 (2013)

[Dil14] DILL, Markus: Bei Predictive Analytics setzt SAP HANA zur Aufholjagd an. In: *www.isreport.de* 8 (2014)

[Dou15] DOUG HENSCHEN: *10 Cloud Analytics & BI Platforms For Business*. `http://www.informationweek.com/cloud/software-as-a-service/10-cloud-analytics-and-bi-platforms-for-business/d/d-id/1318724`, 2015. – Accessed on November 26, 2015.

[GRC09] GOMEZ, Jorge M. ; RAUTENSTRAUCH, Claus ; CISSEK, Peter: *Einfuhrung in Business Intelligence mit SAP NetWeaver 7.0*. Berlin, Heidelberg : Springer Vieweg, 2009

[Gro15] GRONWOLD, Klaus-Dieter: *Integrierte Business-Informationssysteme: ERP, SCM, CRM, BI, Big Data Analytics - Prozesssimulation, Rollenspiel, Serious Gaming*. Berlin : Springer Vieweg, 2015

[Hah05] HAHNE, Michael: *SAP Business Information Warehouse. Mehrdimensionale Datenmodellierung*. Berlin, Heidelberg : Springer-Verlag, 2005

[Inm05] INMON, William H.: *Building the Data Warehouse. 4. Auflage.* Indianapolis : John Wiley & Sons, 2005

[Jef13] JEFF BERTOLUCCI: *Big Data Analytics: Descriptive Vs. Predictive Vs. Prescriptive*. http://www.informationweek.com/big-data/big-data-analytics/big-data-analytics-descriptive-vs-predictive-vs-prescriptive/d/d-id/1113279, 2013. – Accessed on November 26, 2015.

[Jen08] JENNI LEHMAN: *Magic Quadrants and MarketScopes: How Gartner Evaluates Vendors Within a Market*. http://www.gartner.com/doc/486094/magic-quadrants-marketscopes-gartner-evaluates#a2016506220, January 2008. – Accessed on November 15, 2015.

[KBM10] KEMPER, Hans-Georg ; BAARS, Henning ; MEHANNA, Walid: *Business Intelligence - Grundlagen und praktische Anwendungen: Eine Einfuhrung in die IT-basierte Managementunterstutzung*. Berlin, Heidelberg : Vieweg & Teubner, 2010

[Kla13] KLAUS MANHART: *Datenanalyse als Cloud-Service*. http://www.tecchannel.de/software/bi/2043874/datenanalyse_als_cloud_service/index2.html, 2013. – Accessed on November 27, 2015.

[KS15] KURZE, Dr. C. ; SCHOPP, Michael: BI-Spektrum, Fachzeitschrift fur Business Intelligence. In: *BI-Spektrum* 10 (2015), Nr. 4

[ML13] MULLER, Roland M. ; LENZ, Hans-Joachim: *Business Intelligence*. Berlin, Heidelberg : Springer Vieweg, 2013

[o.V10] O.V.: Business Intelligence verlagert den Fokus von der Analyse auf Prognosen. In: *Business Intelligence* 14 (2010), Juni

[Rit15] RITA L. SALLAM AND BILL HOSTMANN AND KURT SCHLEGEL AND JOAO TAPADINHAS AND JOSH PARENTEAU AND THOMAS W. OSTREICH: *Magic Quadrant for Business Intelligence and Analytics Platforms*. http://www.gartner.com/technology/reprints.do?id=1-2AH4Q85&ct=150224&st=sb, May 2015. – Accessed on November 14, 2015.

[SEV09] SEVEN PRINCIPLES AG: *BUSINESS INTELLIGENCE & BIG DATA*. http://www.7p-group.com/leistungen/business-intelligence-big-data/, 2009. – Accessed on November 19, 2015.

[Sta15] STATISTICA: *Prognose zum Volumen der jahrlich generierten digitalen Datenmenge weltweit in den Jahren 2005 bis 2020 (in Exabyte)*. http://de.statista.com/statistik/daten/studie/267974/umfrage/prognose-zum-weltweit-generierten-datenvolumen/, 2015. – Accessed on November 03, 2015.

[Sub13] SUBU SANGAMESWAR: *Big Data - An introduction*. http://www.amazon.com/dp/B00AZCK0Y4?tag=kiq-free-r-20, 2013. – Accessed on November 23, 2015.

[Tol14] TOLE, Alexandru A.: Cloud Computing and Business Intelligence. In: *Database Systems Journal* 5 (2014), Nr. 4

Abschließende Erklärung

Ich versichere hiermit, dass ich meine Seminararbeit „Was ist Business Intelligence?“ selbständig und ohne fremde Hilfe angefertigt habe, und dass ich alle von anderen Autoren wörtlich übernommenen Stellen wie auch die sich an die Gedankengänge anderer Autoren eng anlegenden Ausführungen meiner Arbeit besonders gekennzeichnet und die Quellen zitiert habe.

Oldenburg, den 26. September 2016 Muhammed Altuntas

Apache Hive

Seminararbeit

Themenersteller:	Prof. Dr.-Ing. habil. Jorge Marx Gómez
Betreuer:	M. Eng. & Tech. Viktor Dmitriyev Dr.-Ing. Andreas Solsbach M.Sc. Ola Mustafa
Vorgelegt von:	Kevin Aland kevin.aland@uni-oldenburg.de
Abgabetermin:	2. Dezember 2015

Inhaltsverzeichnis

Abkürzungen

HDFS Hadoop Distributed File System

SQL Structured Query Language

UDFs User-Defined Functions()

HiveQL Hive Query Language

Abbildungen

Tabellen

1 Einleitung

In dieser Seminararbeit soll untersucht werden, welche Funktionalitäten Hive anbietet um das in Java geschriebene Framework Apache Hadoop zu erweitern.
Dazu wird untersucht, was die grundlegenden Eigenschaften von Hive sind, wo sich Hive in der Systemlandschaft von Apache Hadoop befindet, welche Systemarchitektur vorliegt und wie die Abfragesprache HiveQL funktioniert. Abschließend wird ein Fazit verfasst.

2 Hive, eine Erweiterung für Apache Hadoop

2.1 Was ist Hive?

Hive ist eine Data Warehouse Software, die die Abfrage und Verwaltung von großen Datenmenge in verteilten Speichersystemen erlaubt.
Dazu verwendet es eine SQL ähnliche Sprache namens HiveQL. Diese Sprache übersetzt SQL-artige Abfragen in MapReduce-Jobs, die dann wiederum in Hadoop ausgeführt werden. Es erlaubt ebenfalls traditionelle MapReduce - Programmierer ihre spezifizierten MapReduce-Skripte zu verwenden, falls es zu schwierig sein sollte, diese Logik in HiveQL auszudrücken.
In Hive wird nicht vorgeschrieben, dass gelesen oder geschriebene Daten in einem Hive Format sein muss. Für OLTP-Workloads und Echtzeit-Abfragen ist Hive nicht geeignet. Es ist am besten für die Verarbeitung von großen Datenmengen in einer nicht echtzeitnahen Analyse geeignet.
Hive ist ein Open Source Projekt unter der Führung von der Apache Software Foundation. Früher war Hive ein Teilprojekt von Apache Hadoop. Zurzeit ist es aber ein eigenständiges Top-Level Projekt. Der ursprüngliche Entwickler von Hive war Facebook, der das Projekt der Open-Source-Gemeinde zur Verfügung stellte. In dem Unterkapitel *Der Entwickler von Apache Hive* wird näher darauf eingegangen [Noma].

2.1.1 Data Warehouse

Wie bereits erwähnt ist Hive eine Data Warehouse Software. Aber was genau ist Data Warehouse? Wofür wird es eingesetzt und welche Kriterien muss eine Data Warehouse Software erfüllen. In diesem Kapitel werden diese Frage näher erläutert.
Ein Data Warehouse integriert Informationen aus vielen unterschiedlichen Quellen in einer für die Entscheidungsfindung optimierte Datenbank. W.H. (Bill) Immon gilt als der Vater des Konzepts des Data Warehouse. Er definiert und charakterisiert ein Data Warehouse als „themenorientierte, integrierte, zeitbezogene und nicht-flüchtige Datenbank zur Unterstützung von Managemententscheidungen“.

Themenorientiert: Themenorientiert bedeutet, dass nicht einzelne Aspekte wie z.B. Transaktionen von Bedeutung sind, sondern Kennzahlen wie der Umsatz eines bestimmten Geschäftsbereiches eines Unternehmens von Interesse für Managemententscheidungen sind.
Integriert: Ein Date Warehouse muss integriert sein, d.h. es muss Daten aus unterschiedlichen Quellen und Fremdsystemen in einheitliche Formate umsetzen können.
Zeitlich veränderlich: Zeitlich veränderlich bezieht sich auf die Zeitabhängigkeit der Daten. Bei operativen Systemen sind die Daten so aktuell wie im Moment der Eingabe und reichen in der Regel 30-90 Tage zurück. In einer Date Warehouse Umgebung stellen vorhandene Daten immer eine Art ‚Schnappschuss' eines bestimmten Zeitpunkts oder Raumes dar. Dabei werden die Daten periodisch aktualisiert und nicht in Echtzeit verarbeitet.
Nicht-flüchtig: Hierbei ist zu verstehen, dass Daten die Einmal im Data Warehouse eingebracht und gespeichert wurden, weder modifiziert noch entfernt werden dürfen. Dadurch kann in Laufe der Zeit eine Historisierung der extrahierten Zustände und Daten der Quellesysteme erreicht werden [Far11].
Man muss beim Themenbereich Data Warehouse zwischen drei Bezeichnungen differenzieren. Während der Begriff nur die eigentliche Datenbank bezeichnet, beschreibt ein Data-Warehouse-System die gesamte technische Infrastruktur zur Beschaffung, Speicherung und Auswertung der Daten. Dazu kommt noch der Begriff Data Warehousing, der den Fluss und die Verarbeitung der Daten aus den Datenquellen bis zum Analyseergebnis beim Anwender beschreibt. Im Nachfolgenden werden diese Schritte aufgezeigt und die Anwendung von Hive diesen Schritten zugeordnet.

1. Die Extraktion der relevanten Daten aus Quellsystemen
2. Die Transformation und Bereinigung der Daten der Quellsysteme
3. Laden der bereinigten, konsistenten Daten in das Data Warehouse
4. Dauerhafte Speicherung der Daten im Data Warehouse
5. Bereitstellung der zu Analysezwecken benötigten Datenbestände aus dem Data Warehouse
6. Auswertung und Analyse der Datenbestände [Far11]

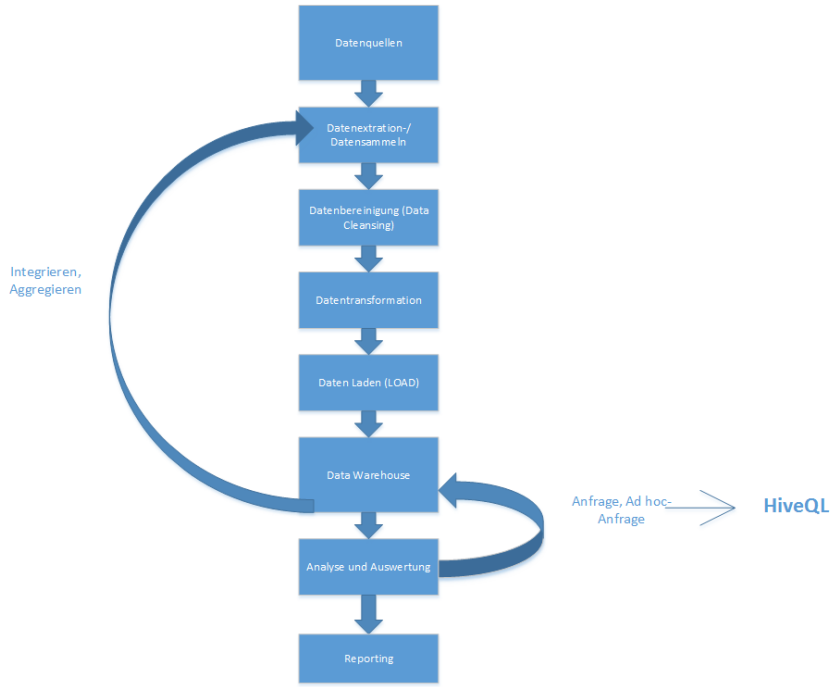


Abbildung 2.1: Data Warehouse, vgl. [KFark11, Abb. 2.1]

2.1.2 Der Entwickler von Apache Hive

Der Ursprüngliche Entwickler von Hive ist Facebook. Durch die immer mehr aufkommende Datenmenge musste eine geeignete Software entworfen werden, der die große Datenmenge analysieren kann. Die Entwicklung von Hive begann 2007 und der Open Source Release erfolgte ein Jahr später. Nach dem Open Source Release entwickelte sich Hive immer weiter. Zu der Zeit wurden 200 GB Daten am Tag erzeugt. Intern hat Facebook Hive verwendet für Berichterstattungen, Ad-hoc-Abfragen und Analysezwecke. Hauptmotivationsgrund für die Entwicklung von Hive war die Beobachtung. In Verbindung mit Hadoop, der Skalierbarkeit versprach und Map/Reduce-Verfahren konnte mit Hive auch SQL-Abfragen erzeugt werden. Ohne Hive würden Abfrageaufträge immens länger dauern [Thu].

2.1.3 Hive in der Systemlandschaft von Apache Hadoop

Hadoop wird im Zusammenhang mit Hive eingesetzt. Daher ist es wichtig zu wissen, was Hadoop überhaupt ist und wie bzw. wo genau sich Hive in der Systemlandschaft von Hadoop eingliedert.
Hadoop ist ein Apache Open Source Framework, dass in Java geschrieben ist und die verteilte Verarbeitung von großen Datenmengen über Clustern von Compu-

tern durchführen kann. Dabei umfasst Hadoop zahlreiche Teilprojekte, eines davon ist Hive. Die Hauptarchitektur besteht aus zwei Hauptkomponenten.

- Das Hadoop Distributed File System (HDFS) wird durch eine Master/Slave Architektur dargestellt. Dabei besteht ein Cluster aus einem einzelnen Master, dem NameNode. Der ist zuständig für die Verwaltung des Dateisystemes und dessen Namensraum und regelt den Zugriff von Clients. Die Slaves bilden die DataNodes, die gewöhnlich auf genau einem Knoten operieren. Es können aber auch mehrere auf einem Knoten arbeiten. DataNodes verwalten den am Knoten zur Verfügung gestellten Speicher.

- MapReduce koordiniert nebenläufige und verteilte Berechnungen auf den riesigen Datenmengen, die üblicherweise im Petabyte-Bereich stattfinden. In Hadoop können Clients Berechnungen mittels Jobs durchführen, ein Job wird dabei in Tasks unterteilt (Map-Tasks und Reduce-Tasks). Für die Ausführung eines Jobs sind wiederum zwei verschiedene Knoten zuständig (Jobtracker und Tasktracker). Der Jobtracker koordiniert alle Jobs und die Tasktracker sind für die Ausführung verantwortlich [JFrey12].

Abbildung 2.2 zeigt das Zusammenspiel der Hauptkomponenten von Hadoop mit Hive. Hier wird noch nicht auf die einzelnen Komponenten von Hive eingegangen. Das geschieht erst in dem nächsten Kapitel. Grundsätzlich kann gesagt werden, dass sich Hive auf der Oberseite von Hadoop befindet. Die Daten werden ins HDFS abgelegt und mit Hive/MapReduce können diese Daten mit klassischen Abfragen analysiert werden.

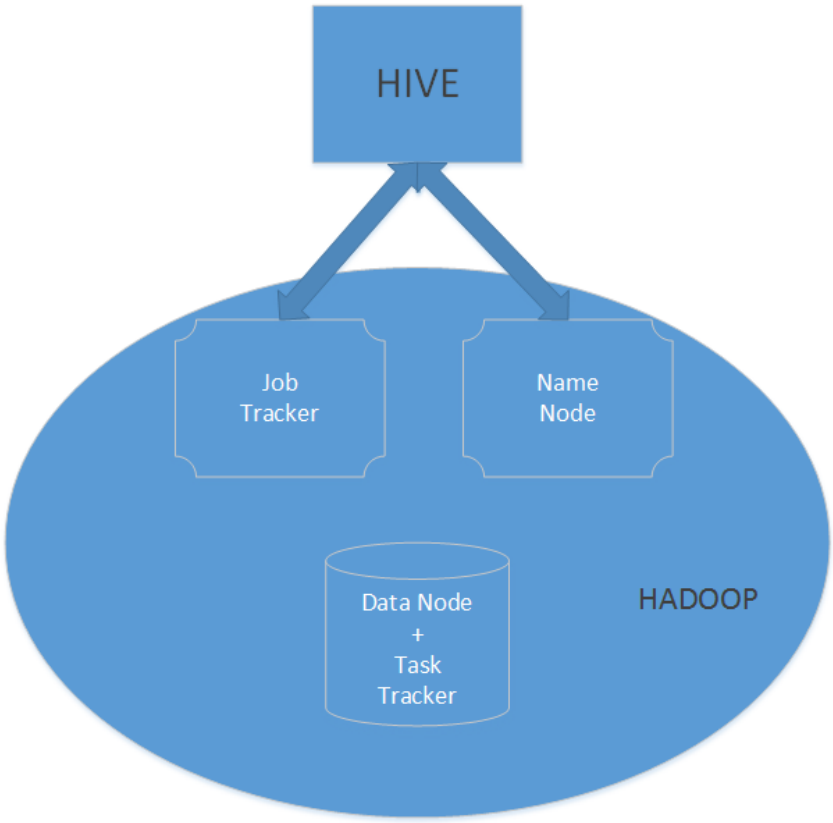


Abbildung 2.2: Hadoop - Hive Systemlandschaft, vgl. [AThu, Abb. 1]

3 Systemarchitektur und Komponenten

In diesem Kapitel werden die einzelnen Komponenten von Hive vorgestellt und deren Zusammenspiel erläutert.

User Interface: Mit der User Interface kann eine Interaktion zwischen User und dem HDFS hergestellt werden. Hive unterstützt Hive Web UI, Hive Command Line und Hive HD Insight (Windows Server)

Metastore: Die Komponente ist zuständig für die Speicherung des Systemkatalogs und deren Metadaten, die Informationen über Tabellen, Spalten, Partitionen, Datentypen und das HDFS mapping enthalten.

Driver: Der Driver regelt den Lifecycle einer HiveQL Abfrage. Er enthält ebenfalls Informationen über eine Session und sämtlichen Statistiken.

HiveQL Process Engine: Mit dieser Komponente wird der traditionelle Ansatz von MapReduce aufgebrochen. Anstelle ein MapReduce Programm in Java zu entwickeln, kann eine einfache Query für einen MapReduce Job aufgestellt werden.

Execution Engine: Die Execution Engine ist die Verbindung zwischen der HiveQL Process Engine und MapReduce. Sie verarbeitet die Query und generiert ein Ergebnis das genauso ist wie ein MapReduce Ergebnis.

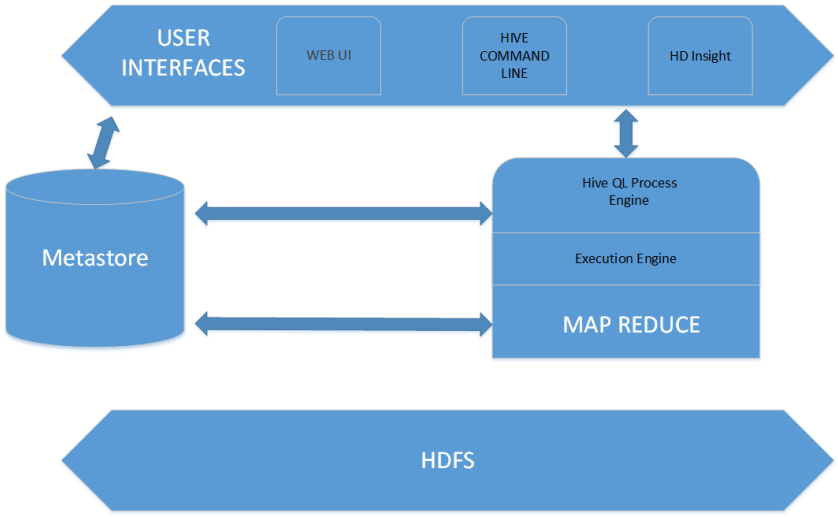


Abbildung 3.1: Hive Komponenten, vgl. [TPoint, Abb. 1]

Die Grundlegenden Komponten von Hive und Hadoop wurden dargestellt. Jetzt fehlt noch die Interaktion der einzelnen Komponenten untereinander. In der nachfolgenden Grafik wird diese Interaktion in einem durchnummerierten Ablauf dargestellt.

1. Ausführen der Query: In dem ersten Schritt beginnt der Hive Ablauf. Das Hive Interface, je nachdem ob eine Kommandozeile oder eine Webbenutzeroberfläche verwendet wird, sendet eine Query an dem Driver damit diese Ausgeführt werden kann.

2. Holt Hilfe: Im zweiten Schritt holt sich der Driver die Hilfe vom Query Compiler. Der Compiler parst die Query, um zu überprüfen ob die Syntax korrekt ist. Ebenso wird überprüft, ob der Query Plan oder die Anforderungen der Query in Ordnung sind.

3. Metadaten anfordern: In diesem Schritt sendet der Compiler eine Anfrage an dem Meta Store, was irgendeine Datenbank sein kann.

4. Metadaten liefern: In dem nächsten Schritt antwortet der Meta Store auf die Anfrage des Compilers und sendet die benötigten Metadaten.

5. Query überprüfen: Im fünften Schritt werden die Anforderungen mit den Metadaten vom Compiler überprüft. Sollte alles korrekt sein, ist das Parsen und Kompilieren der Query abgeschlossen.

6. Query ausführen: Der Driver sendet den Ausführungsbefehl an die Execution Engine.

7. Ausführungsbefehl: Dieser Schritt verbindet Hive mit Hadoop. Der Prozess des Ausführungsbefehl ist eine MapReduce Aufgabe. Dabei sendet die Execution Engine den Auftrag an den JobTracker, der wiederum im NameNode enthalten ist und weißt den Auftrag einem TaskTracker zu, der im DataNode enthalten ist. Also wird in diesem wichtigen Schritt der MapReduce-Job mit einem SQL ähnlichen Query ausgeführt.

7.1 Metadaten operationen: Während der Ausführung führt die Execution Engine Metadaten-Operationen mit dem Meta Store durch.

8. Ergebnis liefern: In Schritt acht empfängt die Execution Engine die Ergebnisse von den DataNodes.

9. Ergebnisse übermitteln: Im vorletzten Schritt sendet die Execution Engine die angeforderten relevanten Daten dem Driver.

10. Ergebnisse dem Interface bereitstellen: Im letzten Schritt sendet der Driver die Ergebnisse dem Hive Interface.

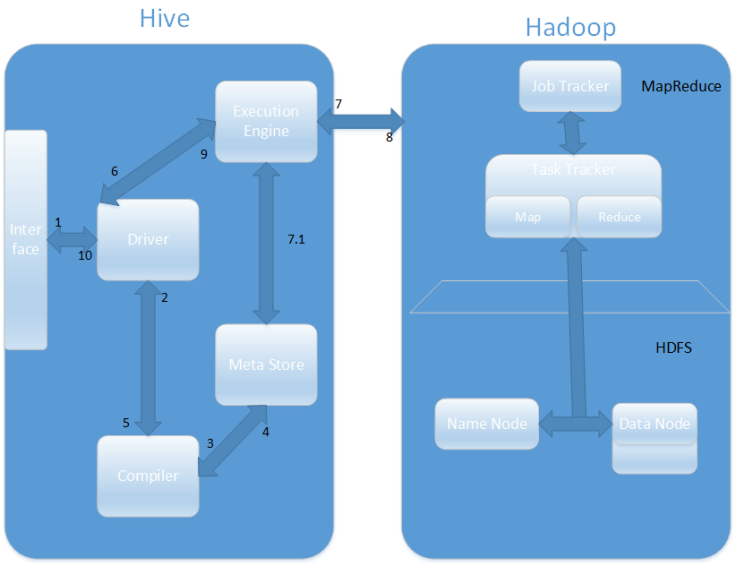


Abbildung 3.2: Hive Komponenten, vgl. [TPoint, Abb.2]

4 HiveQL

4.1 Hive Datentypen

Hive besitzt drei unterschiedliche Datentypen, die in der Tabellenerzeugung involviert sind.

- Primitive Datentypen
- Null Values
- Complex Types

4.1.1 Primitive Datentypen

Hive unterstützt viele Größen von Integer und Floating-Point Typen(Float und Double), Boolean Type und Strings. In Hive Version 0.8.0 sind zwei neue Typen dazugekommen, Timestamp und Binary.
Der Timestamp kann als Integer, Float oder String dargestellt werden. Als Integer können ganzzahlige Zeitangaben angeben werden, wie z.B. Dezember 1, 2015 und als Float können Zeitangaben im Nanosekunden Bereich ausgegeben werden(Bis zu neun Nachkommastellen). Für das String Format wird die JDBC Data String Format Konvention verwendet, YYYY-MM-DD hh:mm:ss.fffffffff.
Der Binary Type verhält sich ähnlich zu dem VARBinary Type, der in anderen Relationalen Datenbanken Anwendung findet. Dadurch kann verhindert werden, das Hive die Daten als Nummern oder Strings parst. Falls unterschiedliche Datentypen verglichen werden sollen castet Hive implizit Datentypen zu höher gelegenen Datentypen (z.B. Integer zu Double).

Fehlende Werte werden mit dem Spezialwert NULL ausgedrückt[Rut12] [Nomb].

4.1.2 Complex Types

Bei den Komplexen Datentypen unterstützt Hive Arrays, Maps und Structs. Um eine strukturierte Datenabfolge auszudrücken kann der Typ struct genutzt werden. Eine Adresse wird immer in der gleichen Reihenfolge festgelegt Straße, Stadt,

Bundesland und Postleitzahl. Mit strut kann diese Reihenfolge beispielsweise so ausgedrückt werden **struct<street:STRING, city:STRING, state:STRING, zip:INT>**. Beim Datentyp Map kann eine Sammlung von Key-Value Paaren angeben werden [Rut12][Nomb].

4.2 Database Handling

Hive ist eine Datenbanktechnologie mit der Datenbanken und Tabellen erstellt werden, um dann strukturierte Daten zu analysieren. Die strukturierten Daten werden in einer zuvor angelegten Tabelle gespeichert und darauf können dann SQL-Abfragen durchgeführt werden, um die Daten zu analysieren.
Um eine Datenbank anzulegen muss der folgenden Ausdruck verwendet werden.

SQL 4.1: Create Table

```
1 hive > CREATE DATABASE [IF NOT EXISTS] userdb;
```

Als nächstes müssen Tabellen generiert werden. Die zuvor erwähnten Datentypen können dabei genutzt werden.

SQL 4.2: Create Table mit Parametern

```
1 hive > CREATE TABLE IF NOT EXISTS employee ( eid int, name String,
2   salary String, destination String)
```

Nach der Deklaration der Spaltennamen können noch optionale Angaben gemacht werden, wie z.B. die Pfadangabe zu der eben generierten Datenbank. Sollte **IF NOT EXISTS** nicht angeben werden und die Tabelle existiert schon, dann ignoriert Hive die Angabe und erstellt die Tabelle trotzdem. Sollte die Generierung der Tabelle erfolgreich gewesen sein, wird ein Feedback, wie lange die Generierung gedauert hat, zurückgegeben.
Nachdem in SQL eine Tabelle erstellt wurde, können die Daten mit dem **INSERT** Statement eingefügt werden. In Hive dagegen werden Daten mit dem **LOAD DATA** Statement eingefügt.

SQL 4.3: Load Data in Hive

```
1 hive > LOAD DATA LOCAL INPATH '/home/kevin/test.txt'
2       OVERWRITE INTO TABLE employee;
```

Mit dem Befehl **LOAD DATA INPATH** wird angegeben, wo sich die einzufügende Datei befindet. **INTO TABLE** gibt an, in welche Tabelle die Daten aus der Datei platziert werden soll. **LOCAL** und **OVERWRITE** sind Optionale Befehle.
Darüber hinaus können Datanbanken und Tabellen gedroppt werden. Wenn eine Tabelle vom Metastore gedroppt werden muss, dann werden alle Daten aus der Tabelle und seine Metadaten gelöscht. Der Befehl lautet wie folgt.

SQL 4.4: Drop Table in Hive

```
1 hive > DROP TABLE employee;
```

Hive bietet auch die Möglichkeit Tabellen in Partitionen aufzuteilen. Dadurch können die Werte in den Tabellen nach bestimmten Kategorien aufgeteilt werden. Das hat den Zweck, dass es einfacher ist eine bestimmte Auswahl an Daten abzufragen und der Prozess terminiert schneller als ohne Partitionierung. Tabellen oder Partitionen sind wiederum in sogenannten Buckets unterteilt. Durch die Partitionierung können Suchaufwände reduziert werden. Das folgende Beispiel erklärt es anschaulich. Es sollen bestimmte Arbeitnehmer, die im Jahr 2015 eingestellt wurden, herausgesucht werden. Das kann gelöst werden indem eine Tabelle in der Beispielsweise alle Arbeitnehmereinstellungen seid Unternehmensgründung so aufgeteilt werden, dass nur die Einstellungen aus dem Jahre 2015 enthalten sind und kann darauf dann seine Analysen durchführen. Dadurch wird der Suchaufwand reduziert [Rut12][Nomb].

4.3 Funktionen, Operatoren und UDFs

In Hive sind von Haus aus schon Funktionen enthalten, aber der User kann auch eigene Funktionen in Java Implementieren und damit das HiveQL erweitern. Solche Funktionen heißen **User-Defined Functions (UDFs)**. Wenn so eine individuelle Funktion der Session hinzufügt wird, agiert die Funktionen wie eine bereits enthaltene Funktion in Hive.
Um einen Überblick über alle enthaltenen Funktionen zu bekommen, muss der folgenden Befehl verwenden werden.

SQL 4.5: Show-Funktion

```
1 hive > SHOW FUNCTION;
```

Der nächste Befehl gibt eine Beschreibung der Funktion an.

SQL 4.6: Describe-Funktion

```
1 hive > DESCRIBE FUNCTION;
```

Damit eine Funktion auf eine Tabelle angewendet werden kann, muss folgender Befehl ausgeführt werden

SQL 4.7: Funktion in Hive

```
1 SELECT "FUNCTION"(column1,column2) AS x FROM table;
```

[JRut12][TPoint].

4.3.1 Funktionen

Bei den vorhanden Funktionen sind mathematische Operationen enthalten wie z.B. das Runden einer Zahl.

SQL 4.8: Round-Funktion

```
1 hive > SELECT round(2.6) from temp;
```

Bei dieser Funktion will er die Zahl 2.6 aufrunden, dass Gegenstück dazu wäre die Funktion floor. Hier wird dann abgerundet. String Manipulation ist in Hive auch enthalten. Hier kann z.B. mit **reverse()** ein String umkehrt werden.
Ebenso können in Hive Aggregierungsfunktionen ausgeführt werden, die z.B. die Gesamtsumme der Monatsumsätze im Dezember ermittelt.

SQL 4.9: Sum-Funktion

```
1 hive > SELECT sum('Dezember') FROM Monatsumsatz;
```

[Rut12][Nomb].

4.3.2 Operator

In Hive können wie bei anderen Programmiersprachen auch Operatoren verwendet werden.

Relational Operators: Mit diesem Operator können zwei Operanten verleichen werden.

SQL 4.10: Relationaler Operator

```
1 hive > SELECT * FROM employee WHERE Salary > 50000;
```

Als Ergebnis werden die Arbeitnehmer, die ein Gehalt über 50.000 Euro erhalten, geliefert.
Arithmetic Operators: Dieser Operator kann arithmetische Aufgaben durchführen und z.B. zwei Nummern addieren.

Logical Operators: Der logische Operator kann TRUE or FALSE zurückgeben. Damit können die Ausdrücke auf ihren Wahrheitsgehalt überprüft werden und dementsprechend weitere Operationen ausgeführt werden.

Complex Operators: Mit dem letzten Operator können komplexe Typen ausgewertet werden. Beispielsweise kann mit dem Ausdruck M[key] der entsprechend Wert zum Schlüssel in der Map ausgegeben werden [Rut12][Nomb].

4.3.3 UDFs

User haben die Möglichkeit in Hive eigene Funktionen zu schreiben. Ohne dieses Verfahren müssen bestimmte MapReduce Algorithmen genommen werden oder die Aufgabe in ein anderes System verlagert werden. Dadurch müssen die Daten zwischen verschiedenen Systemen ausgetauscht werden und das kostet wiederum Zeit. UDFs hingegen arbeiten in dem gleichen Prozess, indem der Query für eine bestimmte Aufgabe ausgeführt wird. Dadurch wird effektiv gearbeitet und erspart sich die Komplexität bei der Integration unterschiedlicher Systeme [Rut12][Nomb].

4.4 HiveQL Queries

In diesem Unterkapitel werden die Statements von HiveQL vorgestellt.

Mit dem **SELECT** Statement werden die Daten einer Tabelle abgefragt. Die **WHERE** Klausel funktioniert wie eine Bedingung. Es liefert die Ergebnisse, die die angegebene Bedingung erfüllt. Mit den in Hive vorhanden Operatoren, den Attributen und den dazugehörigen Werten der Tabelle kann eine Bedingung aufgestellt werden.
Das **ORDER BY** Statement ist zum Sortieren der Tabelle zuständig. Man kann angeben, ob das Ergebnis aufsteigend oder absteigend sortiert werden soll. Das **GROUP BY** Statement ist dafür zuständig Datensätze gruppiert nach bestimmten Auswertungskriterien auflisten.
JOINS können genutzt werden, um bestimmte Felder zu kombinieren.

SQL 4.11: TEST

```
1 hive > SELECT c.ID, c.NAME, c.AGE, o.AMOUNT > FROM CUSTOMERS c
      JOIN ORDERS o > ON (c.ID = o.CUSTOMERID);
```

Als erstes wird mit dem Select ausgewählt welche spalten im Ergebnis angezeigt werden sollen. Mit dem **JOIN ORDERS** Statement werden zwei Tabellen mit einander verknüpft und mit **ON** wird geschaut wo die Kunden-ID(c.ID) und die

Bestell-ID(o.ID) übereinstimmen. Dadurch wird ein Ergebnis ausgegeben, dass angibt welcher Kunde eine Bestellung aufgegeben hat [Rut12][Nomb].

5 Fazit

Hive hat den großen Vorteil das mit SQL ähnlichen Befehlen gearbeitet werden kann und jeder der diese Datenbanksprache beherrscht kann schnell mit Hive umgehen.
Mit Hive können Data Warehouse Analyse einfach und schnell gestaltet werden ohne auf verschiedene MapReduce Programme zurückzugreifen. Hive übernimmt dabei die ganze Arbeit und kann sich auf die reine Query Bearbeitung fokussieren. Hive übersetzt von alleine den Query in ein MapReduce Job. Komplizierte Zeilen von Java Code werden damit umgangen und auf das Minimum einer Query reduziert. Ebenso können komplexe Joins mit wenig Aufwand erstellt werden ohne dabei auf mehrzeiligen undurchsichtigen Java Code zurückzugreifen.
Allerdings kann Hive nur eingesetzt werden, wenn die Auswertung zwischen wenigen Minuten und einigen Stunden dauert. Antwortzeiten im Millisekundenbereich wie bei konventionelle relationalen Datenbanksystemen, sind mit Hive und Hadoop nicht zu erreichen. Für Real Time Querys kann die Impala-Architektur parallel zu Hive installiert werden und kann mit Abfragen die kleinen Ergebnismengen erzeugen deutliche Geschwindigkeitsvorteile erzielen. Eine andere Möglichkeit ist das von Facebook 2013 entwickelte Presto, das eine interaktive SQL-Engine für Hadoop darstellen soll und für die Analyse von großen Datenmengen geeignet ist. Dabei soll Presto zehnmal schneller sein als das Gespann aus Hive und Map/Reduce. Bei großen Datenmenge in Petabytebereich kommt das Gespann aus Map/Reduce und Hive an seine Grenzen. Aus diesem Grund hat Facebook Pesto entwickelt, das für kurze Antwortzeiten optimiert ist [Ihl][War].

Literatur

[Far11] FARKISCH, Kiumars: *Data-Warehouse-System kompakt - Aufbau, Architektur, Grundfunktion*. Deutschland : Springer, 2011

[Ihl] IHLENFELD, Jens: *Wie Facebook 300 Petabyte in Echtzeit analysiert*

[Noma] NOMINANDUM, Nomen: *Apache Foundation Hive*

[Nomb] NOMINANDUM, Nomen: *Hive Tutorial*

[Rut12] RUTBERGLEN, Jason: *Programming Hive*. USA : O'Reilly, 2012

[Thu] THUSOO, Ashish: Hive - A Warehousing Solution Over a Map-Reduce Framework.

[War] WARTALA, Ramon: *Realtime SQL mit Hadoop Getrennt marschieren*, `http://www.heise.de/ix/artikel/Getrennt-marschieren-1919751.html`

Abschließende Erklärung

Ich versichere hiermit, dass ich meine Seminararbeit Apache Hive selbständig und ohne fremde Hilfe angefertigt habe, und dass ich alle von anderen Autoren wörtlich übernommenen Stellen wie auch die sich an die Gedankengänge anderer Autoren eng anlegenden Ausführungen meiner Arbeit besonders gekennzeichnet und die Quellen zitiert habe.

Oldenburg, den 26. September 2016

MapReduce und Pig

Seminararbeit

Themenersteller:	Prof. Dr.-Ing. habil. Jorge Marx Gómez
Betreuer:	M. Eng. & Tech. Viktor Dmitriyev Dr.-Ing. Andreas Solsbach M.Sc. Ola Mustafa
Vorgelegt von:	Farzaneh Haidary farzaneh.haidary@uni-oldenburg.de
Abgabetermin:	2. Dezember 2015

Inhaltsverzeichnis

Abkürzungen

DAG	Directed Acyclic Graph
BSON	Binary JSON
GFS	Google File System
HAWQ	Hadoop with Query
HDFS	Hadoop Distributed File System
JSON	JavaScript Object Notation
MR	Map/Reduce
UDFs	User Defined Functions
XML	Extensible Markup Language
XPath	XML Path Language
YARN	Yet Another Resource Negotiator

Abbildungen

Tabellen

1 Motivation

Nehmen wir an, dass unsere Dateien, die wir verarbeiten möchten, auf einer Maschine mit einer CPU und einer Festplatte liegen. Falls der Datenbestand, wie zum Beispiel Statistiken für das Machine Learning, zusammenhängend auf der Festplatte gespeichert wurden, dann können wir sie komplett in den Arbeitsspeicher des Computers laden, sie berechnen und verarbeiten. Wenn die Datenmenge jedoch nicht in unsere Speicherressourcen passt, müssen wir einen anderen Weg für die Verarbeitung der Dateien finden, um unser Data-Mining trotzdem durchführen zu können. So würde der Computer die Dateien der Festplatte aufteilen und jeweils stückweise verarbeiten. Dieser sequentielle Vorgang braucht mehr Zeit, weil ständig die Datenhappen von der Festplatte in den Speicher transferiert, bearbeitet und nacheinander deren Ergebnisse gespeichert werden müssen. Ein solches System könnte zum Beispiel in jeder Sekunde 100 Megabytes an Daten transformieren. Eine Big Data-Quelle mit einer Größe von 7 Petabyte würde bereits ungefähr 30 Minuten nur für das Lesen der Daten beanspruchen. Somit stößt eine Architektur mit lediglich einem Knoten der Speicherung und Verarbeitung schnell an ihre Grenzen.

Das Programmierkonzept MapReduce wurde 2004 von Google-Forschern erfunden (siehe [DG08]) und brachte ebenfalls ein neues verteiltes Dateisystem namens GFS (Google File System) (siehe [GGL03]) mit sich. Beide Paper kombiniert miteinander versprachen einen neuen Ansatz, um das angesprochene Problem zu lösen. Jedoch verblieben die für die Umsetzung nötigen Softwareprodukte proprietär und damit im Besitz von Google. Doug Cutting entwickelte, inspiriert von dem Konzept, für seine Suchmaschine Apache Nutch (sehr ähnlich zu Google) ein neues Projekt, dem er den Namen 'Hadoop' gab. Apache Hadoop als Open-Source-Software hat die Einstiegshürden in die Welt der verteilten Berechnung auf Big-Data-Quellen drastisch gesenkt. Es stellt eine freie Implementierung von MapReduce und somit einen wichtigen Meilenstein in der Bewältigung riesiger Datenaufkommen dar.

Um es Nicht-Programmierern wie zum Beispiel Analysten zu ermöglichen, ebenfalls von MapReduce zu profitieren, anstatt die Jobs erst in einer Programmiersprache wie Java schreiben zu müssen, wurde bereits 2006 ein Tool namens Apache Pig entwickelt. Dieses stammt von der Forschungsabteilung bei Yahoo und

stellt eine leichter zu verstehende Hochsprache namens PigLatin zur Verfügung, die vor der Ausführung ad-hoc zum Beispiel in MapReduce-Jobs übersetzt wird. Pig befindet sich seit 2007 ebenfalls in der Apache Software Foundation, bringt viele vorgefertigte Plugins für die meisten Standardaufgaben der Big-Data-Analyse mit und ist somit das zweitwichtigste Thema der vorliegenden Seminararbeit.

2 Grundlagen

In diesem Kapitel werden die Grundlagen für die vorliegende Seminararbeit gelegt. Es wird auf die Hintergründe der wichtigsten Konzepte und die Architekturen der verwendeten Werkzeuge eingegangen. Zunächst geht es um das Konzept MapReduce und dessen Implementierung Apache Hadoop. Im Weiteren geht es um Apache Pig als Tool in dessen Ökosystem.

2.1 Geschichte von Hadoop

Die Bibliothek *Apache Lucene* dient der Suche von Wörtern innerhalb von Texten. Sie fand zum ersten Mal 2002 in der Suchmaschine *Apache Nutch* Verwendung. Im Jahre 2003 veröffentlichte Google ein Paper über das GFS (Google File System) (siehe [GGL03]). Dieses beschreibt eine Architektur zur Verwaltung und effizienten Speicherung von großen Datenaufkommen (auch Big Data genannt). Der Hauptentwickler der beiden Open-Source-Projekte Lucene und Nutch, Doug Cutting, beschloss daraufhin, für Nutch ein eigenes verteiltes Dateisystem auf Grundlage von GFS zu schaffen, das sogenannte NDFS (Nutch Distributed File System).

2004 folgte ein zweites Paper von Google über den sogenannten 'MapReduce'-Algorithmus (siehe [DG08]), welcher sich mit der Bearbeitung von großen Datenmengen auf einem Cluster beschäftigt. Noch im selben Jahr veröffentlichten Doug Cutting und Mike Cafarella eine erste eigene Implementierung von MapReduce sowie eine neue Version ihres Dateisystems, das nun HDFS (Hadoop Distributed File System) genannt wurde. Darauf aufbauend wurde die Basis von Nutch 2005 derart umgebaut, dass es nun MapReduce und das verteilte Dateisystem für seinen Kern nutzen sollte. 2006 trat dieser Kern als eigenes Projekt namens Hadoop ins Rampenlicht.

Zu dieser Zeit wechselte Doug Cutting zu Yahoo und brachte dieses Knowhow mit in den Google-Konkurrenten. 2008 stellte Yahoo schließlich das Projekt der Öffentlichkeit als Open-Source-Software vor. Zudem stieg Anfang 2008 Hadoop zu einem Top-Level-Projekt in der Apache Foundation auf und wurde somit unabhängig von Apache Nutch. Schließlich handelte es sich dabei grundsätzlich um ein Problem der großen Datenmengen, das auch viele andere Anwendungen in vergleichbarer Art und Weise hatten und haben.

2.2 Hadoop Distributed Filesystem (HDFS)

Um zu verstehen, wie es möglich ist, dass ein Hadoop-Cluster über hunderte (oder sogar tausende) von Knoten skalieren kann, muss zuerst das HDFS beschrieben werden. Das *Hadoop Distributed File System* (HDFS) folgt dem Vorbild des *Google File System* und stellt daher, wie der Name schon sagt, ein verteiltes Dateisystem dar. Die Daten in einem Hadoop-Cluster werden in kleinere Stücke, Blöcke genannt, zerlegt und auf die einzelnen Knoten verteilt. Auf diese Weise können die Map- und Reduce-Funktionen auf kleineren Teilmengen ihres viel größeren Datensatzes ausgeführt werden. Hierdurch ergibt sich die horizontale Skalierbarkeit, die für große Datenverarbeitungen im Rahmen von Big Data benötigt wird [Wad14].

HDFS nutzt das Konzept eines Blockes, welcher die kleinste Einheit für das Schreiben und Lesen von Daten auf einer Disk darstellt. Die Größe eines Blockes in diesem verteilten Dateisystem ist üblicherweise 64 oder 128 Megabytes, während die Blockgröße auf dem physischen Datenträger jedoch normalerweise 512 oder 4096 Bytes beträgt. Da im HDFS die Dateien nur einmal geschrieben ('append' genannt), dafür aber häufig gelesen werden, müssen Operationen zum Lesen effizienter sein als die zum Schreiben. Beim Schreiben hingegen muss die Replikation beachtet werden, damit die Daten nicht bei Verlust eines Knotens verloren gehen.

2.2.1 Architektur

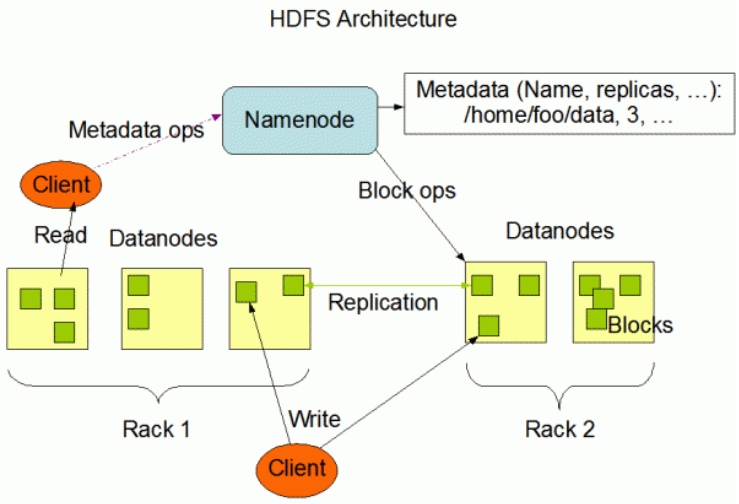


Abbildung 2.1: HDFS-Architektur (aus [Apa15c])

Ein HDFS-Cluster verfügt über mindestens zwei Knoten. Das hierbei verwendete Muster ist das Master-Worker-Pattern. Eine NameNode wird als Master betrieben und muss über mindestens eine DataNode als Worker verfügen. Ein Master-

Knoten, genannt NameNode, verfolgt und verwaltet alle Metadaten des Dateisystems über Verzeichnisstrukturen und Dateien. Diese Informationen werden auf dem lokalen Datenträger in zweierlei Form gespeichert, dem NameSpace-Image und -EditLog. Zudem kennt ein NameNode alle DataNodes, da es darauf die Blöcke einer Datei verteilt und deshalb die Adressen von jedem Speicherplatz für einen späteren Lesevorgang verfolgen muss.

Wie in Abbildung 2.1 zu sehen, übernehmen die sogenannten DataNodes die Rolle der Slaves und sind für die Verwaltung der Nutzdaten im HDFS auf den einzelnen Clusterknoten zuständig. Der Benutzer greift auf das Dateisystem zu und kommuniziert mit NameNode und DataNodes. DataNodes sind die Workhorses des Dateisystems. Sie speichern und rufen die Blöcke ab, wenn auf sie von den Clients oder der NameNode aus zugegriffen wird. Zudem berichten sie an die NameNode eine Liste von Blöcken, die sie schon gespeichert haben, wiederum zurück. Ohne die NameNode ist es nicht möglich, das Dateisystem zu verwenden; Es wäre somit ein SPOF (Single Point of Failure). Deswegen gibt es zusätzlich automatische Backups der Daten im Zuge der Replikation sowie die Möglichkeit, eine sekundäre NameNode einzurichten.

2.3 MapReduce

Ein verteiltes Dateisystem ermöglicht und erfordert insbesondere neuartige Programmierparadigmen. MapReduce ist ein solches Programmiermodell für die Verarbeitung von großen Datenmengen. Es gibt darin zwei Hauptphasen: Die Map-Phase, welche Operationen auf den Eingabedaten durchführt, und die Reduce-Phase, welche die Ergebnisse der Map-Phase zu einem einzelnen Ergebnis zusammenführt. Der Benutzer definiert lediglich eine Map- sowie eine Reduce-Funktion, um ein entsprechendes Framework für das verteilte Rechnen nutzen zu können. Die restlichen Aufgaben wie zum Beispiel die Verteilung und Ausführung von Jobs werden automatisch erledigt und senken somit die Einstiegshürde für den unerfahrenen Programmierer [DG08].

2.3.1 Apache Hadoop

Das Programmiermodell MapReduce war lange Zeit im proprietären Besitz von Google und somit nicht für die Allgemeinheit verfügbar. Daraufhin wurde es zwar als wissenschaftliches Papier veröffentlicht, war jedoch auch nicht außerhalb einer kleinen Gemeinschaft bekannt. Mehrere unabhängige Implementierungen des Modells würden einen größeren Aufwand erzeugen und zudem viel Knowhow vieler einzelner Entwickler erfordern.

Apache Hadoop ist ein Java-basiertes Open-Source-Framework für die Analyse von großen Datenmengen. Die Daten und Rechenoperationen werden hierbei auf viele Rechner mit Standardhardware verteilt. Es stellt somit eine freie Implementierung von MapReduce dar. Normale Anwendungen können lediglich vertikal skalieren (scale up). So nutzt die Java Virtual Machine (JVM) auf einem einzelnen Rechner beispielsweise mehr Threads, mehr Arbeitsspeicher etc. Eine Hadoop-gestützte Software kann jedoch zusätzlich horizontal skalieren (scale out). So sind theoretisch unbegrenzt viele Knoten für MapReduce sowie HDFS nutzbar.

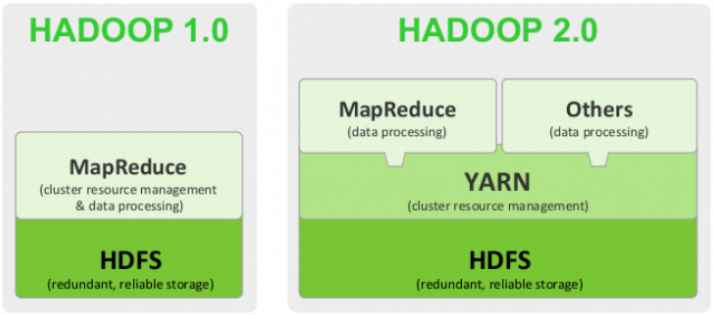


Abbildung 2.2: Gegenüberstellung von Hadoop 1.x und 2.x (aus [Whi12])

Die Architektur von Hadoop in Version 1.x besteht aus zwei Hauptkomponenten: Hadoop Distributed File System (HDFS) und MapReduce. In dem neueren Versionszweig 2.x kommt zudem auch YARN (Yet Another Resource Negotiator) hinzu, welches von MapReduce abstrahiert und auch die parallele Ausführung von andersartigen Jobs in einem Cluster erlaubt (siehe Abbildung 2.2).

Im Wesentlichen arbeitet Hadoop mit einem verteilten Modell, welches somit nicht über mächtige Einzelrechner wie Mainframes funktioniert, sondern über beträchtlich günstigere, gewöhnliche Standardhardware. Diese wird für ein System eines Masters sowie mehrerer beziehungsweise beliebiger Slave-Nodes eingesetzt. Für jeden Slave-Node gibt es je einen Task Tracker, der die Ausführung von Tasks anzeigt, sowie eine Data Node, welche die jeweilig benötigten Daten zur Verarbeitung auf diesem Knoten vorhält. Der Master-Node besitzt zusätzlich einen Job Tracker, der die Jobausführung überwacht, und eine Name Node, welche zur Auffindung von Dateien im Gesamtsystem dient [Apa10]. Die einzelnen Details wurden bereits in den jeweiligen Abschnitten zu MapReduce und HDFS genannt.

2.4 Apache Pig

Das Ökosystem um Hadoop besteht aus den verschiedensten Tools, welche hier Projekte genannt und beinahe beliebig zusammen kombiniert werden können. Ei-

nes dieser Projekte ist Apache Pig, welches im Juni 2015 in der Version 0.15 veröffentlicht wurde [Apa15a]. Es ist eine Plattform für die Analyse von großen Datenmengen. So ermöglicht sie die Erstellung von Programmen für Apache Hadoop, indem sie eine Hochsprache für die Definition von Datenflüssen namens Pig Latin bietet.

Pig wurde in einer Forschungseinrichtung bei Yahoo im Jahr 2006 entwickelt und dann bereits 2007 in die Apache Foundation überführt. Das Tool ist in der Programmiersprache Java verfasst und führt seine Skripte im Apache Hadoop YARN aus, unter Zuhilfenahme von MapReduce und HDFS. Die Pig Latin-Skripte sind eine Alternative zu Java für das Erstellen von MapReduce-Lösungen. Es macht die Arbeit mit MapReduce einfacher, da der Nutzer sich nicht in Java auskennen muss. Stattdessen kann der Nutzer seine Funktionen oder Erweiterungen auch in anderen Sprachen wie zum Beispiel JavaScript, Python, Ruby, etc. schreiben [Apa15d]. Diese heißen dann UDFs (User Defined Functions).

Pig kann auf zwei verschiedene Arten eingesetzt werden: Per Grunt-Shell oder per Java-Kommandozeile beziehungsweise -Bibliothek. Zudem gibt es zwei Betriebsarten (im Englischen 'modes' genannt) in der Pig-Version für Apache Hadoop 2.x: Der erste ist der lokale Modus, der lediglich Zugriff auf eine einzelne Maschine bietet, wobei sich alle Dateien im lokalen Dateisystem befinden. Diesen gibt es zusätzlich in einer Apache Tez-Variante, 'tez_local' genannt. Der zweite ist der Server-Modus, der Zugang zu einem Hadoop-Cluster benötigt und seine Daten in einem zentralen HDFS speichert. Bei letzterem wird entweder ein MapReduce- oder ein Tez-Job von Pig erzeugt und zum Cluster geschickt [Apa15b].

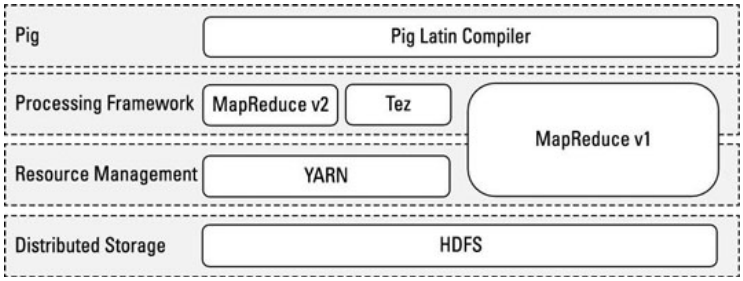


Abbildung 2.3: Architektur von Pig (aus [deR15])

2.4.1 Architektur

In der Abbildung 2.3 [deR15] ist die Architektur von Apache Pig zu sehen. Die unterste Schicht stellt die Datenspeicherung per HDFS zur Verfügung und ist damit die Grundlage für alles weitere. Darauf bauen die beiden Schichten der Ressourcenverwaltung und des Verarbeitungsframeworks auf. Bei Hadoop 1.x sind diese

Aufgaben untrennbar in lediglich einem Modul namens MapReduce miteinander verworren. Hadoop 2.x lagert die Ressourcenverwaltung in einem eigenen Modul namens YARN aus. Darauf setzen dann verschiedene Frameworks zur Verarbeitung von großen Datenmengen auf. Zum einen ist dies die zweite Version von MapReduce, welche die Ausführung auch alter Jobs der ersten Version ermöglicht. Zudem gibt es nun ein Framework namens Apache Tez, welches seine Jobs als direkte azyklische Graphen ausführt und daher das Modell von Pig direkt unterstützen kann [Par13]. Dies führt zu einer höheren Ausführungsgeschwindigkeit, da ansonsten unter Umständen mehrere MapReduce-Jobs miteinander verschachtelt werden müssen. Die oberste Schicht ist der Pig Latin-Compiler, welcher dem Benutzer ermöglicht, dasselbe Skript ohne irgendeine Änderung auch auf verschiedenen Plattformen zu nutzen.

2.4.2 Laden und Speichern

In Apache Pig ist eine wichtiger Teil die Datenmanipulation. Dafür ist es nötig, die Dateien in Pig zu laden, zu verarbeiten und am Ende zu speichern. Dafür gibt es zwei Funktionsarten, 'Load' und 'Store'-Funktionen. Diese können symmetrisch sein, müssen es aber nicht, da nicht jedes Format sowohl geladen, als auch gespeichert werden muss oder kann. Mit 'Load' werden die Daten aus der Datei in die Pig-Skripte importiert und mit 'Store' werden die Daten wiederum aus Pig exportiert. In der Tabelle 2.1 sind ein paar Beispiele für verschiedene verbreitete Formate aufgelistet.

Name	**Format**	**Laden/Speichern**
org.apache.pig- .builtin.TextLoader()	Plain Text	ja/nein
org.apache.pig.-builtin.JsonLoader	JSON	ja/ja
org.apache.pig.- piggybank.storage.XMLLoader	XML	ja/nein
com.mongodb.hadoop.-pig.- MongoLoader [FDN13]	BSON	ja/ja
org.apache.pig.-piggybank.- storage.HiveColumnarLoader	Hive	ja/nein

Tabelle 2.1: Verschiedene Formate zum Laden und/oder Speichern in Pig

So kann der Benutzer zum Beispiel Daten im Format JSON mit dem Befehl *LOAD 'file' USING org.apache.pig.piggybank.storage.JsonLoader()* laden. Nach der Verarbeitung ist es zudem möglich, die Ergebnisse wiederum im Format JSON abzu-

speichern. Manche Formate hingegen werden von Pig derzeit nur für das Laden unterstützt, wie zum Beispiel XML. Allerdings bietet Apache Pig auch eine Programmierschnittstelle (API genannt) zur Erweiterung der in Pig Latin verfügbaren Funktionen an. Diese Plugins können je nach Vorliebe des jeweiligen Entwicklers in Java, Python, JavaScript, Ruby oder Groovy programmiert werden. Die entstehenden Java-Archive müssen dann zu Beginn eines Skriptes per *REGISTER* geladen werden.

2.4.3 Unterstützung von Apache Hive

Apache Hive ist ein Open-Source-Projekt im Ökosystem von Hadoop, welches für die Abfrage und Analyse von in HDFS gespeicherten großen Datenmengen gedacht ist. Abfragen werden in der Sprache HiveQL ausgedrückt, welche sehr ähnlich wie SQL aussieht. Hive übersetzt diese SQL-Abfragen zu MapReduce- oder auch Tez-Jobs, die daraufhin in Hadoops YARN ausgeführt werden. Dies sieht wie folgt aus: Der Benutzer kann eine Abfrage über die Benutzeroberfläche abschicken. Die Abfrage wird vom Driver empfangen, woraufhin dieser den Query-Plan vom Compiler abruft. Danach fordert der Compiler die Metadaten vom Metastore an. Mit Hilfe der Metadaten erstellt der Compiler den Plan und sendet diesen an den Driver. Dieser Plan wird an die Execution Engine geschickt, welche die Anfrage mit Hilfe der Hadoop-Infrastruktur beantwortet. Sobald die Ergebnisse vorliegen, werden diese an den Benutzer ausgeliefert.

In der Abbildung 2.4 ist als zentraler Knotenpunkt die relationale Abstraktionsschicht *HCatalog* zu sehen. Sie ermöglicht das Lesen und Schreiben verschiedenster Dateiformate und stellt durch ihre Knotenfunktion eine Möglichkeit dar, dass zahlreiche Tools im Hadoop-Ökosystem ihre Daten untereinander interoperabel austauschen können [Lev13]. Hier an dieser Stelle ist vor allem die Verbindung zwischen Apache Pig und Apache Hive von Interesse. Zur Veranschaulichung wird im Folgenden dasselbe Beispiel einmal in HiveQL und daraufhin einmal in Pig realisiert.

Der aufgeführte Code ist ein Beispiel von HiveQL. Die erste Zeile zeigt eine JDBC-Url, so wie sie ein Java-Programm für die Verbindung mit einem Apache Hive-Server benötigt. In der zweiten Zeile selektieren wir das durchschnittliche Gehalt unserer Mitarbeiter/innen. Am Ende speichern wir das Ergebnis in einer anderen Tabelle, wofür wir jedoch angesichts des Fehlens von Variablen das SELECT-Statement wiederholen müssen.

Das Listing **??** zeigt sinngemäß dasselbe Beispiel in der Hochsprache PigLatin. Die erste Zeile sorgt dafür, die Tabelle 'employees' mit Hilfe des bereits bei Pig mitgelieferten *HCatLoader* aus dem HCatalog zu laden. Sie wurde zuvor zum Beispiel

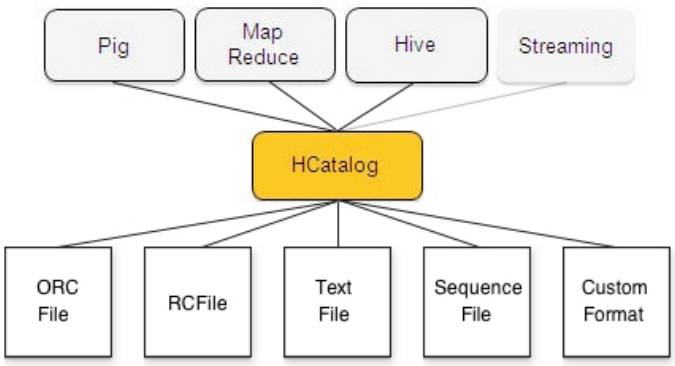


Abbildung 2.4: HCatalog als Verbindung verschiedener Tools (aus [Lev13])

SQL 2.1: Beispiel in HiveQL

```
/* jdbc:hive2://<host>:<port>/<db> */
SELECT AVG(salary) FROM employees;
INSERT INTO results
SELECT AVG(salary) FROM employees;
```

Bash 2.2: Beispiel in PigLatin

```
employees = LOAD 'employees'
USING org.apache.hcatalog.pig.HCatLoader();
salaries = GROUP employees ALL;
out = FOREACH salaries GENERATE AVG(employees.salary);
DUMP out;
STORE out INTO 'results' USING (...).pig.HCatStorer();
```

als spalten-orientierte RCFile (Record Columnar File) von Hive im HDFS gespeichert (siehe Empfehlung laut [Pro14]). Sobald die Datei geladen wurde, gruppieren wir die Einträge der Tabelle. Auf dieser Gruppierung ermitteln wir das durchschnittliche Gehalt unserer Mitarbeiter/innen. Mit Hilfe des *DUMP*-Befehls wird das Ergebnis bei der Skriptausführung angezeigt. Zuletzt speichern wir letzteres in der Datei 'results'.

2.4.4 Unterstützung von Apache HAWQ

Apache HAWQ ist ein neues, experimentelles Mitglied (erst seit September 2015) in der Hadoop-Familie und stammt vom Big-Data-Spezialisten Pivotal. Es tritt in Konkurrenz zum bereits lange etablierten Apache Hive, da es ebenfalls einen SQL-Dialekt mit sich bringt, jedoch damit wirbt, Anfragen massiv parallel zu verarbeiten (MPP). Dadurch eigne es sich besonders als analytische Datenbank für Unternehmen mit hohen Anforderungen an die Geschwindigkeit. Dies ist eventuell auch dadurch bedingt, dass es im Gegensatz zu Hive direkt vom HDFS liest und wiederum auf dieses ohne Umwege schreibt [Piv15].

Bash 2.3: Beispiel in PigLatin für HAWQ

```
1   employees = LOAD 'hawq://mdw1:5432/company'
2   USING com.gopivotal.pig.HawqLoader('employees');
3   ...
```

Wie in Listing **??** zu sehen, gibt es für die Kombination von Pig und HAWQ eine Klasse namens *com.gopivotal.pig.HawqLoader*, welche sich zu einem beliebigen Server (Host:Port) mitsamt Datenbankangabe verbinden kann und das Laden seiner Daten ermöglicht [Piv14].

3 Beispiele

Stellen wir uns vor, dass ein Nutzer sich zur Pariser Klimakonferenz über die globale Erwärmung informieren möchte. Hierzu möchte er sich die durchschnittliche Temperatur pro Jahr im Laufe der letzten Jahre anschauen und sucht dabei einen Aufwärtstrend. Ein Dienstleister bietet ihm dafür eine Webanwendung, welche nun eine sehr große Datenmenge von stündlichen Temperaturen vielleicht für ganz Deutschland aggregieren muss. Um dies zu realisieren, verwenden wir das Hadoop-Ökosystem. Die Daten werden dafür zunächst im Rohformat auf unser HDFS gespeichert. Im ersten Beispiel werden wir daraufhin einen Standardjob im MapReduce-Framework von Hadoop ausführen. Das zweite Beispiel wird dieselbe Aufgabe mit Apache Pig umsetzen.

3.1 Wetterdaten im XML-Format

Im Listing **??** ist zum besseren Verständnis des nachfolgenden Codes ein Ausschnitt der Wetterdaten zu sehen. Die Daten stammen vom deutschen Wetterdienst [1] und liegen in einer Größe von mindestens 30 bis etwa 200 MB pro Datei vor. Eine Datei enthält jeweils lediglich die Daten einer Stadt beziehungsweise der dort befindlichen Hauptmessstation.

Unterhalb der Wurzel 'Data' gibt es beliebig viele 'entry'-Elemente. Jedes 'entry'-Element enthält sieben Unterelemente, welche die eigentlichen Daten kodieren. Zum jeweiligen Messwert von Lufttemperatur und relativer Feuchtigkeit wird die ID der Messstation, das Datum der Messung, das Qualitätsniveau sowie die Version der Datenstruktur gespeichert. In den beiden nachfolgenden Umsetzungsbeispielen sind lediglich das Datum sowie die Temperatur interessant und werden per XPath extrahiert.

3.2 Umsetzung mit MapReduce

Wie in Abbildung 3.1 zu sehen, verfügt der typische MapReduce-Job im Hadoop-Framework über eine Mapper-, eine Reducer- und eine Driver-Klasse. Zuerst liest

[1] siehe `http://www.dwd.de/DE/leistungen/klimadatendeutschland/`

XML 3.1: XML-Beispiel für Wetterdaten

```xml
<?xml version="1.0" encoding="UTF8"?>
<Data>
  <entry>
    <STATIONS_ID>403</STATIONS_ID>
    <MESS_DATUM>2002010100</MESS_DATUM>
    <QUALITAETS_NIVEAU>3</QUALITAETS_NIVEAU>
    <STRUKTUR_VERSION>38</STRUKTUR_VERSION>
    <LUFTTEMPERATUR>-5.5</LUFTTEMPERATUR>
    <REL_FEUCHTE>88.0</REL_FEUCHTE>
    <eor>eor</eor>
  </entry>
  ...
</Data>
```

der Mapper die Eingabedaten und produziert daraus Zwischendaten. Danach nimmt die Reducer-Klasse diese Zwischendaten auf und kombiniert sie. Die Driver-Klasse leitet von der Hadoop-Klasse *Configured* ab und implementiert das Tool-Interface, worüber der MapReduce-Job konfiguriert und ausgeführt wird. Sie verbindet somit als zentrales Element die anderen beiden Klassen und folglich die Tasks miteinander. Der Programmierer muss selbst schreiben, wie und welche Datenformate verarbeitet und kombiniert werden sollen, als auch in welchem Format diese ausgegeben werden.

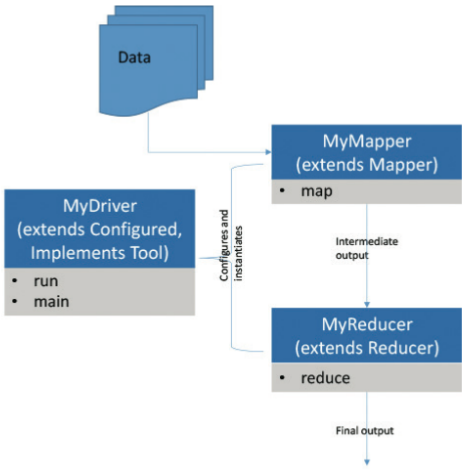


Abbildung 3.1: Struktur von MapReduce (aus [Rog15])

Im Folgenden werden nun die spezifisch für die Verarbeitung von Wetterdaten des Beispiels notwendigen abgeleiteten Klassen gezeigt und detailliert beschrie-

ben. Eine einfachere Variante dieses vorgestellten Jobs wäre beispielsweise die Umwandlung aller in Celsius gemessenen Temperaturen in Fahrenheit. Dabei wird kein Reduce-Task benötigt, da die einzelnen Ergebnisse keinerlei Abhängigkeiten zueinander besitzen. Somit kann die riesige Datenmenge vollkommen parallel, das heißt ohne Wartezeit, verarbeitet werden [Whi12].

Eine kompliziertere Variante des vorgestellten Jobs wäre beispielsweise die Berechnung sowohl von Temperaturen als auch von Feuchtigkeiten. Dabei würden zwei Reduce-Tasks benötigt, da sich jeweils einer um eine Art von Messwert kümmert. Die Partitionierung der Map-Ergebnisse erfolgt dann mit Hilfe einer Hashfunktion. Im vorliegenden Beispiel werden beide Reduce-Tasks gleichermaßen beschickt. Jeder Reduce-Task weist eine besondere Merge-Funktion auf: So verwendet der erste Task lediglich die Temperatur, während sich der zweite mit der Feuchtigkeit beschäftigt [Whi12].

Java 3.2: Driver-Klasse in Java

```
1  public class Driver {
2    public static void main(String[] args) throws Exception {
3      Configuration cfg = new Configuration();
4      cfg.set("textinputformat.record.delimiter",
5              WeatherMap.END_TAG);
6      Job job = Job.getInstance(cfg, "Weather");
7      job.setOutputKeyClass(ShortWritable.class);
8      job.setOutputValueClass(DoubleWritable.class);
9      job.setMapperClass(WeatherMap.class);
10     job.setReducerClass(WeatherReduce.class);
11     job.setInputFormatClass(TextInputFormat.class);
12     FileInputFormat.addInputPath(job, new Path("/data"));
13     job.setOutputFormatClass(TextOutputFormat.class);
14     FileOutputFormat.setOutputPath(job, new Path("/result"));
15     job.waitForCompletion(true);
16   }
17 }
```

Das Listing **??** zeigt die Driver-Klasse, welche den Input, Output, Mapper und Reducer konfiguriert und somit den zentralen Verbindungspunkt zwischen allen Klassen darstellt. Der erste Schritt besteht darin, eine neue Hadoop-Konfiguration zu erstellen. Da ein XML-basiertes Dateiformat eingelesen werden soll, wird im Beispiel hier angegeben, dass der Trennungsausdruck für die Eingabedaten nicht wie sonst üblich ein Zeilenumbruch ist, sondern dass die XML-Datei jeweils anhand des schließenden 'entry'-Tags aufgesplittet werden soll. Nun kann ein neuer *Job* erstellt werden, der diese Konfiguration und den Namen 'Weather' erhält. Der Output des Jobs soll am Ende für jedes Jahr vom Typ *ShortWritable* jeweils eine Temperatur vom Typ *DoubleWritable* sein. Die Klassen *WeatherMap* und *Weather-*

Reduce stellen die Map- sowie die Reduce-Funktion des Jobs dar. Die Datenquelle für den Job ist das Verzeichnis '/data' auf dem HDFS und die einzelnen darin enthaltenen XML-Dateien werden jeweils mit dem *TextInputFormat* von Hadoop eingelesen. Gespeichert werden die Ergebnisse im *TextOutputFormat* auf dem HDFS unter '/result'. Zum Schluss des Drivers wird der entstandene Job an den Cluster zur Ausführung geschickt und auf dessen Fertigstellung gewartet.

Java 3.3: Mapper-Klasse in Java

```
1  public class WeatherMap extends Mapper<LongWritable,
2         Text, ShortWritable, DoubleWritable> {
3
4    private static final String TAG = "entry";
5    private static final String START_TAG = "<" + TAG + ">";
6    static final String END_TAG = "</" + TAG + ">";
7
8    @Override
9    protected void map(LongWritable key, Text val, Context ctx)
10       throws IOException, InterruptedException {
11     int begin;
12     String xml = val.toString();
13     if ((begin = xml.indexOf(START_TAG)) < 0) return;
14     xml = xml.substring(begin) + END_TAG;
15     try {
16       XPathFactory factory = XPathFactory.newInstance();
17       XPath xpath = factory.newXPath();
18       String jahrXPath = "/entry/MESS_DATUM/text()";
19       String jahrWert = xpath.evaluate(jahrXPath, toSrc(xml));
20       if (jahrWert == null || jahrWert.isEmpty()) return;
21       short jahr = Short.parseShort(jahrWert.substring(0, 4));
22       String tempXPath = "/entry/LUFTTEMPERATUR/text()";
23       String tempWert = xpath.evaluate(tempXPath, toSrc(xml));
24       double temp = Double.parseDouble(tempWert);
25       ctx.write(new ShortWritable(jahr),
26                 new DoubleWritable(temp));
27     } catch (Exception e) { throw new IOException(xml, e); }
28   }
29 }
```

Die Mapper-Klasse im Listing **??** ist für die Extraktion der Temperaturdaten aus unserer XML-Datei zuständig. Zuallererst wird der von Hadoop kommende Text in einen String umgewandelt. Wir suchen den Start-Tag von 'entry' in diesem XML-Fragment und schneiden eventuell davor stehende Zeichen (Leerzeichen etc.) ab. Daraufhin selektieren wir aus dem Eintrag per XPath das Messdatum und konvertieren die ersten vier Stellen zu einer einfachen Zahl namens 'jahr'. Die Lufttemperatur der Messung finden wir ebenfalls per XPath und konvertieren die-

se zur Fließkommazahl 'temp'. Zum Abschluss wird die *write*-Methode des Kontextes aufgerufen und übergibt das Zwischenergebnis für diesen einzelnen Eintrag der XML-Datei an das Hadoop-Framework für MapReduce.

Java 3.4: Reduce-Klasse in Java

```
1  public class WeatherReduce extends Reducer<ShortWritable,
2                DoubleWritable, ShortWritable, DoubleWritable> {
3    @Override
4    protected void reduce(ShortWritable key,
5                Iterable<DoubleWritable> values, Context context)
6            throws IOException, InterruptedException {
7      double sum = 0;
8    long count = 0;
9      for (DoubleWritable value : values) {
10     sum += value.get();
11     count++;
12     }
13     double average = sum / count;
14     context.write(key, new DoubleWritable(average));
15   }
16 }
```

In der Reduce-Klasse (siehe Listing **??**) werden zunächst zwei Variablen erstellt; einmal für die Summe aller Temperaturen und einmal deren Anzahl. Daraufhin durchläuft eine Schleife alle Temperaturen zu einem Jahr, addiert diese zur Summe und erhöht den Zähler um eins. Am Ende wird der Durchschnitt gebildet durch die Division der Summe mit der Anzahl. Durch den Aufruf der *write*-Methode des Kontextes wird das Ergebnis an das Hadoop-Framework für MapReduce zurückgegeben und der Job ist damit zu Ende.

3.3 Umsetzung mit Pig

Im Folgenden wird nun die Hochsprache von Apache Pig, PigLatin genannt, kurz erläutert, welche der Programmiersprache Python ähnelt. Es gibt zwei verschiedene Möglichkeiten, wie die folgenden Kommandos verwendet werden können: Einerseits lässt sich jedes Kommando interaktiv in der Shell von Pig schreiben und iterativ jeweils das Ergebnisse begutachten. Die andere Möglichkeit ist, alle Kommandos als ein Skript zu schreiben, welches Pig daraufhin nur einmal ausführt und am Ende dessen Ergebnisse präsentiert.

In Listing **??** wird hier nun ein Beispielskript für die Verarbeitung von Wetterdaten gegeben. Zuerst wird mit dem Befehl *REGISTER* die Erweiterung 'piggybank'

Bash 3.5: Pig-Skript für Wetterverarbeitung

```
1  REGISTER /root/pig/contrib/piggybank/java/piggybank.jar;
2  DEFINE XPath org.apache.pig.piggybank.evaluation.xml.XPath();
3  xml = LOAD '/data/weatherBerlin.xml' USING
4        org.apache.pig.piggybank.storage.XMLLoader('entry')
5        AS (x:chararray);
6
7  data = FOREACH xml GENERATE
8         SUBSTRING(XPath(x, 'entry/MESS_DATUM'), 0, 4) AS jahr,
9         (double)XPath(x, 'entry/LUFTTEMPERATUR') AS temp;
10 pro_jahr = GROUP data BY jahr;
11 temp_average = FOREACH pro_jahr GENERATE
12                group AS jahr, AVG(data.temp) AS avgtemp;
13 ordered_temp_avg = ORDER temp_average BY jahr ASC;
14 DUMP ordered_temp_avg;
15 STORE ordered_temp_avg INTO '/result/tempaverages'
16       USING JsonStorage();
```

geladen, um XML-Dateien einlesen zu können. *DEFINE* lädt daraufhin den darin enthaltenen XPath-Operator unter dem Alias 'XPath', um mit dieser 'XML Path Language' Teile des Wetter-XML-Dokumentes zu adressieren und auszuwerten. Für das Lesen von Daten aus einer Datei ist die *LOAD*-Funktion zuständig, welche sowohl aus einem lokalen als auch einem entfernten HDFS seinen Input beziehen kann. Hier im Beispiel werden alle Einträge aus der Datei 'weatherBerlin.xml' im Ordner '/data' mit Hilfe des in PiggyBank enthaltenen *XMLLoader* importiert und unter der Variablen 'xml' abgelegt. *FOREACH* ist eine Schleife, die über alle Elemente von 'xml' iteriert. Das Beispiel greift auf den XPath 'entry/MESS_DATUM' zu, extrahiert vom Datum lediglich die ersten vier Stellen und kombiniert dieses mit der Lufttemperatur. Die *GROUP*-Operation gruppiert diese Daten nach dem Jahr und im nächsten *FOREACH* wird daraus zu jedem Jahr der Durchschnitt der Temperatur gebildet. Zum Schluss werden diese Jahresdurchschnittstemperaturen aufsteigend nach dem jeweiligen Jahr geordnet. Der *DUMP*-Befehl ist für die Ausgabe der Ergebnisse auf der Kommandozeile gedacht, während *Store* diese hier als JSON im HDFS unter dem Ordner '/result' persistiert.

4 Zusammenfassung

In der vorliegenden Seminararbeit wurde zunächst ein Überblick über MapReduce und dessen Open-Source-Implementierung Apache Hadoop gegeben. Dazu gehörte die Behandlung von dessen grundlegender Komponente HDFS und die dahinter liegenden Konzepte wie Map- und Reduce-Tasks. Danach wurde Apache Pig als Tool im Hadoop-Ökosystem vorgestellt. Insbesondere wurde die Kombination von Pig mit anderen Projekten dargestellt. Bei der Betrachtung seiner Architektur wurde neben MapReduce ebenfalls Apache Tez als neuere Technologie besprochen. Im Falle von Apache Hive und HAWQ sind in PigLatin geschriebene Skripte als Beispiel zur Veranschaulichung gegeben worden. Den praktisch angelegten Teil bildet ein Anwendungsfall aus dem Bereich der Wetterforschung. Hierfür wurden sowohl ein MapReduce-Job programmiert als auch ein Skript in PigLatin verfasst und dargestellt.

4.1 Fazit

Die vorliegende Arbeit hat gezeigt, dass MapReduce als Konzept sich sehr gut für Anwendungen im Bereich BigData eignet. Große Datenmengen lassen sich auf relativ einfache Art und Weise effizient auf einem Cluster mit gewöhnlicher Standardhardware verarbeiten. Wenn wir jedoch die Definition eines solchen Jobs in der Programmiersprache Java und in der Hochsprache PigLatin vergleichen, können wir einen großen Vorteil in den bereits mitgelieferten Funktionen von Apache Pig erkennen. So ist Pig wesentlich einfacher für einen Anwender ohne Programmierkenntnisse, da er oder sie in PigLatin ähnliche Konzepte wie in SQL vorfindet. SQL-Begriffe wie WHERE lassen sich durch FILTER ersetzen und Gruppierungen sind beinahe gleich. Während sich bei direkter Verwendung der Hadoop-API in Java der Anwender selbst um die Eingabe- und Ausgabeformate kümmern muss, stehen in Pig zahlreiche Dateiformate sofort einsetzbar zur Verfügung. So muss in Java eine XML-Datei selbst in ihre Einzelteile zerlegt und gelesen werden, während in Pig der *XMLLoader* mitgeliefert wird. Auch zur Ausgabe von Daten muss in Java ein Format wie JSON anders als in Pig (siehe *JSONStorage*) selbst geschrieben werden. Insbesondere die wohl für viele interessante Anbindung von relationalen Abfragesystemen wie Apache Hive oder HAWQ ist in Pig wesentlich kom-

fortabler.

4.2 Ausblick

Heutzutage steigt die Datenmenge auch in kleinen und mittleren Unternehmen immer mehr an. So gab der bekannte Hadoop-Distributor Hortonworks diverse Studien in Auftrag, um einen Ausblick auf 2016 zu geben. Demnach werden in Zukunft immer mehr intelligente Geräte auf den Markt kommen, die immer mehr Daten liefern. Selbst kleinere Unternehmen können somit mit größeren Mengen von Sensordaten konfrontiert werden. Das *Internet of Things* wird folglich zum *Internet of Anything*. Zum Beispiel durch intelligente Stromzähler steigen jedoch auch die Sicherheitsanforderungen an ein Analysesystem, da nun private, personenbezogene Informationen vor unberechtigtem Zugriff geschützt werden müssen. Zukünftig wird vor allem die effektive Echtzeitverarbeitung weiter an Bedeutung gewinnen, da Analysten nicht mehr lange auf ein DataWarehouse warten möchten. Zusammengefasst geht die Professionalisierung der BigData-Software weiter voran und mehr traditionelle Unternehmen evaluieren oder verwenden bereits immer leichter einzusetzende Hadoop-Tools [Neu15].

Literatur

[Apa10] APACHE, Foundation: *JobTracker*. https://wiki.apache.org/hadoop/JobTracker, Juni 2010

[Apa15a] APACHE, Foundation: *Apache Pig Releases*. http://pig.apache.org/releases.html, Juni 2015

[Apa15b] APACHE, Foundation: *Getting Started*. http://pig.apache.org/docs/r0.15.0/start.html#execution-modes, Juni 2015

[Apa15c] APACHE, Foundation: *HDFS Architecture*. http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html, Juni 2015

[Apa15d] APACHE, Foundation: *User Defined Functions*. http://pig.apache.org/docs/r0.15.0/udf.html, Juni 2015

[deR15] DEROOS, Dirk: *The Pig Architecture in Hadoop*. http://www.dummies.com/how-to/content/the-pig-architecture-in-hadoop.html, 2015

[DG08] DEAN, Jeffrey ; GHEMAWAT, Sanjay: MapReduce: Simplified data processing on large clusters. In: *Communications of the ACM* 51 (2008), Nr. 1, S. 107–113

[FDN13] FREDERIC DANG NGOC, Francois Dang N.: *Using Hadoop Pig with MongoDB*. https://chimpler.wordpress.com/2013/02/07/using-hadoop-pig-with-mongodb/, Juli 2013

[GGL03] GHEMAWAT, Sanjay ; GOBIOFF, Howard ; LEUNG, Shun-Tak: The Google file system. In: *ACM SIGOPS operating systems review* Bd. 37 ACM, 2003, S. 29–43

[Lev13] LEVERENZ, Lefty: *HCatalog UsingHCat*. https://cwiki.apache.org/confluence/display/Hive/HCatalog+UsingHCat, Oktober 2013

[Neu15] NEUMANN, Alexander: *Ausblick auf 2016: Fünf Trends für Big Data*. http://www.heise.de/developer/meldung/Ausblick-auf-2016-Fuenf-Trends-fuer-Big-Data-3024918.html, November 2015

[Par13] PARK, Cheolsoo: *Pig on Tez*. https://cwiki.apache.org/confluence/display/PIG/Pig+on+Tez, Oktober 2013

[Piv14] PIVOTAL, Field E.: *Class HawqLoader*. http://pivotal-field-engineering.github.io/pmr-common/pmr/apidocs/com/gopivotal/pig/HawqLoader.html, 2014

[Piv15] PIVOTAL, Software: *Pivotal HD: HAWQ A True SQL Engine for Hadoop*. http://pivotal.io/big-data/white-paper/a-true-sql-engine-for-hadoop-pivotal-hd-hawq, 2015

[Pro14] PROKOPP, Christian: *Faster Big Data on Hadoop with Hive and RCFile*. http://www.semantikoz.com/blog/faster-big-data-hadoop-hive-rcfile/, März 2014

[Rog15] ROGUEWAVE, Software: *Using JMSL in Hadoop MapReduce Applications*. http://www.roguewave.com/getattachment/afe53105-2ebd-48e9-8c33-2d2f7ebebc90/Using-JMSL-in-Hadoop-MapReduce-applications, Juli 2015

[Wad14] WADKAR, Sameer: *Pro Apache Hadoop*. Berkeley, CA New York, NY : Apress, 2014. – ISBN 978–1–4302–4863–7

[Whi12] WHITE, Tom: *Hadoop, The Definitive Guide*. Beijing : O'Reilly, 2012

Abschließende Erklärung

Ich versichere hiermit, dass ich meine Seminararbeit *"MapReduce und Pig"* selbständig und ohne fremde Hilfe angefertigt habe, und dass ich alle von anderen Autoren wörtlich übernommenen Stellen wie auch die sich an die Gedankengänge anderer Autoren eng anlegenden Ausführungen meiner Arbeit besonders gekennzeichnet und die Quellen zitiert habe.

Oldenburg, den 26. September 2016 Farzaneh Haidary

Components of the Apache Hadoop Ecosystem

Seminar Paper

Director:	Prof. Dr.-Ing. habil. Jorge Marx Gómez
Supervisors:	M. Eng. & Tech. Viktor Dmitriyev Dr.-Ing. Andreas Solsbach
Submitted by:	Uwe Kopplin uwe.kopplin@uni-oldenburg.de
Submission date:	2. Dezember 2015

Contents

Abbreviations

API	Application Programming Interface
Bash	Bourne-again shell
JDK	Java Development Kit
CEWE	CEWE Stiftung & Co. KGaA
BI	Business Intelligence
CSV	Comma-separated values
DAG	Direct Acyclic Graph
DFM	DigiFoto Maker
ETL	Extract Transform Load
FTP	File Transfer Protocol
GFS	Google File System
hPDL	Hadoop Process Definition Language
HDFS	Hadoop Distributed File System
JAR	Java Archive
JVM	Java Virtual Machine
JRE	Java Runtime Environment
KDD	Knowledge Discovery in Databases
VM	Virtuelle Maschine
VM	Virtuelle Maschine
RDBMS	Relational Database Management System
SQL	Structured Query Language
XML	Extensible Markup Language
Yarn	Yet Another Resource Manager

List of Figures

Tabellen

1 Introduction

This seminar paper is written as part of the Project Group at the Carl von Ossietzky University in Oldenburg. The project is carried out in cooperation with the german company CEWE Stiftung & Co. KGaA.

1.1 CEWE Stiftung & Co. KGaA

The CEWE Stiftung & Co. KGaA (abbreviated below as "CEWE") is a medium-sized company, whose main business is the photo development. In order to withstand the technological change from analog to digital photo products and its impact on the demand, the CEWE PHOTO BOOK has been designed. This product is a great success and quickly became the most profitable of all products. So in 2014, for example, 5.9 million copies were sold.[CEW15, p. 2] With its 11 production sites, 3200 employees and a turnover of € 523.8 million in 2014, CEWE has established its brand in the European market.[CEW15, p. 5]

Very relevant to this project group is the DigiFoto Maker, a device which allows customers instant printing of photos and ordering photo products in a point of sale.

1.2 Objective

The DigiFoto Makers submit an XML file after every interaction with a client. These files are going to be analysed with Apache Hadoop in the ongoing project, called *DASH* (Data Analysis with Hadoop). To with automatic processing of BI tasks, Hadoop brings a lot of components, called the *Hadoop Ecosystem*. The objective of this seminar paper is to gain and present information about important components of the Hadoop Ecosystem. To achieve this, the neccessary steps will be identified and suitable components will be tested and evaluated. The most important ones are being presented in this paper.

2 Apache Hadoop

Apache Hadoop is an open-source framework which is written in Java. The advantages of Hadoop are the scalability and its ability to work on computer clusters. These allow the execution of very large computing processes. In this chapter a short summary of the history of Apache Hadoop and its general architecture will be described.

2.1 History

In 2002, Doug Cutting and Mike Cafarella were trying to create a scalable webcrawler. Until that time, it was common to use relational databases for storing contents of indexes [SP15, p. 10]. However, the scalability has been a problem, particularly because additional hardware resources are required to store the large amounts of data, which get almost exponentially more expensive in these dimensions. To solve this problem, a new framework was developed, which incorporated the file-based system GFS (Google File System).[Whi12, p.9] The new software was called the "Hadoop". Yahoo has been quick to recognize the benefits of this project and hired Cutting on the continuation of this project to Yahoo.

In 2008, a major milestone for the project was achieved through Yahoo: The Hadoop application "Yahoo! Search Webmap" runs on a 10,000 core Linux cluster. The production data generated are used since for every query.[Bal08] Today Apache Hadoop is a popular framework, which is used for large companies like Facebook, Google and Amazon.[Apa15a]

2.2 General Architecture

Apache Hadoop performs its computing processes parallel on multiple computer clusters. For this purpose, the MapReduce algorithm of Google Inc. is used. This procedure utilized to divide very large processes and distribute them on multiple computers. Once the processes are complete, the results are combined and used. This allows a good scalability at low cost for hardware. Due to its architecture Hadoop is a useful tool for processing Big Data. What exact purpose this is used for and what precisely is calculated is customizable by the developers.

To gather and use data, different interfaces can be set up for Hadoop. In the following figure (2.1) of Hortonworks, a company which specializes in the development of Hadoop, this is illustrated.

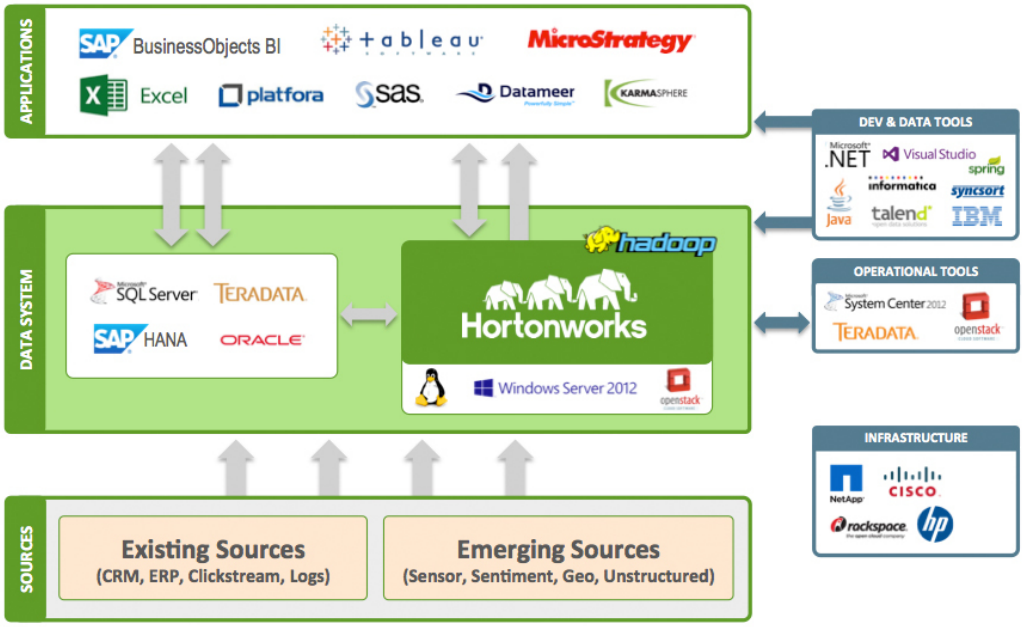


Figure 2.1: Interfaces for Apache Hadoop [Hor15c]

As shown in figure 2.1, various different sources can be used to feed Hadoop with data such as relational databases[1] or file systems. To achieve this, fitting components have to be installed and configurated. It is for example possible to receive data from file systems or webservices with Apache Flume.[2] Apache Hadoop stores the collected and generated data in its own file system, the Hadoop Distributed File System (HDFS).

The two components HDFS and YARN both form the layer of data and resource management for Hadoop. While HDFS provides the storage, YARN manages the way the data can be processed.

2.2.1 HDFS

The HDFS (Hadoop Distributed File System) is a file system that is used by Apache Hadoop for data storage. It is based on the Google File System (GFS) and was developed with Java. In the background HDFS divides files into multiple parts and

[1] See Chapter 3.1.1.
[2] See Chapter 3.1.2.

is able to store these on separate machines.[Lam10, p. 38] Users and other software components that use HDFS do not have to deal with this, because it happens on a deeper layer. This feature makes HDFS very scalable and cost-effective. It also allows the storing and processing of files with a size of up to 200 petabytes.[Hor15b] Due to its storage capabilites, HDFS is a useful tool to process Big Data.
HDFS is not a standard Unix filesystem, so the usual tools like *cp* and *ls* don't work on it.[Lam10, p.38] To work with files, Hadoop provides its own tools. These can be accessed in the following fashion:

hadoop fs -cmd <args>

Whereas *-cmd* represents the command and *<args>* the parameters. The commands work similar to the ones that are natively on Unix. For example to list a direction or remove a file these commands will be useful:

hadoop fs –ls
hadoop fs –rm onsiteSession95831.xml

To load a file from the local filesystem into the HDFS or retrieve a file from the HDFS to the local filesystem, *put* and *get* can be used. Example:

hadoop fs -put onsiteSession95831.xml
hadoop fs -get onsiteSession95831.xml

Hadoop components however are able save their data on the HDFS alone and do not require to import or export it to the native file system of the operating system.

2.2.2 YARN

YARN is an abbreviation for *"Yet Another Resource Negotiator"*. It involves a cluster management technology for Hadoop and represents a resource scheduler. YARN seperates the ressource management and scheduling from the data processing into two daemons. This allows multiple simultaneous accesses to data on Hadoop clusters in a variety of ways. In addition, other non-MapReduce components are able to use the HDFS. In the book "Apache Hadoop YARN" by A. Murthy et al., the following possibility is described: *"Applications written in any language can [...] take advantage of the combined Hadoop compute and storage assets within any size cluster."*.[MVE+14, p. 21]

YARN can be viewed as a resource layer between the Hadoop components and other applications and the Hadoop Distributed File System (HDFS). This is illustrated in figure 3.1 on page 491. As the resource layer, YARN provides access in multiple ways, such as the ones being listed [Hor15d]:

- Interactive SQL
- Real-time Streaming
- Data Science and
- Batch Processing

Due to its ability to separate data processing from scheduling, YARN enables new possibilities for other applications and processing of Big Data.

3 Components of the Hadoop Ecosystem

The Hadoop ecosystem is a collection of software components that enable and extend the functionality of Hadoop. The main tasks include the storing and analyzing of data and the management of clusters used by Hadoop.[SP15, p. 10] The following illustration (figure 3.1) of the press agency Dataconomy Media GmbH shows features combined with the enabling Hadoop components.

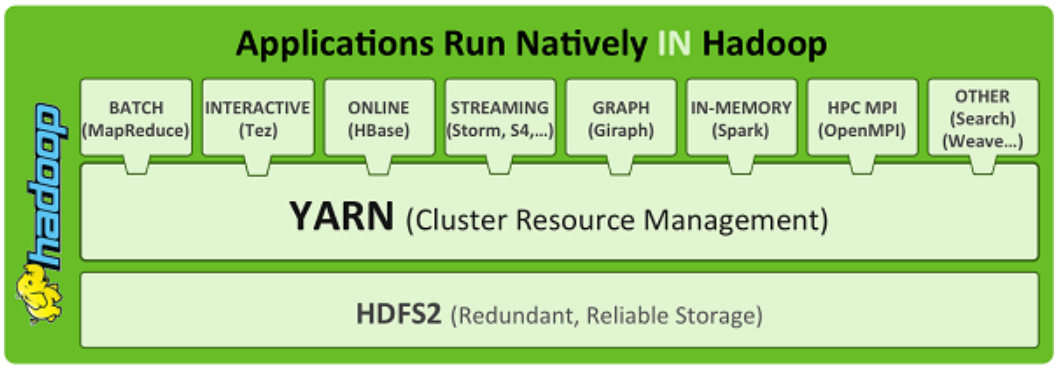


Figure 3.1: Components of the Apache Hadoop Ecosystem [McN14]

The components in the illustration (figure 3.1) are not all of the existing components.

In the sections of this chapter, several Hadoop components are presented and the usage briefly exemplified. To give you as reader a broad overview of the functionalities of Hadoop components, the choice of presented components is based on a diverse selection.

3.1 ETL-Processing

To gain valuable information out of processed data, we need to start collecting data and transform it into a coherent format in the first place. For this purpose, data has to be loaded into the HDFS at first. To achieve this, fitting components

have to be found. As shown in section 2.2.1, HDFS itself provides batch commands for retreiving data from a local file system. There are however other components that are suitable for various ways of importing data that will be presented in this section. In the upcoming project, XML files on an external FTP server will have to be loaded into the HDFS regularly. Additionally, data from relational databases that connects to the XML files could potentially be available and needs to be loaded into Hadoop aswell.

Given that Hive and other components mostly work best on column-based data like CSV files and the loaded data are hierarchically structured, a transformation has to be considered. It is possible to parse the XML when analysing the data, which would turn this into an ELT-Process, rather than ETL. Preprocessing the data can improve the performance but also possibly means a loss of data due to the different structures.

These tasks can be managed by individual applications. The Hadoop ecosystem does however provide components that are able handle this.

3.1.1 Sqoop

Apache Sqoop is a software component that functions as an interface between the Hadoop Ecosystem and relational database management systems (abbreviated below as "RDBMS"). Sqoop allows the transfer of big data sets between the HDFS and ordinary databases by executing shell commands.[Whi12, p. 527] To demonstrate an import of a RDBMS table, an import command is being shown below:

SQL 3.1: Import of an RDBMS Table via Sqoop

```
1 sqoop import –connect "jdbc:sqlserver://192.168.178.64:1521;
2 database=OLCWDWH;
3 username=user;
4 password=passwd"
5 –table OFSOrder
6 –hive–import –schema DWH
```

YARN manages the execution of this command. After importing the data into the HDFS, the data will be available to Hive. To enable the communication between Sqoop and the RDBMS it is neccessary to install the respective JDBC-driver. This can be done by simply putting the driver, contained in a JAR file, into Sqoops lib-folder.

3.1.2 Flume

Apache Flume is a tool for collecting, aggregating and moving large sets of log data. The data can be received by various external sources, such as webservers or other Hadoop components. Flume follows a predefined data flow. To demonstrate this, an illustration is shown below (figure 3.2).[Apa15c]

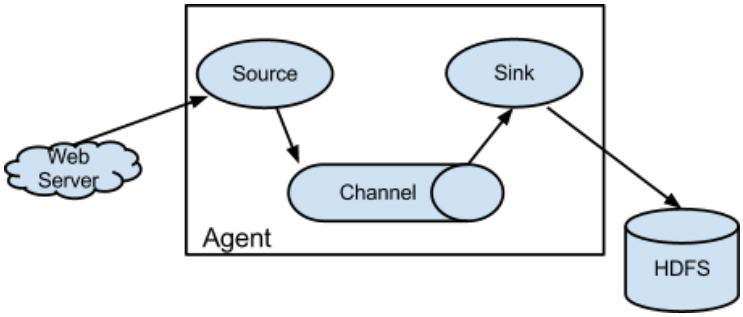


Figure 3.2: Flume Workflow

The figure 3.2 shows the general data flow of Flume. A so called *agent*, a JVM process, handles the data flow. The flow itself is separated into three steps:

1. Ingestion of data by the **Source**. The source represents an interface for receiving external data through events. Multiple sources can be configured to allow data ingestion from various external sources, such as webservices or Avro.

2. Temporarily storing the events in the **Channel**, an internal storage, until the next step is executed. The channel can be configured as a *FileChannel* for example, meaning that the event will be stored in the local file system.

3. Persisting the data in an external repository by the **Sink**. The Sink can be configurated to store the data in various different external storages, such as the HDFS.

3.1.3 Pig

Apache Pig is a useful tool for extracting, transforming and loading data. For this purpose, Pig provides a script language called *Pig Latin*. This script language brings another abstraction layer to default MapReduce algorithms because Pig automatically creates MapReduce-Jobs to execute the scripts.[JL15] It is additionally possible to run Pig Scripts via Tez or simply locally. This can be controlled by parameters. *Pig Latin* brings a good set of commands to the table to manage data. It is well-documented on their projects website.[1] Pig has a big community, possibly due to its good accessibility.

A benefit of Pig is its *Grunt Shell* in which developers can execute Pig Latin commands step by step. This is helpful for developing and testing Pig Scripts. By using Pig, developers lose control of the precise execution of their processes. This can be seen as an advantage and a disadvantage, because it serves as a time saver but also might cause applications run less efficient than possible.

3.2 Knowledge Discovery

After performing the typical *ETL process* and getting data in a structured format into the HDFS, it is time to gain valuable knowledge from it. To help achieve this, the Hadoop Ecosystem provides some useful components to handle the *Knowledge Discovery*.

3.2.1 Hive

Hive provides a data warehouse infrastructure for Apache Hadoop. Initially Facebook started this project to improve their data processing. Today however, it is an open source project which is listed as a top level project at the Apache Software Foundation. Hive contributes data summarization, adhoc querying and analysis of datasets. Hive is able to store data in a structured way onto the HDFS and allows querying the data via HiveQL, a language that is based on SQL. HiveQL does additionally provide the option to plug in developers custom MapReduce-algorithms.[Ven09, p. 5 & 330] Hive querys can be executed either in the Ambari webinterface or via command line.

The data in Hive is categorized in the following way:

Tables are the equivalent to tables in relational databases. Each table is stored in HDFS as a directory. Data in these tables is stored as files in the respective

[1] See Website: https://pig.apache.org/docs/r0.14.0/basic.html

directories. For a query of a table, Hive looks through the files in the directory table.

Partitions can be used to determine the distribution of data in a table. It is possible to split the directory into subdirectories based on a column value.

Buckets are what partitions are divided into. Each Bucket is saved as a file in the HDFS directory of a partition.[TSJ+09]

The HiveQL command for the creation of a table is shown below to demonstrate the use of partitions and buckets:

SQL 3.2: createtable.sql

```
1 CREATE TABLE dfm_user_actions(id INT, currentscreen STRING,
     timestamp TIMESTAMP)
2 PARTITIONED BY(actiontype STRING)
3 CLUSTERED BY(currentscreen) INTO 64 BUCKETS;
```

In the example above, a folder would be created with a sub folder for each action type. Contents of the table would be divided into 64 files.

3.2.2 Mahout

Mahout is a scalable framework for Hadoop which provides machine learning and data mining. The project contains a broad spectrum of diverse machine learning algorithms, such as classification, clustering and collaborative filtering. Algorithms for Mahout are written in Java. To ease the learning and developing of Mahout programs, the website of Apache Mahout provides a set of examples and tutorials.[2]

[2]See Website: https://mahout.apache.org/users/basics/algorithms.html.

3.3 Managing Hadoop

For Hadoop to work, a few components run in the background, such as YARN and the HDFS. For the project group however, these components will not directly be worked on. Instead, Ambari as the administration tool will be presented and Oozie for the management of workflows. An important component that is worth mentioning but won't be addressed in this section is ZooKeeper. Zookeeper manages the installed services and supports the communication between these.

3.3.1 Ambari

Ambari is a software component that intents to make the management of Hadoop easier for system administrators. For this purpose, a web interface is provided. Ambari focuses on three aspects [Apa15b]:

Provisioning of services for the Hadoop cluster and setting up their configuration.

Management of the services. Ambari allows users to start, stop and reconfigure services.

Monitoring the status of a cluster. In addition to this, an alert system is integrated in which administrators can define in what way alerts will be sent or displayed.

Ambari is preinstalled in the Hortonworks Sandbox.

3.3.2 Oozie

Apache Oozie is a workflow and coordinate system for managing Hadoop jobs. It is integrated into the Hadoop Ecosystem and supports scheduling for the following components:

- Apache MapReduce
- Apache Pig
- Apache Hive
- Apache Sqoop

It additionally be used to execute system commands, such as starting a Java program or shell scripts, which allows the usage of many other components.[Hor15a]

Oozie itself runs on a Tomcat server with a database and provides a web application on which the different jobs can be monitored. The database contains workflow definitions and instances of active workflows and their current state. A workflow

job is a set of actions. These are arranged in a *Direct Acyclic Graph* (abbreviated below as "DAG") and specify a predefined order of actions. The DAG itself is written in *Hadoop Process Definition Language* (hPDL), an XML language.

Furthermore, Oozie provides the option of defining *Coordinator jobs*, which can be triggered automatically by defining a specified intervall or by other action such as exploring a new file in a directory which is observed.

The minimum a workflow definition has to contain are the tags that are shown in the following exemplary workflow.xml file:

XML 3.3: workflow.xml

```
<workflow-app
  name="ProcessNewDigifotoMakerSessions"
  xmlns="uri:oozie:workflow:0.1">
    <start to="end"/>
    <end name="end"/>
</workflow-app>
```

In the shown example there are two required *control flow nodes* (start and end) which dictate the job's execution path. The start node declares at which node the workflow begins. In this case the workflow immediatly jumps to the end node and stops the workflow. There are however more control flow nodes, such as *decision, fork, join and kill*. To actually execute processes like pig or mapreduce, *action nodes* are required. Actions in Oozie are processed asynchronously by default.[Apa12]

3.4 Data Processing

The processing large amounts of data can take a lot of resources such as time and hardware. To limit the consumption of resources as far as possible, it makes sense to make processes as efficient as possible. For this reason, the Hadoop ecosystem provides various components. In this section MapReduce, Tez and Spark are presented and evaluated.

3.4.1 MapReduce

MapReduce is not as much of a component, but more a framework for processing large amounts of data. To achieve this, the processing is split over multiple machines in the cluster. This concept of MapReduced is demonstrated by the following figure:

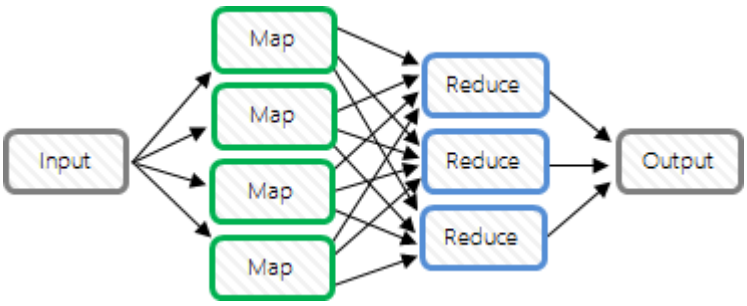


Figure 3.3: MapReduce Workflow.[3]

As seen in figure 3.3 the data that has to be processed is split into several chunks and distributed among machines in the cluster, called *single nodes*. Each of the single nodes computes their set of data in a *map task*. The results of the map task gets stored in a central file system and then aggregated though the *reduce phase*. The MapReduce framework takes care of the parallel scheduling and restarts failed processes.[Apa13] A lot of other components use MapReduce to execute their tasks and thus provide a higher level interface for developers and end users. However, to write individual MapReduce applications, Hadoop provides different APIs for programming languages like Java and C++.

3.4.2 Tez

Tez is component that provides another way to efficiently process data. Unlike with MapReduce, applications running with Tez are more complex to develop. However, the aim of the project is to support the development of components which run their processes with Tez and provide a higher level interface for end users, like Pig and Hive. It is possible to configure these higher level components to use Tez instead of MapReduce, which has shown to bring shorter response times.[Du15, p. 12]

Tez has been developed after MapReduce and brings solutions to weaknesses in MapReduce. In Tez for example the execution of JOIN operations in Hive is optimized, where MapReduce executes multiple tasks and stores intermediate data in the file system only to load it again and use it in another query.

Tez brings performance improvements over MapReduce and should be prioritized. It is however not suitable for the development of individual applications. For this purpose however, Pig provides a higher level functionality while it can be executed using Tez.

[3]Source: Own illustration.

3.4.3 Spark

Spark is comparable to Tez and is a framework that focuses on *In-memory computing* to bring high performance while executing tasks. Unlinke Tez however, it brings a lot of additional components that work on top of its processing module. Figure 3.4 shows the Spark package.

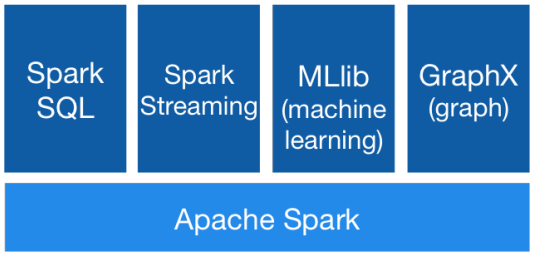


Figure 3.4: The Spark Stack.[4]

For streaming there is *Spark Streaming*, for visualization *GraphX*, machine learning is covered by the library *MLlib* and to work with relational data there's an own SQL module called *Shark*. Shark can be used with the language *Spark SQL* which is very similar to other SQL languages.

Due to the fact that Spark was released after Hive and many users being content with Hive, Spark has added compatibility of HiveQL queries and is able to work on existing Hive data. All in all, Spark brings an allaround package for Data Analysis that covers a lot of BI-user's needs. It is an alternative to building an own system out of many components. It does however require a lot of RAM to process information due to its focus on In-memory computing.[RLOW15, p. 4]

[4]Source: http://spark.apache.org/.

4 Conclusion

In order to gain valuable information out of incoming data, the Hadoop Ecosystem provides a lot of suitable tools. A few of those have been tested and presented in this paper. The performance of components has not been tested due to a limited hardware. Performance however is not the primary factor for the ongoing project in which incoming XML-files will be analysed, because the amount of data is hardly more than one gigabyte per day. The topic of visualization has been left out because it is being discussed in more detail in another paper. In this paper the evaluation and presentation of other (core) components has been prioritized.

Loading data into the HDFS has been an easy task. The biggest challenge however was the initial circumstance of managing data in XML-format instead of a column-based format like CSV. There are two possibilities: Either to transform the data, which for example Pig is suited to do. Or to work with the given format and parse the XML files when required, which is also possible with components like Hive. Preprocessing the files can cause a loss of data because of the nature of hierarchical structure and its incompability with column based formats. Both solutions have to be tested and evaluated further because the decision between both critical for the upcoming effort and the quality of the project's results.

Another subject that has to further be addressed is the chosing between the framework Spark and other components. Spark is newer and its developers claim a better performance. That however is not the most critical factor for this project as explained above.

While researching and testing the components, the general realisation was to use high level components as much as possible. While it is possible to write individual MapReduce applications, components like Pig provide a good set of commands to manage data while having an easy to use script language. Hadoop brings many components to the table that are capable to solve typical problems. Each component that has been looked into has had a good documentation.

Bibliography

[Apa12] APACHE SOFTWARE FOUNDATION: *Oozie Specification, a Hadoop Workflow System*. https://oozie.apache.org/docs/3.2.0-incubating/WorkflowFunctionalSpec.html, 2012. – Online; Accessed on November 6, 2015.

[Apa13] APACHE SOFTWARE FOUNDATION: *Apache Hadoop MapReduce Documentation*. https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html, 2013. – Online; Accessed on November 24, 2015.

[Apa15a] APACHE SOFTWARE FOUNDATION: *Alphabetical list of institutions that are using Hadoop for educational or production uses*. http://wiki.apache.org/hadoop/PoweredBy, 2015. – Online; Accessed on October 20, 2015.

[Apa15b] APACHE SOFTWARE FOUNDATION: *Ambari*. https://ambari.apache.org/, 2015. – Online; Accessed on November 11, 2015.

[Apa15c] APACHE SOFTWARE FOUNDATION: *Apache Flume - A service for streaming logs into Hadoop*. http://flume.apache.org/, 2015. – Online; Accessed on November 1, 2015.

[Apa15d] APACHE SOFTWARE FOUNDATION: *Apache Pig*. https://pig.apache.org/, 2015. – Online; Accessed on December 1, 2015.

[Bal08] BALDESCHWIELER, Eric: *Yahoo! Launches World's Largest Hadoop Production Application*. https://developer.yahoo.com/blogs/hadoop/yahoo-launches-world-largest-hadoop-production-application-398.html, February 2008. – Online; Accessed on October 17, 2015.

[CEW15] CEWE STIFTUNG & CO. KGAA: *Annual Business Report 2014*. Oldenburg : http://company.cewe.de/en/, March 2015. – Accessed on October 14, 2015.

[Du15] DU, Dayong: *Apache Hive Essentials*. Birmingham, UK : Packt Publishing, 2015

[Hor15a] HORTONWORKS: *Apache Oozie*. http://hortonworks.com/hadoop/oozie/, 2015. – Online; Accessed on November 4, 2015.

[Hor15b] HORTONWORKS: *HDFS - A distributed Java-based file system for storing large volumes of data*. http://hortonworks.com/hadoop/hdfs/, 2015. – Online; Accessed on October 27, 2015.

[Hor15c] HORTONWORKS: *Hortonworks Data Platform*. http://hortonworks.com/hdp/, 2015. – Online; Accessed on October 21, 2015.

[Hor15d] HORTONWORKS: *YARN - The Architectural Center of Enterprise Hadoop*. http://hortonworks.com/hadoop/yarn/, 2015. – Online; Accessed on October 31, 2015.

[JL15] JOOS, Thomas ; LITZEL, Nico: *Apache Pig – Datenanalyse für Hadoop*. http://www.bigdata-insider.de/apache-pig-datenanalyse-fuer-hadoop-a-476349/, March 2015. – Online; Accessed on October 25, 2015.

[Lam10] LAM, Chuck: *Hadoop in Action*. Shelter Island, NY : Manning Publications, 2010

[McN14] MCNULTY, Eileen: *Hadoop: The Components You Need to Know*. http://dataconomy.com/hadoop-components-need-know/, June 2014. – Online; Accessed on October 23, 2015.

[MVE+14] MURTHY, Arun ; VAVILAPALLI, Vinod ; EADLINE, Douglas ; NIEMIEC, Joseph ; MARKHAM, Jeff: *Apache Hadoop YARN: Moving beyond MapReduce and Batch Processing with Apache Hadoop 2*. Addison-Wesley Data & Analytics, 2014

[RLOW15] RYZA, Sandy ; LASERSON, Uri ; OWEN, Sean ; WILLS, Josh: *Advanced Analytics with Spark: Patterns for Learning from Data at Scale*. Sebastopol, CA : O'Reilly Media, Inc., 2015

[SP15] SITTO, Kevin ; PRESSER, Marshall: *Field Guide to Hadoop: An Introduction to Hadoop, Its Ecosystem, and Aligned Technologies*. Sebastopol, CA : O'Reilly Media, Inc., 2015

[TSJ+09] THUSOO, Ashish ; SARMA, Joydeep S. ; JAIN, Namit ; SHAO, Zheng ; CHAKKA, Prasad ; ANTHONY, Suresh ; LIU, Hao ; WYCKOFF, Pete ; MURTHY, Raghotham: Hive - A Warehousing Solution Over a Map-Reduce Framework. In: *Proceedings of the VLDB Endowment, Volume 2 Issue 2* (2009), August

[Ven09] VENNER, Jason: *Pro Hadoop*. New York, NY : Apress Media LLC, 2009

[Whi12] WHITE, Tom: *Hadoop: The Definitive Guide*. Sebastopol, CA : O'Reilly Media, Inc., 2012

Statutory Declaration

I solemnly declare that I have drawn up this seminar paper without help from a third party and without having used other means than the indicated resources and auxiliaries and that I have made all passages recognizable as quotations that I have taken in letter or in spirit from the used sources. This paper has not been submitted in the same or similar form in another seminar.

Oldenburg, September 26, 2016 Uwe Kopplin

Open Source Alternatives to Apache Hadoop

Seminar Paper

Director:	Prof. Dr.-Ing. habil. Jorge Marx Gómez
Supervisor:	M. Eng. & Tech. Viktor Dmitriyev Dr.-Ing. Andreas Solsbach M.Sc. Ola Mustafa
Submitted by:	Carina Henkensiefken carina.henkensiefken@uni-oldenburg.de
Submission date:	02. December 2015

Contents

Abbreviations

API	Application Programming Interface
DFM	Digi Foto Maker
DFS	Distributed File System
ECL	Enterprise Control Language
ETL	Extract Transform Load
HPCC	High Performance Computing Cluster
JVM	Java Virtual Machine
SQL	Structured Query Language
YARN	Yet Another Resource Neogitator

Figures

1 Introduction

This seminar paper was written as part of a computer science project at the Carl von Ossietzky University of Oldenburg. This project belongs to the subject "Project Group", in which a group of about twelve master students working on a project topic within one year. The project begins with the seminar phase in which this work has been created.

The topic of the project and therefore the context of this work is the analysis of data using Apache Hadoop. The customer of the project are the Carl von Ossietzky University of Oldenburg and the CEWE Stiftung & Co. KGaA. The data are semi-structured XML files, which are created anonymously by working on the CEWE Photo stations. More details to the topic can be find in chapter 2.

This work deals with the selection process of the system, which has to be used for data analysis. In detail, this means that in the course of work, the existing system specification is compared with possible alternative systems. The system default is Apache Hadoop. This system is described in detail in chapter 3. Because Apache Hadoop is an open source system, the considered alternative systems also are open source systems. There are also commercial solutions and competing systems, which are, however, neglected in this work.

The other systems under consideration are Apache Spark, HPCC Systems and Hydra. Apache Spark is indeed often used as a supplement to Apache Hadoop, but it can also be considered as an alternative system. Further details can be seen in chapter 4. Apache Hadoop consists of many individual components and there are numerous alternatives to various components of the Apache Ecosystems. However, this work tries to find possible fully adequate alternatives to the complete Apache Hadoop ecosystem. The decision of the detailed consideration was fallen on HPCC Systems, which has a closer look in chapter 5, and Hydra, which is described in chapter 6.

After the individual systems have been described for themselves, the systems are compared with default system Apache Hadoop in chapter 7. The focus is on the comparison of the alternative systems with Apache Hadoop. It concludes with the results of this work in chapter 8.

2 Motivation

The topic of the project is to evaluate the data of the CEWE photo stations using Apache Hadoop.

CEWE is the European market leader in photofinishing. The headquarters is located in Oldenburg. CEWE sells its products in 24 different countries. The customers are both the stationary trade and the Internet trade. Overall, more than 30,000 trading partners are supplied by CEWE. (cf. [CEW15])

The CEWE photo stations, even Digi Photo Maker (shortly: DFM) called, are the kiosk stations of CEWE, that be placed at the point-of-sale of the trading partner such as Müller. With the DFM the customer can print directly his photo products locally or he can place an order, which is made in the laboratory of CEWE and sent to the branch, so that the customer can pick up the product. The direct printout on site is OnSite finishing (in short: OSF) called.

While the customer uses the Photo Station, his click behavior will be recorded anonymously and transferred to a semi-structured XML file to CEWE. In addition to the order data in the XML file, also information about the DFM are held like the temperature of the printer.

CEWE wants a possibility of evaluating these data to be able to perform, inter alia, "Near Time" analysis. Also an alarm system should be created to detect failures and downtimes of DFM early. CEWE hopes to be able to improve the customer software by a detailed analysis of the behavior of consumers.

In order to achieve these objectives and requirements, we have to use Apache Hadoop. Apache Hadoop is an ecosystem for analyzing big data. It has a huge number of different components. But why should we use Apache Hadoop and not another open source system for big data analysis? For finding out that, this paper will deal with possible alternatives to Apache Hadoop.

3 Apache Hadoop

Apache Hadoop is an open source framework. It is considered as one of the best large-scale and batch oriented analytics tools. It is a platform for multi-application processing and massively scalable data storage. Hadoop can work with structured as well as non-structured data. It is scalable. That means it can scale up from single servers to thousands of machines, each offering local computation and storage. The library itself is designed to detect and handle failures at the application layer. So there is a highly-available service. (cf. [Fou15])

For Hadoop you need an Apache v2.0 license. Hadoop is independent from the operating system. It works on Linux, Windows and Mac OS. Hadoop works on standard hardware.

3.1 Use Case

Hadoop is for used for big data. So the use of Hadoop makes just sense when one of the following 4 points is true: [Fro13]

- Size of entire data is too big and cannot be handled by current systems.
- Growth velocity of data is high and cannot be managed by current systems.
- Very complex data reporting and analysis from multiple structured and non-structured data sources.
- Data processing can be done offline so not real time.

3.2 Workings

The Apache Hadoop ecosystem has many different components for different tasks. The tasks are divided into data management, data access, data governance and integration, security, and operations. Figure 3.1 shows the different tasks.

Data Management

Data Management is for storing and processing data. The core technology for the efficient scale out storage layer is the Hadoop Distributed File System or short HDFS. HDFS runs on low-cost commodity hardware. It is a Java-based file system

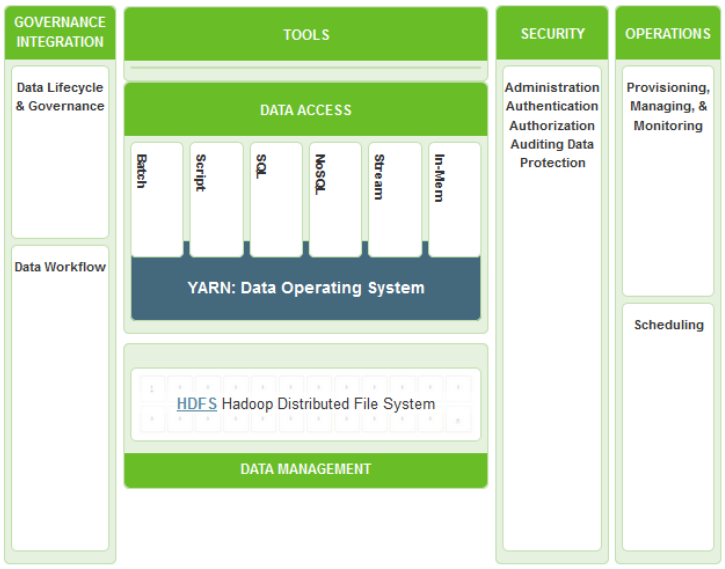


Figure 3.1: Architecture of Apache Hadoop [Horb]

that provides scalable and reliable data storage. Apache Hadoop YARN provides the resource management and pluggable architecture. This is needed for for permitting a plurality of data access methods to operate on data stored in Hadoop. So it is a requirement for Enterprise Hadoop. (cf. [Horb])

Data Access

Data Access describes how to interact with the data in many different ways. Apache Hadoop supports batch processing as well as real-time analysis. There are a variety of components for the different ways. Apache hive is said to be the most common data access technology. It based on the MapReduce-Framework. It enables easy data summarization and ad-hoc queries via an SQL-like interface. Another technology is Apache Pig. It offers scripting capabilities. For real-time processing you can use Apache Storm. Apache HBase provides NoSQL storage. (cf. [Horb])

Data Governance and Integration

This is used to load data easily and quickly. It also manages according to policy. Here also are different components to use for this task like Apache Flume, Apache Falcon or Apache Sqoop. (cf. [Horb])

Security

Security is needed to deal with the requirements of authentication, authorization, accounting and data protection. At every layer of the Hadoop stack from HDFS and YARN to Hive and other data access components it is available. Two examples for security components are Apache Knox and Apache Ranger. (cf. [Horb])

Operations

This is used to provision, manage, monitor and operate Hadoop clusters at scale. Here are also some components that can be used like Apache Ambari, Apache Oozie or Apache ZooKeeper. (cf. [Horb])

4 Apache Spark

Spark is an in-memory data analytics platform. So it requires a lot of memory. The in-memory distributed datasets optimize iterative workloads in addition to interactive queries. It uses the Scala programming language. Spark has a good application programming interface. (cf. [Fin11b], [van])

For Spark you need an Apache v2.0 license. Spark is independent from the operating system. It works on Linux, Windows and Mac OS. Spark works on standard hardware.

Apache Spark can be an alternative to Apache Hadoop. It also can be a complementary to Apache Hadoop. Spark has not use HDFS. It also works with other data platforms like Apache Cassandra. (cf. [Wei15])

4.1 Use Case

Apache Spark is not just used for big data. The focus of Spark is on high speed.

In 2014 Apache Spark achieved a new record at the Daytona-GraySort. The task is to sort 100 TByte. The previous record was 72 minutes from Apache Hadoop. Spark just needed 23 minutes. (cf. [Wei15])

4.2 Workings

Spark runs in-memory on a cluster. It is not bound to the Hadoop MapReduce two-storage paradigm. Spark can also read data directly from the Hadoop Distributed File System. It loads a process into memory and holds it there until it is told otherwise. (cf. [van])

Apache Spark consists of Spark core and a set of libraries. The core is the distributed execution engine. Spark supports the four dominant big data languages Scala, Python, Java and R. The APIs of these languages offer a platform for distributed ETL application development. Additional libraries enable various workloads for streaming, SQL, and machine learning. Spark also contains MLlib, a library that offers an increasing set of machine algorithms for common data science techniques such as classification, regression, collaborative filtering, clustering and dimensionality reduction. (cf. [Hora], [Wei15])

Figure 4.1 shows the architecture of Apache Spark.

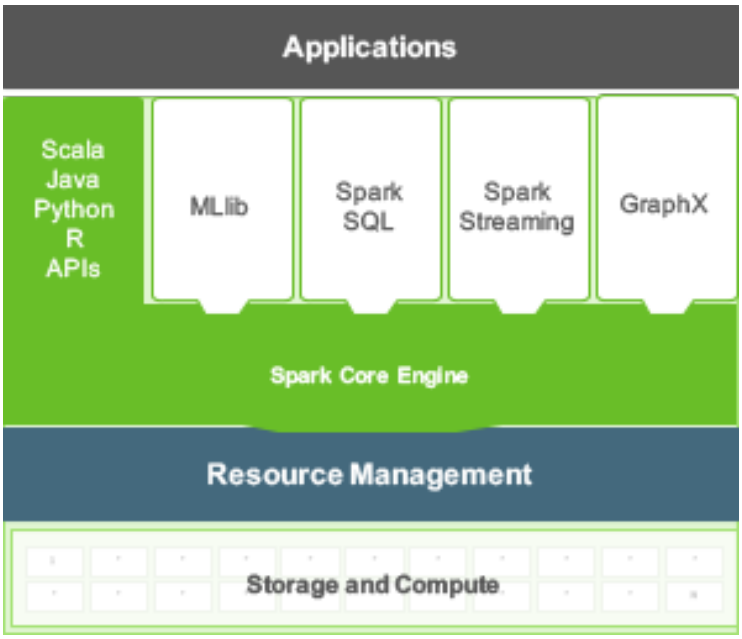


Figure 4.1: Architecture of Apache Spark [Hora]

5 HPCC Systems

HPCC stands for High Performance Computing Cluster. It is an open source cluster computing platform to solve Big Data problems. The platform offers flexibility and scalability. (cf. [Sys], [Lex15b])

For HPCC Systems you need an Apache v2.0 license. It just works on Linux distributions. (cf. [Lex15a])

5.1 Use Case

The HPCC Systems platform is a complete open source solution for big data processing and analytics. It includes the following functions: [Lex15a]

- Massively scalable data refinery environment for ingesting and transforming structured or unstructured big data from different sources
- Data delivery engine for real-time querying and data warehousing
- Powerful programming language optimized for big data processing
- Standards-based web services platform for end user access to big data queries

5.2 Workings

Its architecture and simple yet powerful own data programming language makes it a solution to solve data intensive computing needs. Figure 5.1 shows the HPCC Architecture.

HPCC has two Cluster: a back-end data refinery cluster for ingesting, refining, and transforming big data called Thor and a front-end data delivery cluster supporting high-performance online querying of processed data called Roxie. Data and indexes to support queries are pre-built on Thor and then deployed to Roxie. Both clusters run on standard hardware. The two systems work together to provide an end-to-end solution for big data processing and analytics. (cf. [Lex15a])

Thor distributed file system (short: Thor DFS) is record-oriented. It is optimized for big data ETL processes. It is responsible for consuming large datasets, transforming, linking and indexing that data. The data can be structured as well as non-structured. It functions as a distributed file system with parallel processing power spread across the nodes. A Thor cluster can scale from a single node to

thousands of nodes. It provides a massively parallel job execution environment. It utilizes a master-slave topology in which slaves provide localized data storage and processing power, while the master monitors and coordinates the activities of the slave nodes and communicates job status information. (cf. [Sys])

Roxie distributed file system (short: Roxie DFS) is index-based. It is optimized for concurrent query processing. It provides separate high-performance online query processing and data warehouse capabilities. Each Roxie node runs a server process and an agent process. The server process manages incoming query requests from users, allocates the processing of the queries to the corresponding agents across the Roxy cluster, collect the results, and returns the payload to the client. A Roxy cluster is fault resilient, based on data replication within the cluster. (cf. [Sys])

The programming language ECL stands for Enterprise Control Language. It is a programming language that is especially qualified for the manipulation of Big Data. ECL is a key element in the flexibility and capabilities of the HPCC Systems platform. It reaches big data processing and analysis objectives with a minimum of coding. The ECL compiler is cluster-aware and automatically optimizes code for parallel processing. Programmers need not to be worried about whether their code will be deployed on one node or hundreds of nodes. ECL code compiles into optimized C++ and can be easily extended using C++ libraries. (cf. [Sys])

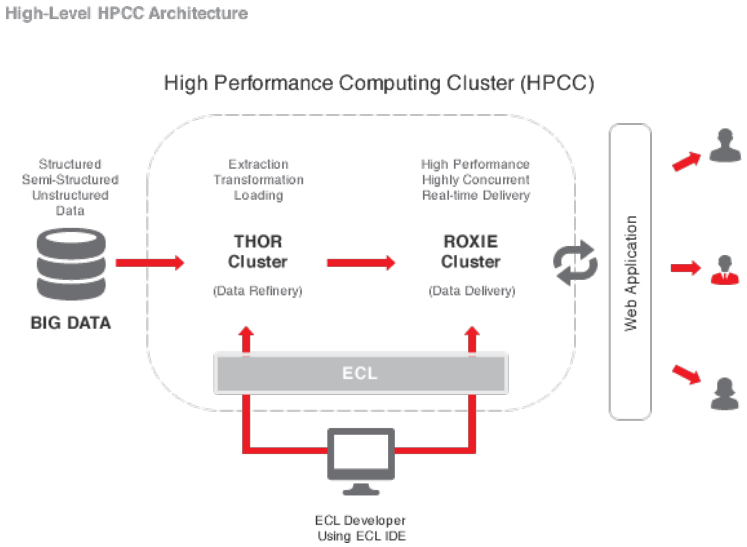


Figure 5.1: Architecture of HPCC Systems [Lex15a]

6 Hydra

Hydra is a distributed data processing and storage system. It supports real-time analytics and batch processing. It uses a tree-based data structure. So it can store and process data across clusters that may have thousands of nodes. The trees are aggregates, summaries, or transformations of the data. These trees can be used to explore (tiny queries), as part of a machine learning pipeline (big queries), or to support live consoles on websites (lots of queries). (cf. [van])

Hydra is available under an open source Apache license version 2.0. It can use HDFS, but also operates on native file systems.

6.1 Use Case

The developer of Hydra describes it like this "I have a lot of data, and there are a lot of things I could try to learn about – so many that I'm not even sure what I want to know. [...] Hydra tries to make it easy to reduce the number of dimensions." [Sri14]

Hydra has four main functional areas: [Abr11a]

- Capability to partition, store, and stream data in a distributed system.
- Software that takes an input data stream, processes that data (filtering, counting, membership detection, etc.) and then stores the batch output in Berkley DB JE which uses sequential writes and indexes the batch output for fast reads. Berkley DB JE is a database that provides a abstraction layer to the data structure.
- A set of software modules that enable ad-hoc queries against tree-based data structures.
- Command and control subsystem responsible for monitoring and executing jobs in the cluster.

6.2 Workings

Hydra has two standard modules, splitter and stream server.

The splitter needs an input stream from a data source. The data source may be of various types. The format of the source data from the stream is described to the

splitter. Each splitter consumes a sub-set of the data based on a hash function. The splitter then uses the stream format information and the job configuration script (usually a JSON file) to filter, transform, and create data elements which will be stored on the local file system. After either all available input data have been used up or the time limit has been reached, the files, which were saved by the splitter, are made available for other processes in the cluster via the stream servers. (cf. [Abr11a])

Stream servers offer access to the files stored on one node to another node in the cluster. The Stream Server transmits raw bytes to clients because they may be in a compressed format. This limits how often the data in the file is decoded and reduces the overall amount of data transmitted over the network. This approach does prevent the Stream Server from applying fine grained filter to the individual log lines. (cf. [Abr11a])

The foundation of Hydra is the tree building process. The TreeBuilder consumes data from one or more data sources (typically a set of stream servers) and then builds a tree based on the input data and a job configuration that defines the desired tree structure. The parent node is dynamically updated anytime new data is added to the tree. For this Hydra uses Stream-lib. Stream-lib is a library that makes it possible to count a large number of elements in a distributed system, find the top-k elements from huge lists of data, and run membership detection using bloom filters. (cf. [Abr11c])

Hydra uses a distributed query system. The query subsystem consists of two main components, QueryMaster and QuerySlave. A single cluster can have multiple QueryMasters and each processing node in the cluster will have one or more QuerySlaves. (cf. [Abr11b])

The QueryMaster takes a query, identifies which job the query is for, sends the query to the QuerySlaves that have data for the specified job. Then the master aggregates the responses from each slave into a single result set that is provided back to the client. (cf. [Abr11b])

QuerySlaves and the QueryMaster communicate via ZooKeeper and RabbitMQ. Because the system is not static, it is important to store the mapping of jobs to nodes in a distributed configuration management system like ZooKeeper. Data in the cluster can move from one node to another when the system re-balances data or when a node fails. This dynamic mapping enables the QueryMaster to only query the nodes that have data and dynamically change the set of nodes queried whenever the location of data in the cluster changes. (cf. [Abr11b])

The query is broken down into local and remote operations. Local operations are performed by the QueryMaster and remote operations are performed by the QuerySlaves. Remote operations allow the system to take advantage of the system

resources available in the cluster. Having each node sorting a subset of the data is much more efficient than having the QueryMaster sort the full set of unsorted data. Remote operations can also significantly reduce the amount of data transferred between the QuerySlave and QueryMaster. (cf. [Abr11b])

7 Comparison of the systems

After each system has been considered in isolation, the systems are now compared with each other. The focus is on the comparison of the respective system to Apache Hadoop.

7.1 Spark vs. Hadoop

Because Spark uses in-memory analytics instead of Hadoop's cluster file system approach, Spark has to be faster than Hadoop. For example, repeated access to the same data is faster at Spark's in-memory storage than at Hadoop. So for applications, where speed is an important role like real-time-analysis, Apache Spark in combination with Cassandra is a better alternative than Apache Hadoop. Moreover Spark is easier to program. Also, it has an interactive mode. (cf. [Fin11b], [van])

Hadoop is the default if it is about data warehouse and offline data analysis. It is for better for big data than Spark. Hadoop's MapReduce still has more security features than Apache Spark. Also MapReduce is still the best with one-pass ETL jobs. (cf. [van], [Wei15])

7.2 HPCC vs. Hadoop

HPCC is considered the most obvious and direct competitor to Hadoop. Hadoop has a better developer ecosystem as HPCC. On the other side, the HPCC stack can supply a more turnkey solution. The programming language ECL should be easier to get started with. So HPCC is easier to program and easier to administer than Hadoop. Also ECL is a language built specifically to tackle the complexities around big data problems. (cf. [Fin11a], [Lex15b])

HPCC has native binaries because it is coded in C++ and compiled to native binaries. Hadoop has no native binaries. It based on a JVM (Java Virtual Machine). cf. [Lex15b]

7.3 Hydra vs. Hadoop

Hydra can tackle some Big Data tasks that Hadoop struggles with. For example, Hydra is better at data exploration. Hadoop has stronger native support for very large, one-off joins. (cf. [van], [Sri14])

8 Conclusion

All in all it can be said that there are many alternatives to Apache Hadoop. They all have advantages and disadvantages. In general they differ in one point most of the other systems. This is for example in Apache Spark the high speed or Hydra's different approach with the tree structure.

So there is for each system to be part of territory in which it is particularly well. Thus the system to be used should be selected depending on the application area.

Literature

[Abr11a] ABRAMS, Matt ; ADDTHIS (Hrsg.): *Clearspring's Big Data Architecture, Part 1*. http://www.addthis.com/blog/2011/05/05/clearsprings-big-data-architecture-part-1/#.Vl75EOKTUZa. Version: 05.05.2011

[Abr11b] ABRAMS, Matt ; ADDTHIS (Hrsg.): *Clearspring's Big Data Architecture, Part 3*. http://www.addthis.com/blog/2011/06/07/clearsprings-big-data-architecture-part-3-2/#.VlndeuKTUZY. Version: 07.06.2011

[Abr11c] ABRAMS, Matt ; ADDTHIS (Hrsg.): *Clearspring's Big Data Architecture, Part 2*. http://www.addthis.com/blog/2011/05/18/clearsprings-big-data-architecture-part-2/#.Vl75R-KTUZZ. Version: 18.05.2011

[CEW15] CEWE STIFTUNG & CO. KGAA: *Herzlich Willkommen bei Europas führendem Fotoservice und innovativem Online-Druck Partner*. http://company.cewe.de/de/unternehmen.html. Version: 2015

[Fin11a] FINLEY, Klint: *3+ Alternatives to Apache Hadoop*. http://siliconangle.com/blog/2011/11/03/3-alternatives-to-apache-hadoop/. Version: 03.11.2011

[Fin11b] FINLEY, Klint: *Another Hadoop Alternative: Spark*. http://siliconangle.com/blog/2011/11/03/another-hadoop-alternative-spark/?angle=services. Version: 03.11.2011

[Fou15] FOUNDATION, The Apache S.: *Welcome to Apache Hadoop!* https://hadoop.apache.org/. Version: 31.10.2015

[Fro13] FROMDEV, Sachin: *10 Hadoop Alternatives - When To Use Hadoop*. http://www.fromdev.com/2013/06/hadoop-when-to-use.html. Version: 2013

[Hora] HORTONWORKS: *What Apache Spark Does*. http://de.hortonworks.com/hadoop/spark/

[Horb] HORTONWORKS: *What is Apache Hadoop?* http://hortonworks.com/hadoop/

[Lex15a] LEXISNEXIS: *HPCC Systems Platform*. https://hpccsystems.com/download/hpcc-platform. Version: 2015

[Lex15b] LEXISNEXIS: *Why HPCC Systems?* https://hpccsystems.com/why-hpcc-systems. Version: 2015

[Sri14] SRINIVAS, Rags ; INFOQ (Hrsg.): *Hydra Takes On Hadoop*. http://www.infoq.com/news/2014/04/hydra. Version: 11.04.2014

[Sys] SYSTEMS, HPCC ; ALTERNATIVETO (Hrsg.): *HPCC Systems*. http://alternativeto.net/software/hpcc-systems/

[van] VAN RIJMENAM, Mark: *Big Data Hadoop Alternatives: What They Offer and Who Uses Them*. https://datafloq.com/read/Big-Data-Hadoop-Alternatives/1135

[Wei15] WEISS, Harald ; COMPUTERWOCHE (Hrsg.): *Apache Spark versus Hadoop*. http://www.computerwoche.de/a/apache-spark-versus-hadoop,3096930. Version: 08.04.2015

Final Declaration

Hereby I confirm that I have made my seminar paper "Open Source Alternatvies to Apache Hadoop" independently and without outside help, and that I specially marked all of other authors literally taken over bodies as well as adhere to the ideas of other authors closely investing versions of my work and the sources have quoted.

Oldenburg, den September 26, 2016 Carina Henkensiefken

Analytische Kapazitäten des Apache Hadoop Ecosystem

Seminararbeit

Themenersteller:	Prof. Dr.-Ing. habil. Jorge Marx Gómez
Betreuer:	M. Eng. & Tech. Viktor Dmitriyev Dr.-Ing. Andreas Solsbach M.Sc. Ola Mustafa
Vorgelegt von:	Felix Kruse Felix.Kruse@uni-oldenburg.de
Abgabetermin:	2. Dezember 2015

Inhaltsverzeichnis

Abkürzungen

DAG	Directed Acylic Graph
ELT	Extract, Load and Transform
ETL	Extract, Transform and Load
HDFS	Hadoop Distributed File System
RDD	Resilient Distributed Dataset
SQL	Structured Query Language
UDF	User defined Function
YARN	Yet Another Resource Negotiator

Abbildungen

Tabellen

1 Einleitung

Daten gewinnen für Unternehmen immer mehr an Bedeutung. Sie werden als vierter Produktionsfaktor neben Arbeitskraft, Rohstoffen und Kapital gesehen (vgl. [Bit12] S. 7) oder auch als Rohstoff des 21.Jahrhunderts bezeichnet (vgl. [BDS15] S. 12). Laut einer Statistik von Statista sollen im Jahr 2015 8.5 Exabyte an Daten produziert werden. Im Jahr 2020 sollen bereits um die 40 Exabyte an Daten generiert werden (vgl. [Sta15])

Diese enormen Datenmassen bergen enorme Potenziale hinsichtlich der Analyse zur Generierung von Informationen und Wissen. Big Data umfasst strategische Ansätze, IT-Architekturen, Methoden und Verfahren, um Informationen und Wissen aus den exponentiell wachsenden, vielfältigen Daten zu generieren. Der Begriff Big Data ist grundlegend durch Datenvolumen (Volume), Datenvielfalt (Variety) und Geschwindigkeit (Velocity) charakterisiert - die sogenannten drei Vs. Mit Datenvolumen sind Volumen in Bereichen von Terabyte bis hin zu Yottabyte gemeint. Die Datenvielfalt beinhaltet strukturierte Daten (bspw. Daten aus relationalen Datenbanken), semistrukturierte Daten (bspw. XML oder JSON) und unstrukturierte Daten (bspw. Bild- und Videodateien). Mit der Geschwindigkeit ist sowohl die Geschwindigkeit der Datenerzeugung und als auch die Geschwindigkeit der Datenanalyse gemeint. (vgl. [Bit12] S. 19 und S. 21 und [BDS15] S. 12)

Big Data kann als spezielles Anwendungsfeld der Business Intelligence angesehen werden. Generell folgen beide Domänen dem Konzept Daten zu allokieren, zu laden, zu verarbeiten, gegebenenfalls zu veredeln und abschließend zu analysieren. Dabei liegt der Fokus von Big Data auf der Skalierbarkeit hinsichtlich der drei Vs. (vgl. [BDS15] S. 12 und [Fis15] S. 30)

Um die analytischen Potenziale von Big Data zu nutzen, sind neue Technologien wie das Hadoop Ecosystem und NoSQL-Datenbanken entstanden. (vgl. [MSH14] S. 2 und [Bit14] S. 14) Diese Seminararbeit betrachtet das Hadoop Ecoysstem hinsichtlich seiner analytischen Kapazitäten. Zunächst wird der Begriff Business Analytics in Kapitel 2 erklärt. In Kapitel 3 werden die wesentlichen Herausforderungen an analytische Systeme durch Big Data skizziert. In Kapitel 4 wird dann das Hadoop Ecosystem - eine Big Data Lösung - vorgestellt. Der Fokus liegt auf den analytischen Komponenten des Hadoop Ecosytem. In Kapitel 5 werden die Einsatzmöglichkeiten der in Kapitel 4 beschriebenen analytischen Komponenten an-

hand von vier Aufgabenbereichen beschrieben. Die Seminararbeit endet mit einem Fazit in Kapitel 6.

2 Business Analytics

Der Begriff Business Analytics fasst die Bereiche descriptive Analytics, predictive Analytics und prescriptive Analytics zusammen. Synonym verwendete Begriffe sind Advanced Analytics oder analytische Informationssysteme. Folgende Aufgaben fallen in die Teilbereiche der Business Analytics (vgl. [Hum14]):

Descriptive Analytics: Die klassische Aufgabe der BI ist die descriptive (dt. deskriptive oder auch beschreibende) Analytics. Aufgabenbereich ist die systematische Auswertung von Vergangenheitsdaten und Bereitstellung eines adhoc Berichtswesens. Die Auswertungen können mit Online Analytical Processing
(OLAP), statistischen Verfahren, Data Mining und Text Mining durchgeführt werden. Beispielsweise könnten die Absatz- und Produktzahlen (der Vergangenheit) pro Zeiteinheit pro Vertriebseinheit in einer Produkthierarchie dargestellt werden und Kunden mittels Data Mining Verfahren in Kundencluster eingeteilt werden.

Predictive Analytics: Mit Hilfe von Data Mining Verfahren kann predictive (dt. prädikative oder auch vorhersagende) Analytics durchgeführt werden. Es wird versucht mit dem Vergleich von historischen und aktuellen Daten Prognosen zu erstellen. Dies können beispielsweise Prognosen zu Kaufwahrscheinlichkeiten, Kündigungswahrscheinlichkeiten oder Absatz- und Umsatzprognosen sein.

Prescriptive Analytics: Prescriptive (dt. präskriptive) Analytics geht einen Schritt weiter. Es werden nicht nur Muster gesucht und Informationen oder Wissen generiert, sondern je nach Anwendungsfall bestimmte (Lösungs-)Aktionen vorgeschlagen. Ziel ist die Automatisierung von Entscheidungsprozessen. Um dies zu ermöglichen, sollten Geschäftsregeln und Modelle der möglichen Handlungsaktionen mit den Analysen verbunden werden.

3 Herausforderungen an analytische Systeme durch Big Data

Um die in der Einleitung beschriebenen analytischen Potenziale von Big Data nutzen zu können, müssen die Daten für die Analyse bereitgestellt werden. In klassischen Business Intelligence Lösungen werden die Daten mittels ETL-Prozessen (Extract, Transform and Load) in ein Data Warehouse überführt und somit der Analyse zur Verfügung gestellt. Hierzu werden die Daten in der Regel normalisiert, validiert und in eine Struktur gebracht, um anschließend in einer relationalen Datenbank persistiert zu werden. Die Datenintegration durch ETL-Prozesse setzt voraus, dass im Vorfeld bekannt ist, was analysiert werden soll und welche Daten dafür benötigt werden. Dieses Vorgehen zieht eine geringe Flexibilität des Datenmodells bezüglich der Erweiterung nach sich. (vgl. [Bit14] S. 27, S. 93-96 und S. und [FUJ13] S. 6)

Die einhergehenden Herausforderungen von Big Data liegen in den 3Vs, die Big Data charakterisieren. Das zunehmende Datenvolumen und die Geschwindigkeit in der die Daten erzeugt werden lassen ein Transformieren vor dem Laden nicht mehr zu. Ein typisches Big Data Vorgehen ist die Daten „as is“ vorerst zu speichern und im Anschluss zu transformieren. Dieser Wandel führt von den klassischen ETL-Prozessen zu ELT-Prozessen. Zusätzlich stellen die nun auch semi- und unstrukturierten Daten die klassischen ETL-Werkzeuge vor Herausforderungen, da diese für den Umgang mit strukturierten Daten ausgelegt sind. Die ETL-/ELT-Tools sollten im Kontext von Big Data mit lose definierten Daten und sich ändernden Schemata umgehen können. (vgl. [Bit14] S. 94)

Ein weiteres Umdenken findet im Kontext von Big Data Analytics statt. Um den Einsatz von Verfahren wie Machine Learning, Data Mining und Predictive Analytics anzuwenden, ist es notwendig möglichst große Mengen von Daten über einen längeren zeitlichen Verlauf bereitstellen zu können. Die für die Analyse relevanten Daten werden in der Regel während des Analyse Prozesses durch mehrere Iterationen identifiziert. Das bedingt, dass die Daten zunächst in ihrer semi- oder unstrukturierten Form gespeichert werden. Für die Speicherung der ungefilterten Daten kann das HDFS von Hadoop (siehe 4.1) genutzt werden. (vgl. [Bit14] S. 95) Dieses Vorgehen der Datenspeicherung wird häuft als Landing Zone, Data Lake

oder Data Reservoir bezeichnet (vgl. [Ehr15] S. 17, S. 20). Die Datentransformationen werden durch die verschiedenartig strukturierten schnell aufkommenden Datenmengen aufwändiger (vgl. [Bit14] S. 95). Damit die Daten analysiert werden können, ist auch im Big Data Analytics Kontext eine gewisse Strukturierung der Daten notwendig (vgl. [Fis15] S. 31). Diese Herausforderungen versucht das Hadoop Ecosystem mit seinen Kernkomponenten und den analytischen Komponenten zu bewältigen.

4 Hadoop Ecoystem

4.1 Kernkomponenten des Hadoop Ecosystem

Hadoop ist ein Top-Level-Projekt der Apache Foundation. Hadoop ist ein open-source Framework für zuverlässige und parallele Datenzugriffe auf skalierbaren verteilten Serverclustern. Die open-source Technologie bietet die Möglichkeit zur Speicherung und Verarbeitung(Analyse) von sehr großen strukturierten, semistrukturierten und/oder unstrukturierten Datenmengen. Hadoop ist kein einzelnes Projekt, sondern setzt sich aus einer Vielzahl von aufeinander aufbauenden Projekten zusammen. Die Kernkomponenten auf denen alle Projekte und somit das gesamte Hadoop Ecosystem aufbauen, werden im Folgenden kurz beschrieben. (vgl. [Bit14] S. 35-41 und [Apa15h])

Hadoop Distributed File System: Das Hadoop Distributed File System (HDFS) ist ein verteiltes Dateisystem. HDFS dient zur zuverlässigen Speicherung und Verwaltung von großen unterschiedlich strukturierten Datenmengen. Das HDFS kann auf herkömmlichen Servern betrieben werden, sodass die großen Datenmengen verhältnismäßig günstig gespeichert werden können. Das HDFS ist in einer Master-Slave-Architektur aufgebaut. Der Master, der sogenannte NameNode, beinhaltet alle Metadaten wie Verzeichnisstrukturen und Ablageorte der Dateien. Die Slaves sind die sogenannten DataNodes und übernehmen die Speicherung der Dateien auf den Servern. Dateien werden dabei in beliebig große Blöcke gesplittet und redundant auf die Server verteilt. (vgl. [Apa13] und [Bit14] S. 35-41 und [Apa15h])

Hadoop YARN: YARN steht für „Yet Another Resource Negotiater“. YARN kann als Betriebssystem von Hadoop angesehen werden, welches die Ressourcen verwaltet. Es bietet ein Framework für das Job-Sheduling und das Resourcenmanagement des Hadoop Clusters. Durch YARN sind vielfältige Datenverarbeitungsmöglichkeiten entstanden, wie interactive SQL, Realtime Streaming, Data Science und Batch Processing. Damit öffnet YARN neue Möglichkeiten für die Analyse mit dem Hadoop Ecosystem. (vgl. [Bit14] S. 35-41 und [Apa15h] und [Hor15b])

Hadoop MapReduce: MapReduce ist ebenfalls ein open-source Framework. Dieses Framework basiert auf YARN und dient der verteilten und parallelen Verarbeitung auf großen unterschiedlich strukturierten Datenmengen, die im HDFS gespeichert sind. (vgl. [Apa15h])

Die beschriebenen Kernkomponenten des Hadoop Ecosystem sind in Abbildung 4.1 dargestellt. Neben den Kernkomponenten sind in Abbildung 4.1 weitere Komponenten des Hadoop Ecosystem dargestellt. Die verschiedenen Komponenten können bestimmten Aufgabenbereichen zugeordnet werden. Zu diesen Aufgabenbereichen zählen Datenspeicherung, Datenzugriff, Datenanalyse und Systemmonitoring. (vgl. [Kum14]) In den Abschnitten 4.2.1, 4.2.4, 4.2.5, 4.2.6 werden die Komponenten Hive, Pig, Mahout und Spark näher beschrieben (siehe Abb. 4.1).Diese Komponenten gehören zu den Aufgabenbereichen Datenzugriff und Datenanalyse des Hadoop Ecosystem.

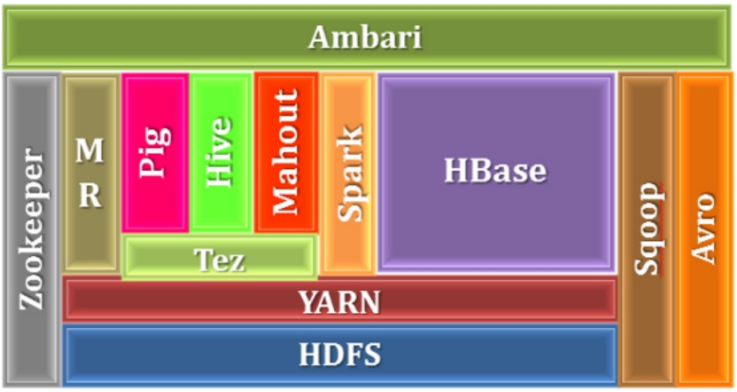


Abbildung 4.1: Überblick über einige Komponenten des Hadoop Ecosystem ([Kum14])

4.2 Analytische Komponenten des Hadoop Ecosystem

4.2.1 Hive

Hive ist ein Tool aus dem Hadoop Ecosystem und ist auf den Kernkomponenten von Hadoop aufgebaut (siehe Abb. 4.1). Hive wird als Data Warehouse Software bezeichnet. Es ermöglicht das Abfragen und Bearbeiten von großen und verteilt gespeicherten Datenmengen. Hierzu nutzt Hive eine SQL ähnliche Abfragesprache HiveQL. Es ist möglich mit Hilfe von Hive Daten zu extrahieren, zu transformieren und zu laden, wie im klassischen ETL-Prozess. Dadurch ist es möglich den verschiedenartig strukturierten Daten eine Struktur aufzulegen. Hive hat Zugriff auf Daten, die im HDFS liegen oder in anderen Datenspeichersystemen bspw. in

HBase. HiveQL kann über MapReduce, Tez ([Apa15g]) oder Spark (siehe 4.2.6) ausgeführt werden. Somit bietet Hive(QL) für analytische Zwecke ähnliche Möglichkeiten wie SQL. HiveQL kann für die interaktive Analyse und Erforschung der Daten genutzt werden. Weiterhin bietet HiveQL ebenso wie SQL neben den klassischen Funktionen wie *SUM(), AVG(), MIN(), MAX()* auch analytische Funktionen, wie *OVER(), PARTITION BY, CUBE, ROLLUP*. Eine weitere analytische Möglichkeit bietet Hive mit den User-Defined Functions (UDFs). In den UDFs können eigene analytische Funktionalitäten implementiert werden, ähnlich wie die Funktionen und Prozeduren in SQL. Zudem bietet Hive viele Funktionalitäten in bestehenden UDFs an, wie zum Beispiel XPath, um XML zu verarbeiten. Mit Hive 2.0 soll Hive HPL/SQL eingeführt werden. HPL/SQL ist ein open source Tool, dass prozedurales SQL für Apache Hive, SparkSQL und auch Impala implementiert. Mit HPL/SQL könnte zukünftig mehr Business Logik und analytische Funktionalität mittels Hive implementiert werden. (vgl. [Apa15g] und [Apa15d] und [HPL15])

Mit Hivemall existiert eine Hive UDF Bibliothek, die Machine Learning Algorithmen bereitstellt. (vgl. [Mak15])

4.2.2 Impala

Impala ist eine open source Software von dem Hersteller Cloudera. Es ist eine massive parallel processing (MPP) SQL Query-Engine, die im Hadoop Ecosystem angesiedelt ist. Hive und Impala ähneln sich stark. Impala nutzt ebenfalls das HDFS oder bspw. HBase und greift auf YARN zu. Im Unterschied zu Hive greift Impala bei Abfragen direkt auf das HDFS zu, statt über MapReduce oder Yarn. Mit diesem Konzept soll die Performance der Abfragen optimiert werden. Die Impala SQL Syntax gleicht der von Hive. (vgl. [Clo15] und [PZ15] und [Sak14])

Mit Impala können die gleichen analytischen Funktionalitäten wie mit Hive umgesetzt werden. Es kann zur interaktiven Analyse und Erforschung der Daten benutzt werden.

4.2.3 HAWQ

Apache HAWQ ist ein Apache Projekt in der „incubation“ Phase. Es ist von Pivotal entwickelt und an Apache übergeben worden. Apache HAWQ kann als Konkurrenz von Impala und Hive angesehen werden. Es kann ins Hadoop Ecosystem integriert werden und ist eine MPP SQL Query Engine. Zudem bietet es die Funktionalität von MPP advanced analytics. HAWQ nutzt den ANSI SQL Standard und greift direkt auf das HDFS oder bspw. HBase zu. Durch die Unterstützung des Standard SQL besteht laut HAWQ ein Vorteil in der Integration sämtlicher auf

SQL basierender BI-Tools zur Visualisierung. Damit bietet HAWQ bis hierhin die gleichen analytischen Möglichkeiten wie Hive und Impala. (vgl. [Haw15])

Der signifikante Unterschied zu Hive und Impala liegt darin, dass HAWQ das Apache Projekt MADlib unterstützt (vgl. [Haw15]). Apache MADlib ist eine open source Bibliothek für skalierbare Machine Learning Algorithmen, die in der Datenbank ausgeführt werden. MAD steht dabei für ([Piv15]):

Magnetic: „Designed to draw different types of data sources and data scientists to a single environment where best practices on analytics can be shared."

Agile: „Built for fast, exploratory and iterative analytics where lightweight modeling is possible and integration of new data is extremely easy."

Deep: „An environment where advanced machine learning and statistical algorithms are supported."

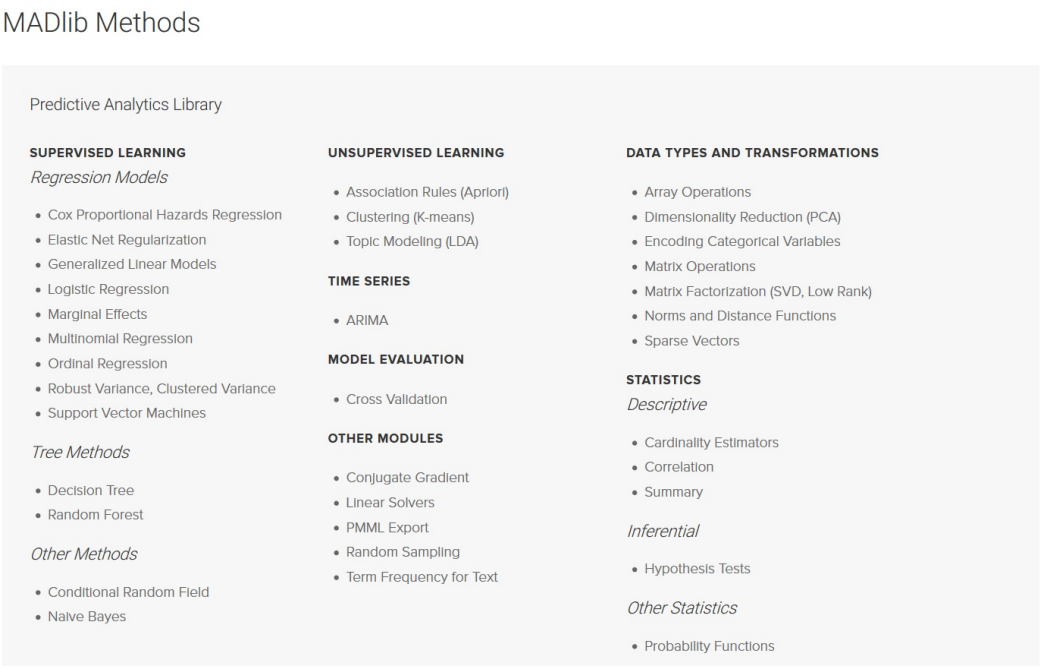


Abbildung 4.2: Überblick über die MADlib Methoden ([Piv15])

Die MADlib Algorithmen werden mittels SQL ausgeführt und sind deshalb einfach zu benutzen. In Abbildung 4.2 ist eine Übersicht über die Funktionen der MADlib abgebildet. (vgl. [Piv15])

Damit lassen sich mit HAWQ nicht nur die klassischen analytischen (Teil der deskriptive Analytics) Funktionalitäten umsetzen, sondern auch die erweiterten analytischen Funktionalitäten wie Data Mining. Mit Data Mining kann dann predictive und prescriptive Analytics realisiert werden.

Neben den klassischen Data Mining Methoden bietet die MADlib auch Methoden zur Datentransformation und Datensichtung (siehe Abb. 4.2). Für die Datensichtung ist bspw. die Methode *Summary* interessant. Die *Summary* Methode bietet zur ersten Datensichtung eine analytische Übersicht über die Ausprägungen der Werte für die Attribute einer Tabelle. Sie kann je Attribut der Tabelle bspw. das Minimum, das Maximum, die Anzahl der Werte, die Anzahl der fehlenden Werte oder die Anzahl der eindeutigen Wertausprägungen ermitteln (vgl. [MAD15]).

4.2.4 Pig

Apache Pig gehört zum Hadoop Ecosystem. Pig ist ein Tool zum Analysieren von großen Datenmengen. Hierzu bedient sich Pig einer eigenen Script Sprache Pig Latin. Pig Skripte werden in MapReduce Jobs umgewandelt und können parallel auf dem HDFS ausgeführt werden. Mit Pig können eigene Funktionen, sogenannte user defined functions (UDF), erstellt werden. Es existieren bereits viele Pig Bibliotheken, in denen viele UDFs implementiert sind. Ein Beispiel ist die Apache DataFU Bibliothek (vgl. [Apa15c]). DataFU bietet statistische Funktionen, um bspw. den Median oder die Variance zu berechnen oder die *COALESCE* Funktion, die es ebenfalls im SQL Standard gibt. (vgl. [Apa15a] und [Apa11] and [Apa15c])

Mit Hilfe von Pig können Daten analysiert und erforscht werden. Dabei agiert Pig im Bereich descriptive Analytics ohne Data Mining.

4.2.5 Mahout

Apache Mahout ist eine Machine Learning Bibliothek, dessen Algorithmen auf Hadoop mit MapReduce oder Spark ausgeführt werden (vgl. [Mah15b]). Mahout kann Bestandteil des Hadoop Ecosystem sein und auf Daten im HDFS zugreifen. Es unterstützt hauptsächlich die analytischen Anwendungsfälle Collaborative Filtering, Clustering, Classification und Frequent itemset mining. (vgl. [Hor15a])

Somit kann Mahout in den descriptive, predictive und prescriptive Analytics Bereichen zum Einsatz kommen. In Abbildung 4.3 sind einige Mahout Algorithmen aufgeführt, die mit MapReduce lauffähig sind.

4.2.6 Spark

Spark ist ein Top-Level Apache Projekt und kann in das Hadoop Ecosystem integriert werden (siehe Abb. 4.1). Apache Spark ist ein hoch skalierbares in Memory arbeitendes Tool zur Datenverarbeitung- und analyse. Es besitzt eine eigene Directed Acylic Graph (DAG) Engine zur Ausführung der Anwendungen. Die Anwendungen können in Java, Python, Scala oder R implementiert werden. Spark läuft

Algorithm	Category	Description
Distributed Item-based Collaborative Filtering	Collaborative Filtering	Estimates a user's preference for one item by looking at his/her preferences for similar items
Collaborative Filtering Using a Parallel Matrix Factorization	Collaborative Filtering	Among a matrix of items that a user has not yet seen, predict which items the user might prefer
Canopy Clustering	Clustering	For preprocessing data before using a K-means or Hierarchical clustering algorithm
Dirichlet Process Clustering	Clustering	Performs Bayesian mixture modeling
Fuzzy K-Means	Clustering	Discovers soft clusters where a particular point can belong to more than one cluster
Hierarchical Clustering	Clustering	Builds a hierarchy of clusters using either an *agglomerative* "bottom up" or *divisive* "top down" approach
K-Means Clustering	Clustering	Aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean
Latent Dirichlet Allocation	Clustering	Automatically and jointly cluster words into "topics" and documents into mixtures of topics
Mean Shift Clustering	Clustering	For finding modes or clusters in 2-dimensional space, where the number of clusters is unknown
Minhash Clustering	Clustering	For quickly estimating similarity between two data sets
Spectral Clustering	Clustering	Cluster points using eigenvectors of matrices derived from the data
Bayesian	Classification	Used to classify objects into binary categories
Random Forests	Classification	An ensemble learning method for classification (and regression) that operate by constructing a multitude of decision trees
Parallel FP Growth Algorithm	Frequent Itemset Mining	Analyzes items in a group and then identifies which items typically appear together

Abbildung 4.3: Überblick der Mahout Algorithmen lauffähig mit MapReduce ([Mah15a])

auf YARN und kann auf die klassischen Hadoop Datenspeicher wie das HDFS oder HBase zugreifen. (vgl. [Apa15b] und [BD15])

Spark bietet die Komponenten Spark SQL, Spark Streaming, MLib (Machine Learning) und GraphX (siehe Abb. 4.4), die folgendes leisten (vgl. [Apa15b] und [BD15]):

Spark SQL: Mit Spark SQL kann auf strukturierte Daten in Spark Anwendungen zurückgegriffen werden. Auf Spark SQL kann aus Java, Scala, Python und R Spark-Anwendungen zugegriffen werden. Zudem bietet Spark einen Hive QL Interpreter, der den Zugriff auf sämtliche Hive Ressourcen ermöglicht. Spark bietet ebenso Schnittstellen über JDBC und ODBC zu anderen Datenbanken und BI-Tools.

Spark Streaming: Spark bietet mit dem Modul Spark Streaming die Möglichkeit

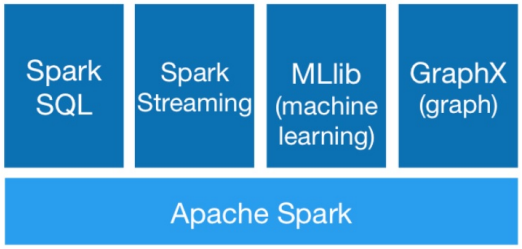


Abbildung 4.4: Überblick der Spark Komponenten ([Apa15b])

Streams (kontinuierlich sendende Datenquellen) zu verarbeiten. Ein sogenannter Discretized Streams (DStreams) beinhaltet eine Folge von Resilient Distributed Datasets (RDDs). Auf diesen DStreams können Transformationen und Aktionen ausgeführt werden, die in Micro-Batches (meist im Sekundentakt) verarbeitet werden. Als Streaming Quelle kann bspw. das HDFS, Flume oder Kafka benutzt werden. Die verarbeiteten Streaming Daten können bspw. in das HDFS, andere Datenbanken oder Dashboards zurückgeführt werden. (vgl. [Apa15f])

MLib: Die Spark MLib ist eine Bibliothek für Machine Learning Algorithmen. Die MLib kann aus Java, Scala, Python und R aufgerufen werden. Unter Python arbeitet Spark mit dem NumPy Package zusammen, dass ein n-dimensionales Array zur Verfügung stellt. Im folgenden sind einige Algorithmen und Funktionalitäten der Spark MLib genannt (vgl. [Apa15e]):

- logistic regression and linear support vector machine (SVM)
- classification and regression tree
- clustering via k-means, Gaussian mixtures (GMM), and power iteration clustering
- principal component analysis (PCA)
- frequent itemset mining via FP-growth and association rules
- summary statistics and hypothesis testing
- ...

GraphX: Die GraphX Bibliothek bietet Funktionalitäten zum Erstellen von Graph-Algorithmen.

Apache Spark bietet mit den beschriebenen Funktionalitäten eine Reihe von analytischen Möglichkeiten. Es kann ebenso wie die SQL-basierten Tools Hive, Impala und HAWQ mit dem eigenen Spark SQL Modul verschiedenartig strukturierte Daten zur Analyse aufbereiten. Die aufbereiteten Daten lassen sich dann mit Spark SQL weiter analysieren und erforschen. Da Spark einen HiveQL Interpreter besitzt, ermöglicht es den vollen funktionalen Umfang von Hive. Durch die Spark

MLib können neben den klassischen analytischen Methoden, wie dem Reporting, auch erweiterte analytische Methoden aus den Bereichen descriptive, predictive und prescriptive Analytics angewandt werden. Mit dem Spark Streaming Modell weitere analytische Funktionalität zur Verfügung, die der near-realtime oder realtime Analyse. Dies umfasst sowohl near-realtime oder realtime Reporting und Statistiken als auch near-realtime und realtime Data Mining und Machine Learning. (vgl. [Apa15b] und [BD15] und [Apa15f])

4.2.7 RapidMiner Radoop

RapidMiner Radoop ist ein Tool zum Verarbeiten und Analysieren von Big Data auf Hadoop. Es kann auf verschiedene Komponenten des Hadoop Ecosystem zugreifen. Zum einen besteht der Datenzugriff und zum anderen die Integration der Sicherheits- und Berechtigungstools aus dem Hadoop Ecosystem. RapidMiner Radoop ist eine Erweiterung des RapidMiner Studios und/oder des RapidMiner Servers. Es bietet eine Plattform zum Erstellen und Betreiben von ETL-Prozessen, Datenanalysen und Machine Learning im Hadoop Ecosystem. RapidMiner Radoop bietet eine graphische Oberfläche (siehe Abb. 4.5) in der die ETL oder analytischen Prozesse modelliert werden können. Die im RapidMiner modellierten Prozesse werden durch den RapidMiner in Hadoop übersetzt. Der RapidMiner Radoop unterstützt Hive, MapReduce, Spark, Mahout und Pig. Die Prozesse werden dabei im entsprechenden Hadoop Cluster ausgeführt, um die Performance des Hadoop Clusters zu nutzen. RapidMiner Radoop beinhaltet viele Operatoren, um die Daten vorzubereiten. Es beinhaltet viele der Algorithmen von Spark und Mahout und bietet sogenannte Script Operatoren in denen beispielsweise SparkR oder PySpark Skripte in den Workflow eingebunden werden können. (vgl. [Rap14] und [Rap15c] und [Rap15a] und [Rap15b])

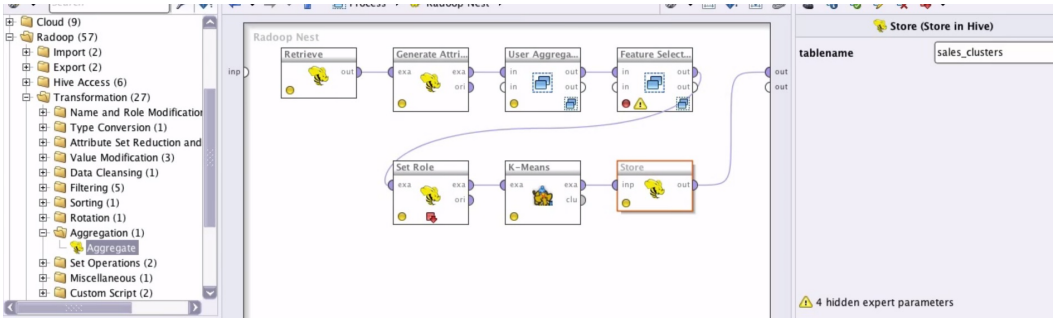


Abbildung 4.5: Ausschnitt der grafischen Oberfläche des Tools Radoop von RapidMiner ([Rap14])

In Abbildung 4.5 ist ein Workflow des RapidMiner Radoop Tools zu sehen. Der

Workflow läuft vollständig in Hadoop. Es werden Daten vorbereitet, die anschließend mit einem K-Means Algorithmus von Mahout geclustert werden. Das Ergebnis wird in eine Hive Tabelle zurück in Hadoop geschrieben. Im linken Fenster des Tools befinden sich alle Operatoren, die in den Workflow eingebunden werden können. Im mittleren Fenster wird der Workflow modelliert (per drag and drop) und dargestellt. Im rechten Fenster können die Eigenschaften der einzelnen Workflow Operatoren festgelegt werden. (vgl. [Rap14])

Das RapidMiner Radoop Tool kann im analytischen Kontext von Big Data und Hadoop vielfältig eingesetzt werden. Es bietet Möglichkeiten zur Datenaufbereitung, einfachen Datenanalyse und zur komplexen Datenanalyse mittels Data Mining und Machine Learning. Zusätzlich bietet es noch Visualisierungsmöglichkeiten von Daten und Ergebnissen.

5 Einsatzmöglichkeiten der analytischen Komponenten des Hadoop Ecosystem

In diesem Kapitel sollen die Einsatzmöglichkeiten der in Abschnitt 4.2 beschriebenen analytischen Komponenten des Hadoop Ecosystem aufgezeigt werden. Hierzu werden die analytischen Komponenten in Tabellen 5.1, 5.2 und 5.3 anhand von vier Aufgabenbereichen gegenübergestellt. Folgende vier Aufgabenbereiche werden dabei betrachtet:

Datenverständnis: In diesem Aufgabenbereich sollen die Tools Unterstützung bieten, um sich mit den Daten vertraut zu machen, erste Erkenntnisse über die Daten zu gewinnen, die Datenqualität zu prüfen oder Daten zu entdecken, die für Analysen nützlich sein könnten.

Datentransformation: Im Aufgabenbereich der Datentransformation sollen die Tools Unterstützung bieten, um die Daten für die Analysemethoden zu bereinigen und zu transformieren. Zusätzlich soll in diesem Aufgabenbereich die Funktionalität betrachtet werden, dass un- und semistrukturierte Daten in eine strukturierte Form für weitere Transformationen und für die Analyse gebracht werden.

Einfache Analysen: Zu diesem Aufgabenbereich zählen die Analysen aus dem Bereich deskriptive Analytics ohne Data Mining und Machine Learning.

Data Mining & Machine Learning: In diesem Aufgabenbereich sollen die Tools Unterstützung bieten, um Data Mining und Machine Learning zu ermöglichen.

Tool	Aufgabe			
	Datenverständnis	**Datentransformation**	**einfache Analysen**	**Data Mining & Machine Learning**
Hive	Sind die Daten in Hive-Tabellen überführt, kann mit den umfangreichen Funktionen und Bibliotheken in Hive ein Datenverständnis generiert werden.	1. Sind die un- oder semistrukturierten Daten in Hive Tabelle abgelegt, können diese mit dem Hive Funktionsspektrum strukturiert werden. 2. Die in Hive Tabellen befindlichen Daten können mit dem Hive Funktionsspektrum für die Analyse bereinigt und transformiert werden.	Mit den Hive Funktionalitäten können einfache Analysen umgesetzt werden.	Mit der Einbindung von UDF-Bibliotheken wie HiveMall ist Data Mining und Machine Learning möglich.
Impala	Sind die Daten in Impala-Tabellen überführt, kann mit den umfangreichen Funktionen und Bibliotheken von Impala ein Datenverständnis generiert werden. Impala bietet hier nahezu den selben Funktionsumfang wie Hive.	1. Sind die un- oder semistrukturierten Daten in Impala Tabelle abgelegt, können diese mit dem Impala Funktionsspektrum strukturiert werden. 2. Die in Impala Tabellen befindlichen Daten können mit dem Impala Funktionsspektrum für die Analyse bereinigt und transformiert werden.	Mit den Impala Funktionalitäten können einfache Analysen umgesetzt werden.	Nach aktueller Recherche keine Data Mining und Machine Learning Lösung für Impala gefunden.

Tabelle 5.1: Einsatzmöglichkeiten analytische Komponenten des Hadoop Ecosystem Teil 1

Tool	Aufgabe			
	Datenverständnis	**Datentransformation**	**einfache Analysen**	**Data Mining & Machine Learning**
HAWQ	Als SQL-Tool auf Hadoop (ebenso wie Hive und Impala) bietet HAWQ ein umfangreiches Funktionsspektrum, um ein Datenverständnis zu generieren. Zudem beinhaltet HAWQ in seiner MADLib weitere spezielle Funktionen (bspw. *SUMMARY()*), die eine Generierung von Datenverständnis besser unterstützen.	1. Sind die un- oder semistrukturierten Daten in HAWQ Tabellen abgelegt, können diese mit dem HAWQ Funktionsspektrum strukturiert werden. 2. Die in HAWQ Tabellen befindlichen Daten können mit dem HAWQ Funktionsspektrum für die Analyse bereinigt und transformiert werden.	Mit den umfassenden SQL Funktionalitäten von HAWQ können einfache Analysen umgesetzt werden.	Bietet mit der MADLib ein breites Spektrum an Data Mining und Machine Learning Funktionalitäten.
Pig	Pig bietet einen umfangreiches Funktionssprektrum, um ein Datenverständnis zu generieren.	1. Mit Pig können die un- und semistrukturierten Daten direkt zugegriffen und in eine strukturierte Form transformiert werden. 2. Mit dem Funktionsspektrum von Pig und bestehenden Pig UDF Bibliotheken können die Daten für die Analyse bereinigt und transformiert werden.	Pig bietet ein Funktionsspektrum mit dem einfache Analysen umgesetzt werden können.	Nach derzeitiger Recherche keine Möglichkeit gefunden.

Tabelle 5.2: Einsatzmöglichkeiten analytische Komponenten des Hadoop Ecosystem Teil 2

Tool	Aufgabe			
	Datenverständnis	**Datentransformation**	**einfache Analysen**	**Data Mining & Machine Learning**
Mahout	Aus der Recherche konnten keine Möglichkeiten identifiziert werden, um die Aufgabe Datenverständnis zu ermöglichen.	Es existiert keine Möglichkeit der Datentransformation.	Aus der Recherche konnten keine Möglichkeiten identifiziert werden, um einfache Analysen zu ermöglichen.	Spezialisiertes Tool für Data Mining und Machine Learning. Mahout bietet ein breites Spektrum an Data Mining und Machine Learning Algorithmen.
Spark	Spark bietet mit SparkQL (HiveQL) Möglichkeiten ein Datenverständnis zu generieren. In der MLib sind weitere Funktionalitäten (bspw. *SUMMARY()*) implementiert, um ein Datenverständnis zu generieren.	1. Spark bietet APIs, um Java, Scala und Python Programme zu erstellen. Mit diesen Programmen können un- und semistrukturierte Daten strukturiert werden. 2. Mit Java, Scala oder Python Programmen, SparkQL oder dem Hive Interpreter oder den Funktionalitäten der MLib können die Daten für die Analysen bereinigt und transformiert werden.	Mit Java, Scala oder Python Programmen, SparkQL (oder dem Hive Interpreter) können einfache Analysen durchgeführt werden.	Spark bietet mit der MLib die Möglichkeit Data Mining und Machine Learning Algorithmen auszuführen.

Tabelle 5.3: Einsatzmöglichkeiten analytische Komponenten des Hadoop Ecosystem Teil 3

Tool	Aufgabe			
	Datenverständnis	**Datentransformation**	**einfache Analysen**	**Data Mining & Machine Learning**
Radoop	Das Tool RapidMiner Radoop bietet Möglichkeiten ein Datenverständnis zu generieren. Mit dem User-Interface können Workflows zum Datenverständnis modelliert und die Ergebnisse visualisiert werden.	1. Un- oder semistrukturierte Daten können nicht in eine strukturierte Form gebracht werden. 2. Radoop bietet eine Vielzahl von Operatoren für die Workflows, mit denen die Daten für die entsprechenden Analysen bereinigt und transformiert werden können.	Das Tool Radoop ermöglicht es einfache Analysen durchzuführen. Primär ist es ausgelegt für Data Mining und Machine Learning.	RapidMiner Radoop ist ein spezialisiertes Data Mining und Machine Learning Tool für Hadoop. Über das User-Interface können viele Data Mining und Machine Learning Algorithmen ausgeführt werden.

Tabelle 5.4: Einsatzmöglichkeiten analytische Komponenten des Hadoop Ecosystem Teil 4

6 Fazit

In dieser Seminararbeit wurden die analytischen Kapazitäten des Apache Hadoop Ecosystem betrachtet. Bei der Recherche wurde festgestellt, dass im Kontext des Apache Hadoop Ecosystem eine Vielzahl von analytischen Lösungen und Tools existieren. Die Seminararbeit wurde auf die Tools Hive, Impala, HAWQ, Pig, Mahout, Spark und RapidMiner Radoop beschränkt. Zusätzlich lassen sich die meisten Tools schwierig miteinander vergleichen, da die Tools oft nur einen Teil gemeinsamer Funktionalitäten besitzen. Viele, der in dieser Seminararbeit betrachteten Tools können parallel und häufig miteinander betrieben werden. Abschließend kann gesagt werden, dass das Apache Hadoop Ecosystem eine Vielzahl von analytischen Möglichkeiten bereitstellt. Die Herausforderung besteht darin, aus den vielen Lösungen und Tools das Optimum für den jeweiligen Anwendungsfall zu ermitteln.

Literatur

[Apa11] APACHE WIKI: *Apache Pig*. https://cwiki.apache.org/confluence/display/PIG/Index. Version: 2011

[Apa13] APACHE: *HDFS Architecture Guide*. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. Version: 2013

[Apa15a] APACHE: *Apache Pig*. http://pig.apache.org/. Version: 2015

[Apa15b] APACHE: *Apache Spark*. http://spark.apache.org/. Version: 2015

[Apa15c] APACHE: *DataFU Pig*. https://datafu.incubator.apache.org/docs/datafu/getting-started.html. Version: 2015

[Apa15d] APACHE: *Language Manual UDF*. https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF. Version: 2015

[Apa15e] APACHE: *Machine Learning Library (MLlib) Guide*. http://spark.apache.org/mllib/. Version: 2015

[Apa15f] APACHE: *Spark Streaming Programming Guide*. http://spark.apache.org/docs/latest/streaming-programming-guide.html. Version: 2015

[Apa15g] APACHE: *Tez*. https://tez.apache.org/. Version: 2015

[Apa15h] APACHE: *Welcome to Apache™ Hadoop! What Is Apache Hadoop?* https://hadoop.apache.org/. Version: 2015

[BD15] BEHRENS ; DR. HENRIK: Spark - der Nachfolger von Hadoop? Neue Big-Data-Plattform. In: *BI-Spektrum* 2015 (2015), Nr. 4, S. 34–37

[BDS15] BALLERSTEDT, Dries ; DITTMER, Dr. C. ; SCHULZ, Peter: Modernisierung klassischer BI-Architekturen durch Big Data: Fluch und Segen zugleich. In: *BI-Spektrum* 2015 (2015), Nr. 1, S. 12–15

[Bit12] BITKOM: *Big Data im Praxiseinsatz – Szenarien, Beispiele, Effekte*. 2012

[Bit14] BITKOM: *Big-Data-Technologien - Wissen für Entscheider: Leitfaden*. 2014

[Clo15] CLOUDERA: *Cloudera Impala Guide*. http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala.html. Version: 2015

[Ehr15] EHRMANTRAUT, Michael: Die Zukunft des klassischen Data Warehouse: Agile Datenversorgung für Self-Service Analytics. In: *BI-Spektrum* 2015 (2015), Nr. 01

[Fis15] FISCHER, Jan-Henrik: Analytics 3.0 verbindet BI und Big Data: Sechs Thesen zu BI, Big Data Analytics und der digitalen Transformation. In: *BI-Spektrum* 2015 (2015), Nr. 4, S. 30–33

[FUJ13] FUJITSU: *White Paper: Lösungsansätze für Big Data*. 2013

[Haw15] HAWQ: *HAWQ*. http://hawq.incubator.apache.org/. Version: 2015

[Hor15a] HORTONWORKS: *Apache Mahout: An algorithm library for scalable machine learning on Hadoop*. http://de.hortonworks.com/hadoop/mahout/#section_1. Version: 2015

[Hor15b] HORTONWORKS: *YARN: The Architectural Center of Enterprise Hadoop*. http://de.hortonworks.com/hadoop/yarn/. Version: 2015

[HPL15] HPLSQL: *Why HPL/SQL*. http://www.hplsql.org/why. Version: 2015

[Hum14] HUMMELTENBERG, Prof. Dr. W.: *Business Intelligence*. http://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/daten-wissen/Business-Intelligence. Version: 2014

[Kum14] KUMAR, Anoop: *How Hadoop is Different from Conventional BI*. http://www.developer.com/db/how-hadoop-is-different-from-conventional-bi.html. Version: 2014

[MAD15] MADLIB: *MADlib Documentation*. http://doc.madlib.net/latest/index.html. Version: 2015

[Mah15a] MAHOUT: *How Mahout works?* http://de.hortonworks.com/hadoop/mahout/#section_2. Version: 2015

[Mah15b] MAHOUT: *Mahout 0.10.1 Features by Engine*. http://mahout.apache.org/users/basics/algorithms.html. Version: 2015

[Mak15] MAKOTO YUI: *Hivemall*. https://github.com/myui/hivemall/wiki. Version: 2015

[MSH14] MOCK, Michael ; SYLLA, Karl-Heinz ; HECKER, Dirk: Skalierbarkeit und Architektur von Big-Data-Anwendungen. In: *OBJEKTspektrum IT-Trends* 2014 (2014), Nr. 2014, S. 1–7

[Piv15] PIVOTAL: *Apache MADlib (incubating): Machine Learning at Scale*. https://pivotal.io/de/madlib. Version: 2015

[PZ15] PLEBAN, Uwe ; ZIMMER, Michael: SQL auf Hadoop - rasante Entwicklungen: Benchmarks zur Messung der Performance von SQL-Implementierungen. In: *BI-Spektrum* 2015 (2015), Nr. 1, S. 22–24

[Rap14] RAPIDMINER: *Best Practices for Applying Advanced Analytics in Hadoop*. https://www.youtube.com/watch?list=PLssWC2d9JhOYlsad84A4kEo_HYdOi2G5p&v=e6VmWAaIqZc. Version: 2014

[Rap15a] RAPIDMINER: *RapidMiner Radoop: Apache Spark Integration*. https://www.youtube.com/watch?list=PLssWC2d9JhOYlsad84A4kEo_HYdOi2G5p&v=wgVFLmZvYZI. Version: 2015

[Rap15b] RAPIDMINER: *RapidMiner Radoop Documentation: Installing RapidMiner Radoop on RapidMiner Studio*. http://docs.rapidminer.com/radoop/installation/radoop-studio-install.html. Version: 2015

[Rap15c] RAPIDMINER: *RapidMiner Radoop: Predictive analytics at unlimited scale extracts full value from Hadoop investments*. https://rapidminer.com/products/radoop/. Version: 2015

[Sak14] SAKR, Sherif: *A brief comparative perspective on SQL access for Hadoop: Back to top Comparison with other big SQL systems*. http://www.ibm.com/developerworks/library/ba-compare-sql-access-hadoop/. Version: 2014

[Sta15] STATISTA ; STATISTA (Hrsg.): *Prognose zum Volumen der jährlich generierten digitalen Datenmenge weltweit in den Jahren 2005 bis 2020 (in Exabyte)*. http://de.statista.com/statistik/daten/studie/267974/umfrage/prognose-zum-weltweit-generierten-datenvolumen/. Version: 2015

Abschließende Erklärung

Ich versichere hiermit, dass ich meine Seminararbeit (Titel der Arbeit)... selbständig und ohne fremde Hilfe angefertigt habe, und dass ich alle von anderen Autoren wörtlich übernommenen Stellen wie auch die sich an die Gedankengänge anderer Autoren eng anlegenden Ausführungen meiner Arbeit besonders gekennzeichnet und die Quellen zitiert habe.

Oldenburg, den 26. September 2016 Felix Kruse

Integration and analytical capabilities of Apache Spark

Seminar Work

Director:	Prof. Dr.-Ing. habil. Jorge Marx Gómez
Supervisors:	M. Eng. & Tech. Viktor Dmitriyev Dr.-Ing. Andreas Solsbach M.Sc. Ola Mustafa
Submitted by:	Ahmad Albuhaisi ahmad.albuhaisi@uni-oldenburg.de
Submission date:	2. Dezember 2015

Contents

Abbreviations

MR	MapReduce
SQL	Structured Query Language
PG	Project Group
RDD	Resilient Distributed Dataset
ML	Machine Learning
YARN	Yet Another Resource Negotiator
HDFS	Hadoop Distributed File System
I/O	Input/Output
API	Application Programing Interface
DB	Data Base
DBMS	Data Base Management System
JDBC	Java Data Base Connectivity
ODBC	Open Data Base Connectivity
SVM	Support Vector Machine

Table Of Figures

Table Of Tables

1 Introduction

As daily generated data are dramatically increasing in the current so called era of Big Data, many software and frameworks have been developed to line with the requirements this new era brings [Pat15]. Apache spark is considered as one of the most interesting projects that extends and generalize MapReduce (MR). The Strong reason behind using Spark is the fact that Apache Spark is easier, faster, and more general. Spark offers the in memory data analysis solution which makes it up to 100x faster than MR [SS15]. Computations like Machine Learning ML, Text Processing, Data Streaming, and SQL queries, required previously different methods and tools, Apache Spark makes it possible to combine all of these required engines in one framework.

1.1 Motivation

We are a project group (PG) consists of 12 master students who work for a year on developing a full system to solve a given problem in the field of computer science. Our goal in this PG is to realize a system using Hadoop ecosystem and it's components to be able to analyze and evaluate the data generated from CEWE photo stations which are distributed in all over Germany and also many other countries in Europe. CEWE Company as the project partner also aims to be able to perform near time analyzes, create an alarm system to detect failures, and improve the customer software. This work should presents the integration and analytical capacities of apache spark as a part of the Apache Hadoop ecosystem. The goal is to achieve a better understanding of the concept, the structure, and the use of apache spark to deliver it as a tool to tackle data analysis problems that are related to out project.

1.2 Gaols and Structure

As mentioned, this research aims to discuss and explain Apache Spark and its analytic capabilities as a part of the Apache Hadoop ecosystem. The main goals of this work is to define Apache Spark and discuss its architecture and components

and give some examples to programing with Spark. This work gives a brief answer on these questions:

- What is apache Spark and what is it really all about?
- Why Apache Spark?
- What is Spark Programming model?
- What are the key concepts of Spark?
- What are the components of Spark?
- What are the key features of Apache Spark components?

Based on this purpose this work is divided into 5 Chapters as followed:

CHAPTER 1 introduces the motivations and objectives and shows the structure of this work and preview some other related works to this subject.

CHAPTER 2 defines Spark and gives a literature overview about its architecture. It presents Spark key features and advantages and goes throw a first steps in Spark Programing.

CHAPTER 3 introduces Spark Programing model. It discusses RDDs concept which is the most important concept in Spark and shows the most common operations - Actions and Transformations - that could be applied on RDDs. The concepts lazy evaluation and Fault tolerance are also discussed in this section. The second section talks about the shared variables in Spark.

CHAPTER 4 illustrates the Spark Platform and components in more details. A brief overview about Spark SQL, Spark MLib, Spark Streaming, and Spark GraphX and their basics and architecture is given in chapter 4.

CHAPTER 5 aims to briefly conclude this research and sum up the ideas mentioned in the previous chapters.

1.3 Related Work

Many other papers and books are written about Spark. In the last years, Spark became one of the most leading topics in the big data and cluster computing fields. The books "Learning Spark" [KKWZ15] and "Advanced Analytics with Spark" [RLOW15] from O'REILLY publisher are - beside two other books from PACKT publisher [Pen15] [SK15] and many other papers, training, and tutorials - the most referenced articles in this work. Writing this work demanded a deeper look on literature overview and to gain a better understanding of Hadoop, Spark and related subjects. Many other summits and courses can be found on the internet to

dig deeply into spark and practice it. edX website offer the course "Introduction to Big Data with Apache Spark" free online, its a very good step to start with before working with spark.

2 Apache Spark Overview

This chapter defines Apache Spark and shows differences and advantages of Spark and MapReduce. It introduces the Spark architecture and illustrates a "wordcount" example using spark.

2.1 What is Apache Spark?

Apache spark [KKWZ15, P. 01-07] is an open source data analytics cluster computing framework originally developed in AMPLab at UC Berkeley. Spark offers APIs in Scala, Python, and Java and available for Mac, Linux and Windows operating systems. It is integrated into Hadoop and can access any Hadoop data source. Spark extends the MapReduce (MR) model, it provides primitives for In-Memory cluster computing which makes it able to load data into a cluster memory so that they could be processed much faster and queried repeatedly. Many large companies including Yahoo and Intel are using spark and had contributed code to it. Spark covers many workloads that required previously different distributed systems such as SQL, Data Streaming, Machine Learning and Graphics Processing. The following chapters will provide a higher level overview of the apache spark concepts and components.

2.2 Spark vs MapReduce

MapReduce [Onl] is a cluster computing and programing model framework usually used to process big data. It consists mainly of two parts: Map and Reduce. This section aims to show the differences between MapReduce and Spark as shown in the following table [GA15].

2.3 Who uses Apache Spark and for what?

Based on the Book “Learning Spark” [KKWZ15, P. 04-07], the main users of Apache Spark can be classified into two groups: Data Scientist and Engineers. Each of these two groups uses Spark for certain purposes and cases which leads to define two major use cases:

	MapReduce	**Spark**
Storage	Disk only	In-memory or on disk
Operations	Map and Reduce	Map, Reduce, Join, Sample, etc...
Execution model	Batch	Batch, interactive, streaming
Programming Languages	Java	Scala, Java, R, and Python

Table 2.1: MapReduce vs SPARK

1. Data Science Tasks: Data science is a term commonly known in the past few years and related at most to data analysis. A data scientist usually has experiences in SQL, Programing (most common in Python), Statistics, ML, and modeling, and his main task goes always around analyzing and modeling data. Data scientist work starts always from a question that should be answered by using interactive shells and queries to analyze data and get the results. The simplicity, built-in libraries, and the fast of Spark makes it so appropriate to perform these tasks and to be adapted as a solution in the data science field.

2. Data Processing Applications: For engineers, Spark is a simple easy way to distribute complex programs across clusters and at the same time hide the complexity of networking and programing. Engineers are usually a large class of software developers mostly interested in software engineering, Interfaces, and object oriented programming. Engineers tends to use Spark because of its reliability, wide functionality and ease to learn and use.

2.4 Spark Architecture

Basically, Spark is a framework to process big data which could be in petabytes. The most important two key factors of such a framework are the computation and scaling abilities. Spark [SK15, P. 07-08] can apply complex computations in a short time over big data. Data distribution and In-Memory computations enable spark to parallelize algorithms instead of doing them one step after another.

Figure one shows the architecture of spark. Spark users start by writing a Spark driver program which is responsible for further spark jobs like distributing and interacting with the data which is represented in Resilient Distributed Datasets (RDDs). It's also responsible for distributing the code over the Worker nodes by passing closures "Functions" to them. Transformations, actions, and fault tolerance

[1]Source: http://spark.apache.org/docs/latest/cluster-overview.html

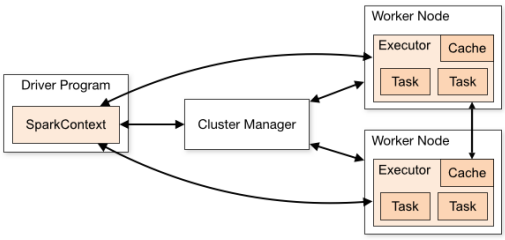


Figure 2.1: Spark Architecture[1]

are all managed also by the spark driver. So, in few word, Spark driver is the core, the brain and the manager of any Spark program.

Spark works with the Hadoop YARN cluster manager which makes it fully compatible with Hadoop. After preparing the RDDs and transformations, Spark driver manages to deliver these program classes to the cluster manager (YARN). Afterwards, YARN is responsible to handle the distribution of this work load over the working nodes. Actions only take part later when we run the program by the term known as lazy evaluation (discussed later). Every worker node has its own RDD and functions "Closure" but can also share and broadcast variables manually (Chapter 2). In spark programming [Cho15], slaves can hold variables in the cache and reuse them in iterations without the need to send or receive data from the driver and the file system which is a big difference between Spark and MapReduce. In MapReduce, the only way for iterations is to save data after each step into the file system and load it again. Spark programing model will be discussed in more detail in the next chapter.

2.5 The first step to a Spark Program

This section aims to illustrate the ease-of-use of spark to apply complex algorithms by showing a short wordcount example using Spark. The following box shows the Spark wordcount example [Bon15]. Word count program is considered as the "Hallo World" in the big data field.

Scala 2.1: Word count program in Scala

```
1  val conf = new SparkConf().setAppName("Spark␣wordcount")
2  val sc = new SparkContext(conf)
3  val file = sc.textFile("hdfs://...")
4  val counts = file.flatMap(line => line.split("␣"))
5  .map(word => (word, 1)).countByKey()
6  counts.saveAsTextFile("hdfs://...")
```

The program "wordcount" searches for all keywords in the source file and counts the occurrences for each word. The example initiate a spark context at first then loads the data from an external file then simply uses RDD transformations ("flatmap" and "map") to build pairs RDD dataset "counts" (String, Int). The function "reduceByKey" then sums all the occurrences of a word that has the same key. The last step will be then saving the results into a file. Next chapter will go through RDD actions and transformations in more details.

3 Spark Components

Apache Spark framework contains multiple integrated components. In its core, Spark is a fast computational engine that takes responsibility for scheduling, distributing and managing programs. Spark core powers also other higher-level components that are responsible for other specialized workload like machine learning and Spark SQL.

Based on the Learning Spark book [KKWZ15], Spark has a very tight integration which comes with many benefits. Because of the spark stack layer structure, any improvement in the lower layer will also be available to the higher level component. For example, Spark SQL and ML will benefit from any optimization or improvement in the core layer of spark. Spark can also save costs, a company needs only to use one software instead of using many other independent software, it can also save maintenance, testing, support, and deployment costs. Last but not least, Spark enables engineers and data scientists to build applications that combine and work on multiple programing models. For example, using Spark makes it available to write an application which do some ML on real-time data that come from data streams and at the same time enable data analysts to query the result, all in one system.

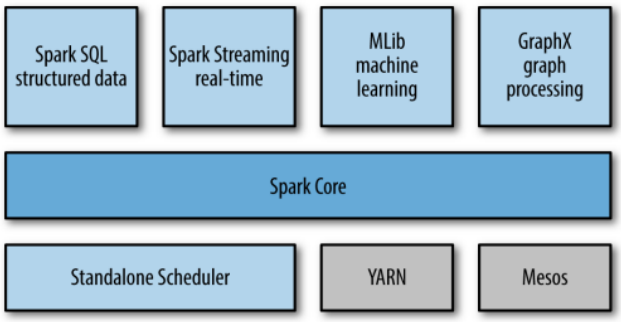


Figure 3.1: Spark Components [KKWZ15]

This chapter briefly introduces the components of Apache Spark shown in Figure 4.1. As discussed in the last chapters, Spark is compatible with both Hadoop over YARN and Mesos. It can also work as a standalone framework. Beside spark core, Spark has other four components. These components are designed to collab-

orate together means that they are all connected together and can also be combined as libraries in applications.

3.1 Spark Core

Spark core [Ban15] is the first and main component of Spark. It contains the basic functionality of spark and is responsible for tasks scheduling, data distribution, fault tolerance, and all I/O interacting. Spark core builds also the RDDs and provides many APIs to manipulate them.

3.2 Spark SQL

Spark SQL component is the latest developed Spark Component which allows Spark to work with structured data and enables querying data using SQL. Shark [AXL+15] was actually the first relational interface on Spark. The aim was to modify Apache Hive system to work over Spark. Despite the good performance of Shark, it had other challenges like working on data inside spark instead of data that are stored in Hive catalog. Another challenge is that Hive was basically developed for MapReduce, which makes it hard to build new features that supports Spark other engines like ML and Stream.

Spark SQL was mainly developed to:

- Support relational processing in both data from inside and outside Spark.
- Use the already existing DBMS methods to provide a high performance.
- Support different data sources like semi-structured data and external data sources.
- Extend applications with other advanced analytics like ML and GraphX.

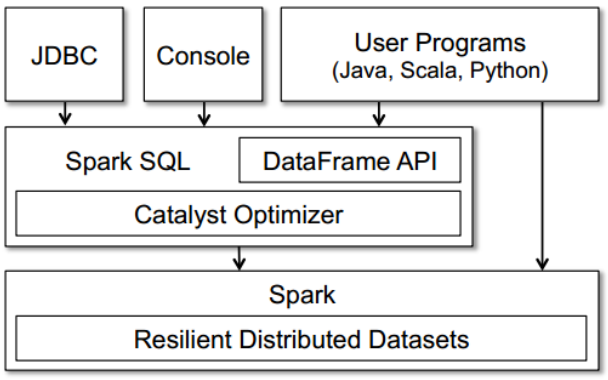


Figure 3.2: Spark SQL Architecture [AXL+15]

Spark SQL works as a library on top of Spark. As shown in figure 4.2; JDBC/ODBC, Console, and also user programs (written in supported languages like Java, Python, and Scala) can all access Spark SQL. Inside Spark SQL, it contains DataFrame API and Catalyst Optimizer. DataFrame API is the equivalent to table in a relational DB, and it's the main abstraction in Spark SQL. DataFrames can be manipulated, and are all lazy executed, nothing occurs until the first output operation like "save" is being executed. Catalyst optimizer is a framework for query optimizations. It can be extended by the use of custom rules. External systems like BI tools can also use catalyst to query data using protocols like JDBC and ODBC.

3.3 Machine Learning with Spark MLib

MLib is a Spark library which contains many functions to support machine learning. It has many learning algorithms and can be accessed from all Spark supported programing languages. Actually, ML is a big subject that has a whole books to discuss it, therefore, this sections gives just a small overview about ML in Spark framework.

Briefly, we can define Spark MLib as a group of functions that can be called on RDDs to do some computations and deliver results [KKWZ15, P. 215 - 218]. Based on training data, algorithms basically try to make predictions or take decisions. ML problems normally include classification, regression, and clustering. Each of this type of problems has its own method, algorithms and objectives. Spark Mlib supports algorithms like logistic regression, naive Bayes classification, SVM, decision trees, random forests, linear regression, k-means clustering and others [Bon15].

3.4 Spark Streaming

Spark streaming is a framework to support processing of real-time data. It is fault-tolerant and supports many data sources like HDFS, Kafka, Flume, Twitter, ZeroMQ and other custom sources. Spark Streaming brings some good features to the table than other Hadoop components, one the of the main features is that Spark streaming and be combined with other Spark components like MLib and GraphX and support real-time processing for these framework [Bon15].

3.5 Spark GarphX

GraphX is a Spark library to manipulate graphs and parallelize graph computations [KKWZ15, P. 04]. It extends the RDD API and enables to create graphs with properties attached to vertexes and edges. GraphX supports many common graph algorithms and provides various operations for graph manipulations.

4 Spark Programing Model

Spark provide two core abstractions for working with data, Resilient distributed datasets known as RDDs, and shared variables. This chapter introduces the main two cores and covers how to work with them.

4.1 RDDs Programming

An RDD [Pen15, P. 14-157] is simply a collection of elements, which is distributable, immutable, parallelizable, and fault-tolerant. RDDs can't be changed after creation, but they can be transformed and actions can be performed on them.

There are two ways to create an RDD [KKWZ15, P. 25-26] [RLOW15, P. 16-17], the first option is to parallelize a collection of data in the driver program, and the second option is to load data from a file as shown on these two boxes [1].

Scala 4.1: Paralellizing a collection of data

```
1 Data = [1, 2, 3, 4, 5]
2 distData = sc.parallelize(data,3)
```

"Data" implies the collection of objects and the second argument "3" is the number of partition. To create RDDs from an external file, we simply pass the file name or directory to the "textfile" function.

Scala 4.2: Loading data from a file

```
1 distFile = sc.textFile("data.txt")
```

After creation, it's still no actual data read or loaded yet into clusters, all the computations are lazy evaluated and take part only when we run the program.

Spark supports also implicit type conversion for RDD of tuple[Key,Value] which known as Pair RDD. Pair RDDs provides the ability to perform methods and functions over the RDD Key. For example reduceByKey(), to aggregate all RDDs that

[1] Source: "http://spark.apache.org/examples.html"

have key of the same type, or join() method which enables merging two RDDs together.

Scala 4.3: Merging two RDDs together

```
1 val lines = sc.textFile("data.txt")
2 val pairs = lines.map(s => (s, 1))
3 val counts = pairs.reduceByKey((a, b) => a + b)
```

This code [2] uses the reduceByKey operation on key-value pairs to count how many times each line of text occurs in a file.

4.1.1 RDD Operations

Creating an RDD means that we have already distributed a collection of elements to be manipulated [Pen15, P. 14-16]. RDDs manipulation are done in Spark by using Operations. Spark operations consist of Transformations and Actions. This section defines Spark Transformations and Actions and give examples to the most common used operations.

Transformations

Transforamtions [KKWZ15, P. 26-27] are RDDs operations that gives back a new RDD after applying it. Transforamtions are always computed lazily (will be discussed later), and only executed when we use them in an action. The following table shows the most common RDDs Transformations and their description.

Note that there are two confusing mapping functions in Spark, map() and flatmap(). The difference between map and flatmap is that, map function is a one-to-one function which means that it takes an element as input and gives one element as output. On the other hand, flatmap returns one or more than one element as output for any input element it takes [SK15, P. 66].

Actions

Actions apply computations or aggregations on RDDs and return the final results to the program to be stored [Pen15, P. 14-16]. Actions are used to change something in the data sets, they force the execution of the transformations and return the processed results. The following table shows the most common RDDs Actions and their description.

[2] Source: "http://spark.apache.org/examples.html"

Transformation	Description
map(func)	Return a new distributed dataset formed by passing each element of the source through a function func.
filter(func)	Return a new dataset formed by selecting those elements of the source on which func returns true.
flatMap(func)	Similar to map, but each input item can be mapped to 0 or more output items (so func should return a group of items rather than a single item).
mapPartitions(func)	Similar to map, but runs separately on each partition (block) of the RDD. Function receives the whole partition as an argument as opposed to map, where the function is called for each RDD element separately.
union(otherRDD)	Return a new dataset that contains the union of the elements in the source dataset and the argument.

Table 4.1: Common RDD Transformations. Source: [Bra14]

Transformation	Description
reduce(func)	Aggregate the elements of the dataset using a function func (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
collect()	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
count()	Return the number of elements in the dataset.
first()	Return the first element of the dataset (similar to take(1)).
take(n)	Return an array with the first n elements of the dataset.
saveAsTextFile(path)	Write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call toString on each element to convert it to a line of text in the file.

Table 4.2: Common RDD Actions [Bra14]

In short words, and comparing Spark to MapReduce, we can say that Transformations in Spark are the Mapping part in MapReduce, and Actions are the Reduce part.

4.1.2 Lazy Evaluation

Lazy evaluation is a very important concept in Spark. It means that there is no any program execution without Actions. Transformation and data loading are just instructions to the program and will not be performed immediately unless we run the program. Spark uses the lazy evaluation concept to reduce as much as possible the passes it has to take over data by grouping all data operations and execute them once instead of doing every step a part [KKWZ15, P. 30].

4.1.3 RDDs caching

Caching is the ability to keep data in memory and considered one of the most important features in Spark. Spark supports caching through the use of the cache function on an RDD as followed.

Scala 4.4: Caching RDDs

```
1 rddFromTextFile.cache
```

After calling the cache function, the first results after executing an Action on an RDD will be brought and stored in the memory. It won't be necessary to write or read data from and into Disk on every iteration process. Data can be directly read from memory and that avoids many I/O disk operations and speeds up data processing [Pen15, P. 18].

4.1.4 Fault Tolerance

A fault-tolerant program is a program that has the ability to keep functioning and delivering results correctly even in the case where some failures happen. Spark supports fault-tolerance by recording the RDD lineage. Spark always know the origin process that were applied on the RDD, that way it can regenerate the RDDs whenever failures occur [Bra14]. RDDs on the other side, have to be intermediately stored in the file system, this way, Spark has always a last good known status for the data and can start generating it stating from this saved good version.

4.2 Shared Variables

Spark distributed the work be sending copies of variables to the clusters, so every node works on the same set of variables but separately, which can lead to unnec-

essary duplications and overhead [Riv15]. Spark avoids that by using the shared variables two concepts: (1) Broadcast Variables and (2) Accumulators.

4.2.1 Broadcast Variables

Broadcast variables means the ability to send some pieces of data just once to the nodes instead of sending it in every closure [Riv15]. These data are usually read only data and used several times in multiple parallel operations like hash tables. Broadcast variables are so easy to be created in Spark and could be done as follows.

Scala 4.5: Broadcasting Variables

```
1 val broadcastList = sc.broadcast(List("a","b","c","d","e","f"))
```

4.2.2 Accumulators

Accumulators in Spark are the same as counters in MapReduce. They are variables that can be read only by the Spark driver. Workers can only access and add to their own local accumulator, the global value can be accessed only by the driver. This way, its always assured that the global accumulator is correctly computed [Pen15, P. 20].

Scala 4.6: Accumulators

```
1 >>> accum = sc.accumulator(0)
2 Accumulator<id=0, value=0>
3 >>> sc.parallelize([1, 2, 3, 4]).foreach(lambda x: accum.add(x))
4 ...
5 10/09/29 18:41:08 INFO SparkContext: Tasks finished in 0.317106 s
6 scala> accum.value
7 10
```

The example [3] uses the built in support of int accumulator. Other types can be programmed by sub classing the AccumulatorParam interface.

[3] Source: "http://spark.apache.org/examples.html"

5 Conclusion

This research is done in a project group consists of 12 students to fulfill the requirements of getting a master degree in informatics in Oldenburg University. It aims to discuss and explain Apache Spark and its analytic capabilities as a part of the Apache Hadoop ecosystem and to answer questions relating to Apache Spark framework.

The world is generating huge amounts of data every second, these data needs to be processed and stored in a fast way to enable engineers and scientist to gain knowledge from these data and make the job easy to take decisions. Spark is a complete framework which contains four libraries to work with big data. Apache Spark is an open source data analytics cluster computing framework that supports Java, Python, and Scala programming languages. It's faster than MapReduce because it supports the In-Memory data analysis.

Spark main data abstractions are RDDs and Shared Variables. RDDs are a set of data that can be distributed and parallelized. They are resilient means that they can't be changed once created, but they can be transformed and actions can be performed on them. There are many RDD transformations and actions that helps in manipulating the RDDs. RDD programing supports also many helpful technologies like lazy evaluation and caching and being fault-tolerance. Shared variables is the second data abstraction and consists of the Broadcasting variables and Accumulators. Broadcast variables are read only variables that should be sent just one time to the clusters, and accumulators are variables that can be read only by the program and can be added by all workers.

Spark five component are: Spark Core, Spark SQL, Spark Streaming, Spark Mlib, and Spark GraphX. Spark combine all these features in framework which brings too many benefits for companies that work on many heterogeneous frameworks. Using spark might also save much testing, maintaining, support and deployment costs. Sparks allow programmers to create applications that use many independent components easily.

Literature

[AXL+15] ARMBRUST, Michael ; XIN, Reynold S. ; LIAN, Cheng ; HUAI, Yin ; LIU, Davies ; BRADLEY, Joseph K. ; MENG, Xiangrui ; KAFTAN, Tomer ; FRANKLIN, Michael J. ; GHODSI, Ali ; ZAHARIA, Matei: Spark SQL: Relational Data Processing in Spark. In: *Databricks Inc. and MIT CSAIL and AMPLab, UC Berkeley* (2015), June. – http://dx.doi.org/10.1145/2723372.2742797

[Ban15] BANSOD, Amol: Efficient Big Data Analysis with Apache Spark in HDFS. In: *International Journal of Engineering and Advanced Technology (IJEAT)* 4 (2015), August

[Bon15] BONAĆI, Marko: *Spark in Action*. Bd. 6. Manning Publications, 2015

[Bra14] BRATUS, Andrii: *RefineOnSpark: a simple and scalable ETL based on Apache Spark and OpenRefine*, UNIVERSITY OF TRENTO, DEPARTMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE, Diplomarbeit, 2014

[Cho15] CHOWDHURY, Mosharaf: Performance and Scalability of Broadcast in Spark. In: *University of California, Berkeley* (2015)

[GA15] GOPALANI, Satish ; ARORA, Rohan: Comparing Apache Spark and Map Reduce with Performance Analysis using K-Means. In: *International Journal of Computer Applications* 113 (2015), Nr. 1

[KKWZ15] KARAU, Holden ; KONWINSKI, Andy ; WENDELL, Patrick ; ZAHARIA, Matai: *Learning Spark*. First Edition. USA : O'REILLY, 2015

[Onl] ONLINE: *Mapreduce Tutorial*, `http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html`

[Pat15] PATIL, Malini M.: Challenges and Issues in Handling Big Data. In: *International Journal of Innovations and Advancement in Computer Science* 4 (2015), Nr. ISSN 2347 – 8616

[Pen15] PENTREATH, Nick: *Machine Learning with Spark*. First Edition. UK : Packt Publishing Ltd., 2015

[Riv15] RIVERA, Juan De Dios S.: Data Analysis on Hadoop - finding tools and applications for Big Data challenges. In: *UPPSALA UNIVERSITET, Department of Information Technology* (2015)

[RLOW15] RYZA, Sandy ; LASERSON, Uri ; OWEN, Sean ; WILLS, Josh: *Advanced Analytics with Spark*. First Edition. USA : O'REILLY, 2015

[SK15] SANKAR, Krishna ; KARAU, Holden: *Fast Data Processing with Spark*. Second Edition. UK : Packt Publishing Ltd., 2015

[SS15] SOOMRO, Tariq R. ; SHORO, Abdul G.: Big Data Analysis: Apache Spark Perspective. In: *Global Journal of Computer Science and Technology* 15 (January 2015), Nr. Online ISSN: 0975-4172

Abschließende Erklärung

I solemnly declare that I have drawn up this seminar paper without help from a third party and without having used other means than the indicated resources and auxiliaries and that I have made all passages recognizable as quotations that I have taken in letter or in spirit from the used sources. This paper has not been submitted in the same or similar form in another seminar.

Oldenburg, den September 26, 2016 Ahmad Albuhaisi

Visualisierungsmöglichkeiten mittels dem Apache Hadoop Ecosystem

Seminararbeit

Themenersteller:	Prof. Dr.-Ing. habil. Jorge Marx Gómez
Betreuer:	M. Eng. & Tech. Viktor Dmitriyev Dr.-Ing. Andreas Solsbach
Vorgelegt von:	Jonas Friedrich jonas.friedrich@uni-oldenburg.de
Abgabetermin:	2. Dezember 2015

Inhaltsverzeichnis

Abkürzungen

HDFS Hadoop Distributed File System

HDP Hortonworks Data Platform

SQL Structured Query Language

TTT Task by Data Type Taxonomy for Information Visualization

VM Virtuelle Maschine

YARN Yet Another Resource Negotiater

Abbildungen

Tabellen

1 Einleitung

In diesem Kapitel wird die Motivation zur Erstellung dieser Arbeit beschrieben. Des Weiteren wird der Inhalt und die Struktur der Arbeit vorgestellt. Diese Seminararbeit wurde im Rahmen der Projektgruppe „DASH“ (Data Analytics with Hadoop) an der Carl von Ossietzky Universität in Oldenburg erstellt. Das Projekt wird in Kooperation mit dem Unternehmen CEWE Stiftung & Co. KGaA im Zeitraum von Oktober 2015 bis Oktober 2016 durchführt.

1.1 Motivation

Mit dem stetig wachsenden Einsatz von Informationstechnologie steigen ebenfalls die gesammelten Datenmengen. Prozesse werden durch IT-Systeme vereinfacht oder automatisiert und protokolliert. Diese gespeicherten Daten bergen eine Menge Potenzial, denn sie können z.B. die Entscheidungsfindung verschiedenster Fragen vereinfachen, bis hin zur Änderung der strategischen Ausrichtung einer Organisation. Um Entscheidungsträgern die Informationen in menschenverständlicher und aggregierter Form darzulegen, sollte eine Visualisierung der Daten vorgenommen werden. (Vgl. [CHU07], S. 4) Auf diese Weise ist es möglich, Änderungen am Markt oder innerhalb einer Organisation festzustellen und zu reagieren. So ist es beispielsweise einfacher, einen Trend in einem zeitlich gegliederten Balkendiagramm zu erkennen, als diesen nach einem Vergleich mehrerer Zahlen einer Tabelle zu schlussfolgern.

Mit der Einführung von Big Data steigt auch der Bedarf an neuartigen Techniken zur Datenvisualisierung. Die Datensets sind für Business- oder Datenanalysten häufig zu umfangreich, um sie mit Hilfe traditioneller Reporting- und Data Mining- Tools anzeigen und interpretieren zu können. In einer von IBM durchgeführten Studie geben beispielsweise „71 Prozent der Studienteilnehmer an, dass ihre Big Data- Projekte auf Fähigkeiten zur Datenvisualisierung angewiesen seien“. ([IBM15], S. 12)

Häufig ist der Begriff Big Data heutzutage doch fast synonym mit Apache Hadoop verknüpft (Vgl. [Neu15]). Der Grundsatz ist dabei allerdings wie immer „Wähle das richtige Tool für deine Aufgabe“.

Im Rahmen dieser Seminararbeit soll der Fokus auf die Visualisierungsmöglichkeiten mittels dem Hadoop Ecosystems gelegt werden. Dabei werden verschiedene Visualisierungskomponenten bzw. Visualisierunglösungen vorgestellt und deren Anwendung bzw. Funktionsweise beschrieben.

1.2 Inhalt und Struktur dieser Arbeit

Wie bereits durch die Motivation verdeutlicht wurde, spielen die Fähigkeiten zur Datenvisualisierung in Big Data- Projekten zunehmend eine große Rolle. Um jedoch einen ersten Überblick über das Thema „Visualisierung“ zu geben, werden in Kapitel 2 hierzu die Grundlagen der Visualisierung näher erläutert. In diesem Zusammenhang wird der Begriff Datenvisualisierung eingeführt sowie die Herausforderungen, Ziele und Aufgaben der Visualisierung beschrieben. Zudem werden in diesem Kapitel verschiedene Datentypen bzw. Diagrammtypen vorgestellt.

Im dritten Kapitel werden verschiedene Visualisierungskomponenten bzw. Visualisierunglösungen des Apache Hadoop Ecosystems vorgestellt und deren Anwendung bzw. Funktionsweise beschrieben.

Abschließend werden in Kapitel 4 die Inhalte der Arbeit zusammengefasst sowie ein Ausblick auf mögliche Einsatzszenarien im Rahmen der Projektgruppe gegeben.

2 Grundlagen der Visualisierung

In diesem Kapitel werden die Grundlagen der Visualisierung beschrieben. Dabei wird der Begriff Datenvisualisierung definiert, sowie der Nutzen und die Herausforderungen der Visualisierung beschrieben. Des Weiteren werden verschiedene Diagrammtypen vorgestellt.

2.1 Datenvisualisierung

Die Visualisierung von Daten unterstützt Zusammenhänge zu erkennen und Daten besser zu verstehen. Der Mensch denkt vielmehr mit Visualisierungswerkzeugen wie Stift, Papier, Flipchart und Whiteboards. Ebenso ist das menschliche Gehirn darauf ausgelegt, Bilder zu speichern und keine Datensätze. Die Loci-Methode, die u.a. häufig von Gedächtnissportlern angewendet wird, arbeitet mit der imaginären Platzierung von Gegenständen in einer bekannten Umgebung und verknüpft dieses mit einer Geschichte. Dieses Vorgehen findet in letzter Zeit häufig in Infografiken Anwendung, die sich aus mehreren Diagrammen zusammensetzen. (Vgl. [Fre14], S. 322)

Der Begriff Datenvisualisierung, oder oft auch synonym Informationsvisualisierung genannt, beschäftigt sich mit computer-unterstützten Methoden zur grafischen Repräsentation großer Mengen von Daten. Die bildlichen Darstellungsmethoden sollen helfen, diese Daten auszuwerten und aus ihnen neue Erkenntnisse zu gewinnen. (Vgl. [Die07], S. 1)

2.2 Herausforderungen

Die erste Herausforderung der Visualisierung ist zunächst diese, dass die eigentliche Fragestellung klar eingegrenzt werden muss. Es gibt z.B. viele verschiedene Diagrammtypen, die sehr unterschiedliche Fragestellungen beantworten und diese sind je nach Sachverhalt mehr oder weniger gut geeignet. So lassen sich beispielsweise einfache Verkaufszahlen klassisch als Balkendiagramm darstellen und pro Monat oder Segment weiter unterteilen. Andere Diagrammtypen wie z.B. eine Heatmap eignen sich etwa für die Darstellung von prozentualen Veränderungen

bei verschiedenen Modellen. Ein geografisches Element, wie eine Karte, veranschaulicht die Verteilung der Zahlen pro Ballungsraum. (Vgl. [Kob15])

Im Umfeld von Apache Hadoop und den „Bordmitteln“ ist die Anzeige der Daten meist auf Kommandozeilentools beschränkt. An dieser Stelle erweitern Big-Data-Suiten den Funktionsumfang und bringen eigene Tools zur Visualisierung und Aufbereitung der im Hadoop gespeicherten Daten mit. Diese Tools bzw. Lösungen werden in Kapitel 3 näher vorgestellt.

Es lassen sich folgende Herausforderungen für die Visualisierung von Daten im Big-Data-Kontext festhalten (Vgl. [Deu15]):

- Immer mehr heterogene Daten (insbesondere abstrakte Daten)
- Immer mehr heterogene Anwendergruppen
- Breiteres Spektrum von Anwendungsbereichen
- Nachfrage nach speziellen visuellen Darstellungen
- Nachfrage nach kundengerechten und einfach zu bedienenden Werkzeugen

2.3 Ziele

Das Ziel der Datenvisualisierung ist die graphische Repräsentation von komplexen oder abstrakten Daten, so dass Zusammenhänge und relevante Eigenschaften intuitiv erfasst werden können. Im Gegensatz zu Listen oder Tabellen bieten grafische Darstellungen eine einheitliche Sicht auf verschiedene Informationsquellen und eröffnen somit, auch bei großen Daten- und Informationsmengen (Big-Data) Analyse- und Bewertungsmöglichkeiten. Ein Vorteil der grafischen Darstellung liegt darin begründet, dass der bewusste Denkvorgang eines Benutzers in einen intuitiven Wahrnehmungsvorgang überführt wird. Durch eine Visualisierung wird der Benutzer beim Erkennen von Informationen entlastet. (Vgl. [Sch15], S. 1)

Die folgenden Ziele der Visualisierung können festhalten werden ([Deu15], S. 7):

- Abbildung von Daten auf visuellen Dimensionen
- Handhaben großer Datenmengen
- Darstellung eines Überblicks von Informationen
- Darstellung von Informationen aus verschiedenen Gesichtswinkeln
- Darstellung von Informationen in unterschiedlichen Detaillierungsgraden
- Gewinnen neuer Erkenntnisse
- Aufzeigen / Erkennen von Zusammenhängen

2.4 Aufgaben

Der Kernansatz der Visualisierung folgt dem Prinzip des Mantra visueller Informationssuche von Ben Shneiderman: „Overview first, zoom and filter, then details on demand.“. Die nachfolgende Aufzählung beschreibt die einzelnen Phasen im Detail ([Shn96]):

- *Überblick:* Die Betrachtung des Gesamtzusammenhangs und das Erkennen von globalen Mustern und Trends.
- *Zoom:* Die genauere Betrachtung einer Teilmenge der Daten.
- *Filter:* Das Auswählen einer Teilmenge basierend auf bestimmten Werten.
- *Details:* Die Betrachtung von Einzeldaten über Objekte, die interaktiv ausgewählt wurden.
- *In Beziehung setzen:* Das Erkennen von Verbindungen und der Vergleich von Werten in den dargestellten Daten.
- *Zeitlicher Verlauf:* Die Nachverfolgung von Aktionen und Erkenntnissen bei Verwendung der Visualisierung.
- *Extrahieren:* Das Markieren, Herausziehen und Weiterverwenden relevanter Daten als Ergebnisse der Visualisierung.

Zusammenfassend lässt sich festhalten, dass nach *Shneiderman* eine Visualisierung den Benutzer bei seiner informationellen Arbeit unterstützen muss. Dabei soll dieser sich einen Überblick über das Angebot machen können und muss dabei die Möglichkeit haben interessante Details herauszoomen und uninteressante herausfiltern zu können. Des Weiteren muss er nach Bedarf weitere Details abrufen und Beziehungen zwischen den Daten herstellen können. Außerdem sollte er die Möglichkeit haben, den eigenen Weg nachzuvollziehen, rückzuverfolgen und eine neue Auswahl treffen zu können.

2.5 Datentypen

Da, wie bereits in Abschnitt 2.3, das Ziel von Visualisierung ist, Daten in visuelle Formate zu transformieren, ist es hilfreich in diesem Zusammenhang einige Datentypen zu definieren, anhand derer eine Kategorisierung vorgenommen werden kann. *Shneiderman* entwickelte hierfür eine „Task by Data Type Taxonomy for Information Visualization“ (kurz: (TTT)), die unter anderem eine Unterscheidung der existierenden Datentypen von Visualisierungen in insgesamt sieben verschiedene Kategorien („one-, two-, threedimensional data, temporal and multi-dimensional, and tree and network data“) vornimmt. Die Idee hinter der

TTT ist, die Datentypen von Visualisierungen zu identifizieren. In der nachfolgenden Tabelle 2.1 sind die sieben verschiedenen Datentypen mit Beispielen ersichtlich ([Shn96], S. 336-343):

Datentyp	**Beispiel**
Eindimensionale Daten (lineare Daten)	Textdokumente, Quellcodes, alphabetische Listen
Zweidimensionale Daten	Geografische Daten, Pläne, Zeitungslayouts
Dreidimensionale Daten	CAD-Modelle z.B. für Architekten
Zeitliche Daten	Zeitstrahl für Projektmanagement oder geschichtliche Veröffentlichungen
Multidimensionale Daten	n-dimensionale Datenbankinhalte
Verzweigungsbaumdaten	Sammlung von Items, die eine Verbindung zu einem übergeordneten Item haben
Netzwerkdaten	Sammlung von Items, die mit einer beliebigen Anzahl von Items verbunden werden können

Tabelle 2.1: Datentypen mit Beispielen

Die Auswahl der Visualisierungstechnik bzw. des Diagrammtyps hängt direkt vom jeweiligen Datentyp, der Menge der Daten und dem Verwendungszweck ab und kann selbstverständlich auch kombiniert werden. Im nachfolgenden Abschnitt 2.6 werden hierfür verschiedene Diagrammtypen vorgestellt.

2.6 Diagrammtypen

Es existiert „ein bunter Stauß“ an unterschiedlichen Diagrammtypen. Es muss jedoch immer beachtet werden, in welchem Big-Data-Kontext diese angewendet werden, da sie sich an spezielle Datentypen richten. (Vgl. [Fre14], S. 321) Nachfolgend wird ein Auszug verschiedener Diagrammtypen vorgestellt, die im Big-Data-Umfeld eingesetzt werden könnten:

Heatmap: Eine Heatmap dient zur Visualisierung von Daten, deren Werte von einem zweidimensionalen Datentyp sind, welche in Form von Farben dargestellt werden. Sie ist hilfreich um in einer großen Datenmenge, intuitiv und schnell, besonders auffällige Werte zu erfassen. (Vgl. [Nau12], S. 87)

Abbildung 2.1: Heatmap zum Klickverhalten auf der Startseite der FH Nordwestschweiz ([FH 15])

Streudiagramm: Ein Streudiagramm verwendet kartesischen Koordinaten um Werte für zwei Variablen eines Datensatzes anzuzeigen. Ein Verändern der Punkte (z.B. Farbe, Form, etc.) ermöglicht es die Anzahl von Variablen zu erhöhen.

Abbildung 2.2: Beispiel eines Streudiagramms ([Wik15])

Liniendiagramm: Ein Liniendiagramm ist ein klassisches Verfahren zur Visualisierung kontinuierlicher Veränderungen. Mehrere Linien ermöglichen den Ver-

gleich von mehreren Datensätzen.

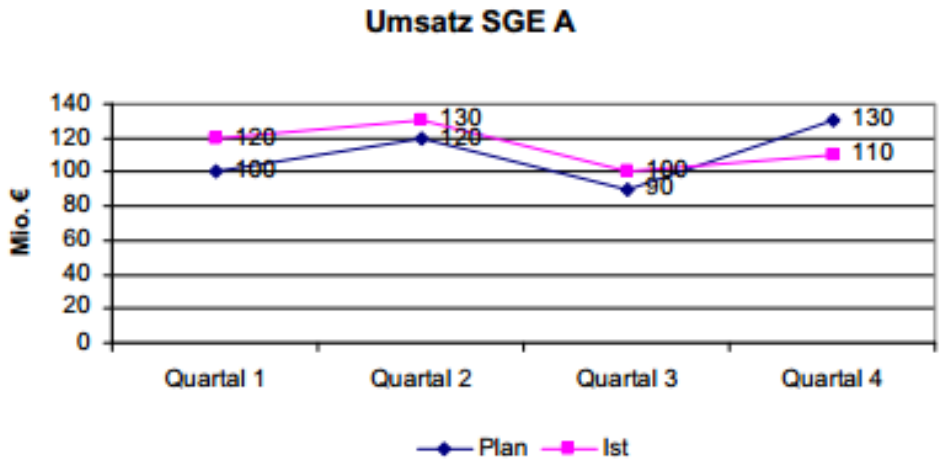


Abbildung 2.3: Beispiel eines Liniendiagramms ([Sch12], S. 91)

Säulen- und Balkendiagramme: Balkendiagramme stellen eine klassische Methode für numerische Vergleiche dar. Sie können in vertikaler, horizontaler, gruppierter oder gestapelter Form verwendet werden.

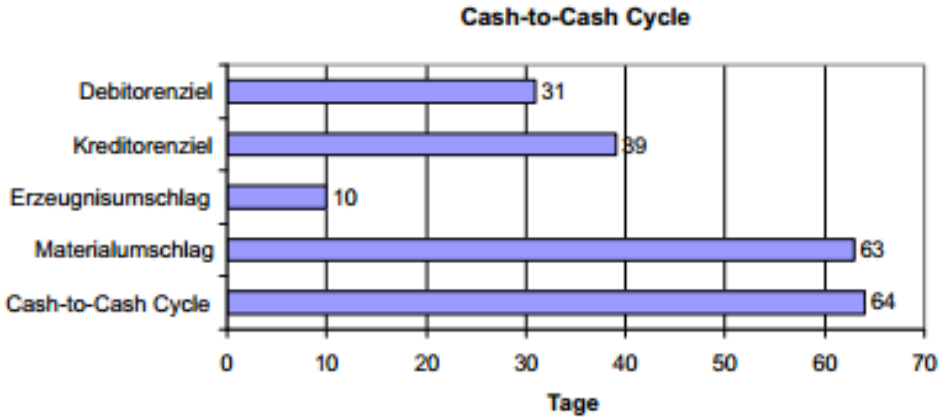


Abbildung 2.4: Beispiel eines Balkendiagramms ([Sch12], S. 88)

Kreisdiagramm: Kreis- oder Kuchendiagramme zeigen Teilmengen einer Gesamtmenge in Form von Kreissektoren. Je größer dabei die Teilmenge eines betrachteten Objektes ist, desto größer ist der Anteil des Kreissektors an der Gesamtmenge (100 % entsprechen 360 Grad) des Vollkreises. ([Sch12], S. 87)

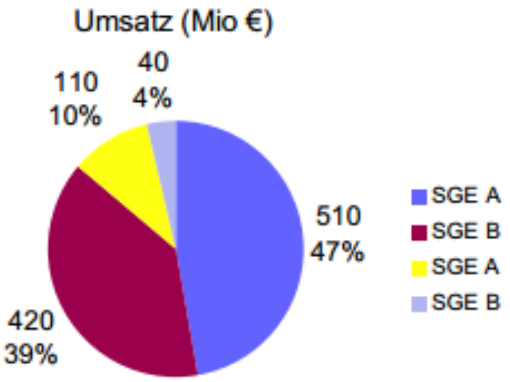


Abbildung 2.5: Beispiel eines Kreis- bzw. Kuchendiagramms ([Sch12], S. 87)

Netzdiagramm: Netz-bzw. Spinnennetzdiagramme helfen dabei für eine ausgewählte Zahl von Beurteilungskriterien Vergleiche durchzuführen, wobei die Skala zur Beurteilung möglichst normiert sein sollte (z.B. durch einheitliche Noten, Punktwerte oder Prozentangaben). Die Werte werden im Netzdiagramm mit Datenpunkten auf den Achsen, ausgehend von dem Mittelpunkt des Netzes, abgetragen. Je weiter ein Wert von der Achse entfernt ist, desto besser (z.B. Zielerreichung) oder schlechter (z.B. Risiken) wird er beurteilt. ([Sch12], S. 97)

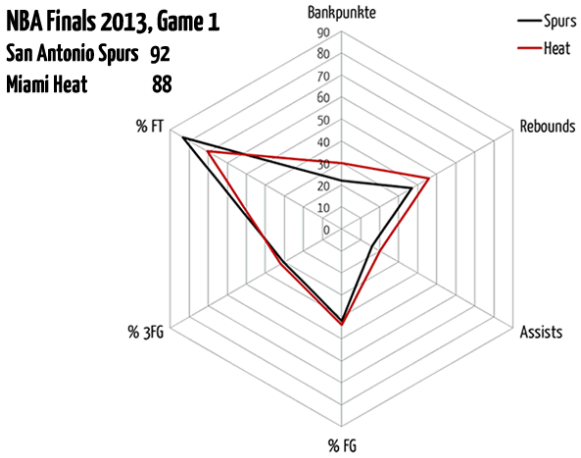


Abbildung 2.6: Beispiel eines Netz- bzw. Spinnennetzdiagramms ([Bas13])

Thematische Karte - Choroplethenkarte: Ein Choroplethenkarte ist eine Thematische Karte bei der die Gebiete im Verhältnis zur Verteilungsdichte des thema-

tischen Objektes eingefärbt, schattiert, gepunktet oder schraffiert sind ([CHU07], S. 76). Die folgende Abbildung 2.7 zeigt eine geographische Landkarte in Verbindung mit der Einfärbung je nach Staatsverschuldung im Verhältnis zum BIP.

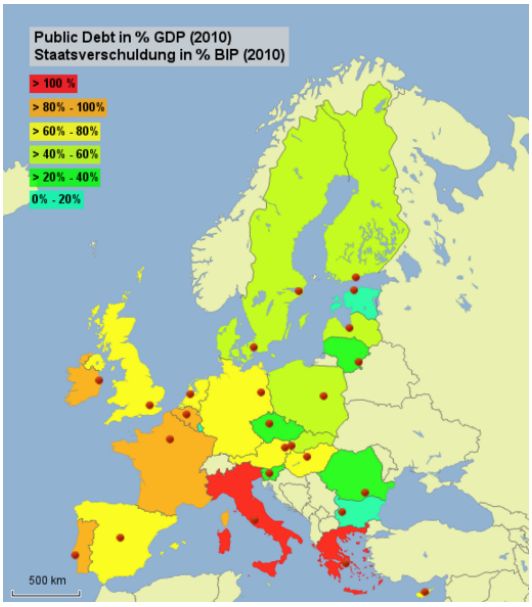


Abbildung 2.7: Staatsverschuldung europäischer Länder in % des BIP im Jahr 2010 ([STE13])

3 Visualisierungskomponenten

In diesem Kapitel wird auf konkrete Visualisierungskomponenten bzw. Visualisierunglösungen des Apache Hadoop Ecosystems eingegangen.

3.1 Apache Zeppelin

Im folgenden Abschnitt wird der Prozess der Verwendung von Apache Zeppelin zur interaktiven Analyse bzw. Visualisierung von Daten eines Apache Hadoop Clusters aufgezeigt.

Apache Zeppelin ist ein „Web-basiertes Notebook“, das interaktive Datenanalysen ermöglicht. Zeppelin stellt dabei ein „mehrsprachiges Backend“ in Form eines Zeppelin-Interpreter-Konzeptes bereit. Dieses erlaubt es verschiedene „Sprachen“ bzw.
Daten-Verarbeitung-Backends in Zeppelin einzubinden. Derzeit bietet Zeppelin Interpreter für Scala (mit Apache Spark), Python (mit Apache Spark), SparkSQL, Hive, Markdown und Shell. Zur Visualisierung der Daten stellt Zeppelin einige grundlegende Diagramme bereit. Visualisierungen sind hierbei jedoch nicht nur auf SparkSQL als Abfragesprache beschränkt. Es kann prinzipiell jede Ausgabe von einer beliebigen „Backend-Sprache“ erfasst und visualisiert werden. Zeppelin stellt ebenfalls die Möglichkeit bereit, Pivot-Tabellen zu erstellen. Dabei können die Werte via Drag & Drop innerhalb eines Pivot-Chart dargestellt werden. So lassen sich beispielsweise einfach Diagramme mit mehreren aggregierten Werten wie *sum*, *count*, *average*, *min*, *max* erstellen. Über das Web-Front-End kann Zeppelin in einer Benutzergruppe gemeinsam genutzt werden. Dabei werden alle Änderungen in Echtzeit angezeigt. Des Weiteren ist es möglich, mit Zeppelin eine Ergebnisseite, ohne Menüstruktur, zu erstellen, um nur die Ergebnisse anzeigen zu lassen. Apache Zeppelin ist Open Source und steht unter der Apache 2 Licensed Software. (Vgl. [Apa15a])

Im Rahmen dieser Arbeit wurde Apache Zeppelin auf der Hortonworks Sandbox (HDP 2.3) manuell nachinstalliert. ([Git15])

In Abbildung 3.1 wird der grobe Aufbau bzw. die Struktur des Zeppelin Notebooks gezeigt. Unter dem Menüpunkt „Notebook“ befindet sich der Teil der Analyse bzw. Visualisierung. Unter dem Menüpunkt „Interpreter“ können die Inter-

preter hinzugefügt oder konfiguriert werden.

Im Prinzip lässt sich die grafische Oberfläche in der Standardeinstellung in vier Boxen unterteilen. So wird in der obersten Box eine Art „Willkommensbox“ bereitgestellt, in der der Benutzer verschiedene Vordefinitionen oder Textausgaben durchführen kann. So kann z.B. über das Command „%md“ (=mark down) etwas auf dem Bildschirm ausgeben werden. Diese Box ist allerdings optional und muss nicht zwingend verwendet bzw. eingeblendet werden. In der zweiten Box, welche die Überschrift „Prepare Data“ trägt, kann der Benutzer festlegen wie die Daten im Vorfeld „präpariert“ werden sollen, um sie anschließend im HDFS von Hadoop abzulegen. Innerhalb der dritten Box „Load Data Into Table“ werden die Daten in Tabellen innerhalb von Apache Spark geschrieben. Die folgende Abbildung 3.1 zeigt einen Ausschnitt der ersten drei Boxen mit Testdaten.

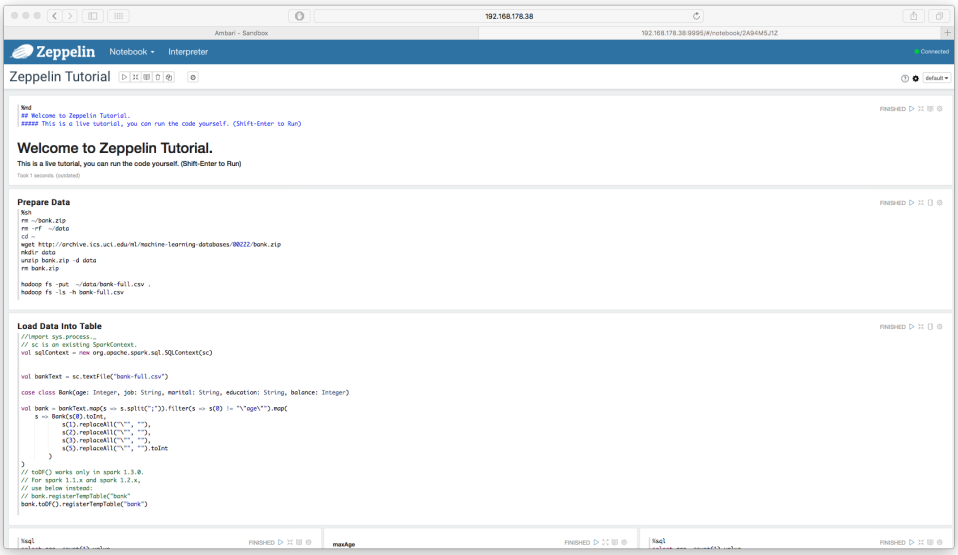


Abbildung 3.1: Aufbau Zeppelin Box 1-3 (Eigene Darstellung)

Die vierte Box ist in der Standardeinstellung eine Reihe von drei nebeneinanderliegenden Boxen. Diese stellen dabei „Interaktionsboxen“ dar. Innerhalb dieser können jeweils die Interpreter angesprochen werden. So kann bspw. über den Befehl „%spark“ oder „%sql“ der jeweilige Interpreter angesprochen werden. Innerhalb der „Interaktionsboxen“ sind ebenfalls die oben erwähnten standardmäßigen Diagramme ersichtlich. Des Weiteren ist in der Mitte der Boxreihe eine Pivot-Tabelle ersichtlich. Die Fünfte Box ist ebenfalls optional und muss nicht zwingend verwendet bzw. eingeblendet werden. Auch hier können verschiedene Textausgaben o.ä. ausgegeben werden. In der folgenden Abbildung 3.2 sind die Boxen 4-5 mit Testdaten ersichtlich.

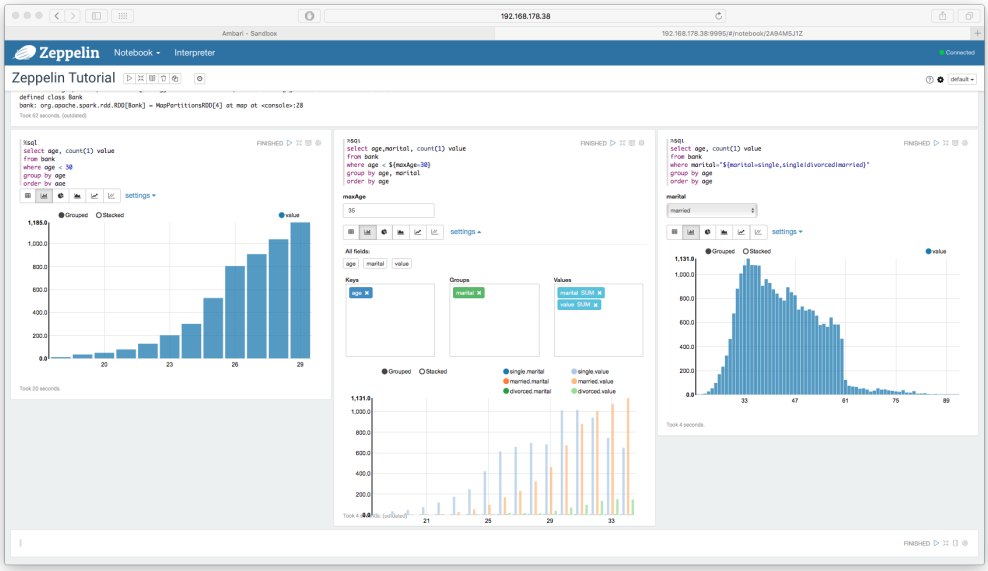


Abbildung 3.2: Aufbau Zeppelin Box 4-5 (Eigene Darstellung)

3.2 Tableau

Tableau ist eine kostenpflichtige Lösung von gleichnamigen Hersteller „Tableau“ um mit Daten aus dem Apache Hadoop Ecosystem arbeiten zu können. Die kostenfreie Variante von Tableau (Tableau Public) ermöglicht dabei keine Verbindung zu einem Hadoop Ecosystem. Im Rahmen dieser Seminararbeit wurde die 14-tätige Testversion von Tableau Desktop verwendet, um aufzuzeigen, welche Visualisierungsmöglichkeiten Tableau im Umfeld bzw. in Verbindung mit dem Hadoop Ecosystem bietet.

Tableau ermöglicht die Verbindung zu Apache Hadoop, wie die meisten BI-Tools, über eine ODBC-Verbindung. Um die Verbindung mit der Virtuellen Maschine von Hortonworks herstellen zu können musste hierfür ein von Hortonwork bereitgestellter ODBC-Treiber heruntergeladen und installiert werden. (Vgl. [Hor15c])

Sobald die Verbindung hergestellt wird, kann Tableau verwendet werden. So lassen sich Querys, Analysen oder Visualisierungen durchführen, in dem Tableau die Daten über Hive aus dem Hadoop Ecosystem lädt. Dabei bestehen drei Optionen die Daten innerhalb von Tableau zu verwenden: Die Option „Connect Live“ ermöglicht es, die Daten über eine „Live-Verbindung“ zu verwenden. Dabei besteht allerdings die Abhängigkeit, dass die Verbindungsgeschwindigkeit der Datenquelle die Performance von Tableau bestimmt. Die zweite Option ermöglicht es, die Daten komplett in Tableau zu importieren. Die dritte Option stellt die Mög-

lichkeit bereit, selbst auszuwählen, welche Daten in Tableau importiert werden. Tableau bietet umfangreiche Visualisierungstechniken. Es lassen sich bspw. innerhalb von Tableau Ad-hoc-Visualisierungen erstellen, so dass Muster und Ausreißer in den Daten, die in im Hadoop-Cluster gespeichert sind, erkennbar gemacht werden können. Des Weiteren kann Tableau dazu verwendet werden, um Visualisierungen mit anderen Benutzern zu teilen. So können beispielsweise operativen Kennzahlen, KPIs oder Monitoring-Dashboards erstellt und zugänglich gemacht werden. Die folgende Abbildung 3.3 zeigt einen Ausschnitt der grafischen Oberfläche von Tableau. In von Tableau wurde dabei eine Choroplethenkarte mit Testdaten erstellt.

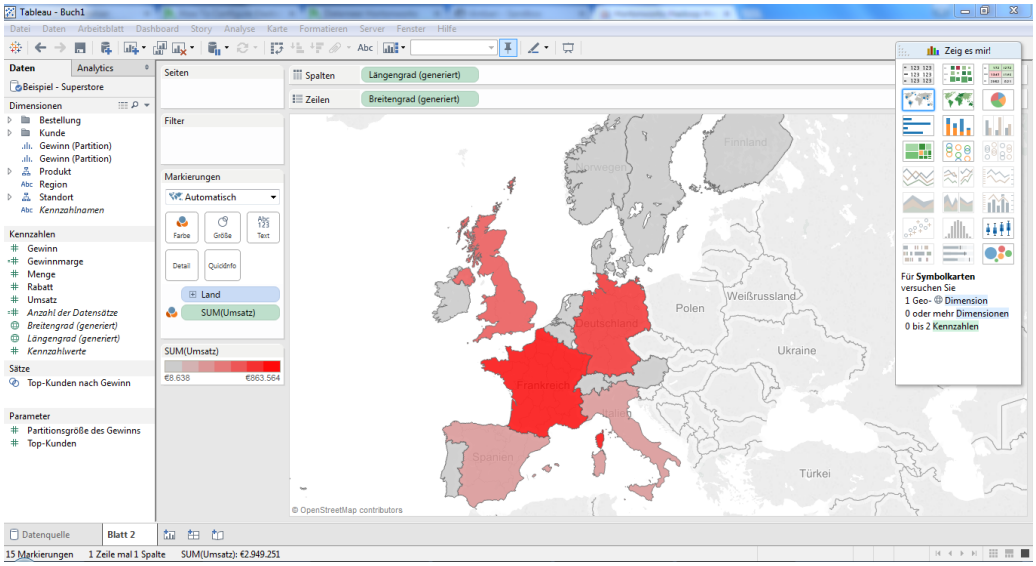


Abbildung 3.3: Ausschnitt der grafischen Oberfläche von Tableau (Eigene Darstellung)

3.3 Kibana

Kibana ist eine browser-basierte Anwendung um Daten zu analysieren bzw. zu visualisieren. Kibana ist eine Open Source-Lösung (Apache-Lizenz) und kann mit dem Hadoop Ecosystem nur in Verbindung mit Elasticsearch für Apache Hadoop verwendet werden. Die Elasticsearch-Engine lässt sich in die Hortonworks Data Platform in Verbindung bringen, um diese und Kibana verwenden zu können. Dafür existier ein ES-Hadoop Connector, welcher es ermöglich Daten zwischen Hadoop/HDFS und Elasticsearch auszutauschen. Elasticsearch ermöglicht eine near Real-Time-Suche und einen Zugriff auf Information im Hadoop. Die folgende Abbildung 3.4 zeigt die Architektur zur Verwendung von Elasticsearch im Hadoop-Umfeld. Darauf aufbauend kann dann Kibana für die Visualisierung verwendet werden. (Vgl. [Hor14b])

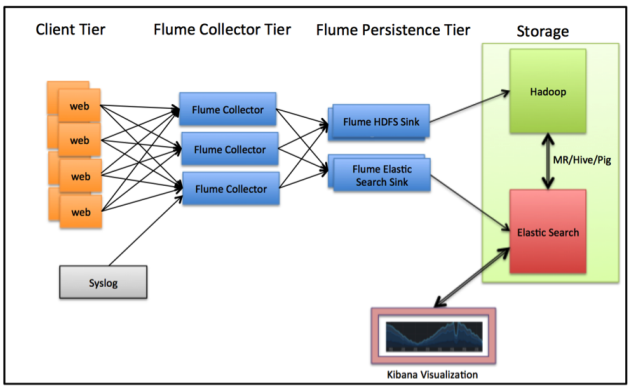


Abbildung 3.4: Architektur Elastic Search/Kibana ([Hor14c])

Kibana als Visualisierungslösung ermöglicht es, nativ mit allen Daten in Elasticsearch, benutzerdefinierte Dashboards zu erstellen. Diese können dabei auch in Form von Echtzeit-Dashboards verwendet und anderen Benutzern innerhalb einer Organisation geteilt werden. In Kibana können Daten, „über den Weg“ Elasticsearch, aus Hadoop/HDFS, aber auch weiteren Lösungen wie Apache Flume oder Fluentd visualisiert werden. (Vgl. [Hor14a]) Die folgende Abbildung 3.5 zeigt einen Ausschnitt der grafischen Oberfläche von Kibana.

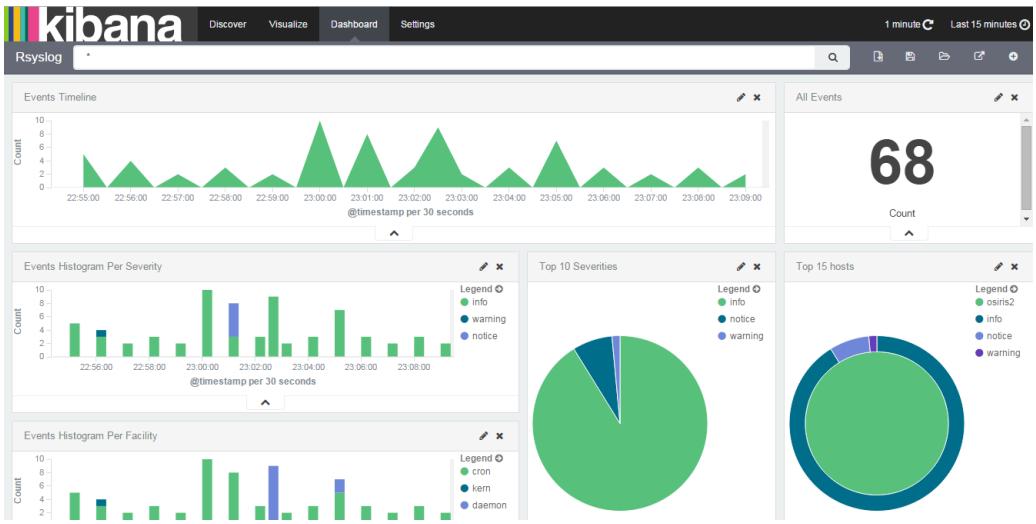


Abbildung 3.5: Ausschnitt der grafischen Oberfläche von Kibana (Eigene Darstellung)

3.4 Datameer

Die Lösung Datameer wurde im Rahmen dieser Arbeit innerhalb einer kostenfreien 14-tätigen Testversion von Hortonworks bereitgestellten Virtuellen Maschine (HDP 2.1) verwendet, da Datameer ein lizenzpflichtiges Produkt ist. Innerhalb der Virtuellen Maschine ist Datameer mit der Hortonworks Sandbox kombiniert, so dass eine sofort nutzbare Hadoop Evaluierungsumgebung verfügbar war. (Vgl. [Hor15a])

Datameer baut auf einer Excel-style Tabellenform auf. Dabei kann zunächst definiert werden, welche Daten analysiert bzw. visualisiert werden sollen. Es stehen viele Werkzeuge zur Datenvisualisierung bereit. Datameer stellt z.B. iterative „Point-and-click Visualisierungen“ und „Drag-and-Drop-Visualisierungen“ bereit, welche es ermöglichen, sich einen bestimmten Datensatz mit einem Klick visualisieren zu lassen. Des Weiteren ist Datameer optimiert für große Datenmengen und kann mehrere Datenquellen unterschiedlicher Art anbinden. Datameer verwendet dabei HDFS und MapReduce des Hadoop Ecosystems. Datameer „harmoniert“ ebenfalls mit Flume und Pig oder Hive. Die folgende Abbildung 3.6 zeigt die Einordnung von Datameer im Hadoop-Kontext. (Vgl. [Mes14] S. 1-11)

Datameer ist eine erweiterbare Lösung, die auch in bereits vorhandene Data-Warehouse- oder BI-Lösungen integriert werden kann. Datameer ermöglicht es zudem über die Datameer App Market-Plattform vorgefertigte analytische Applikationen Anderen Benutzern zur Verfügung zu stellen bzw. diese selber zu verwenden. Die folgende Abbildung 3.7 zeigt einen Ausschnitt der grafischen Oberfläche von Datameer.

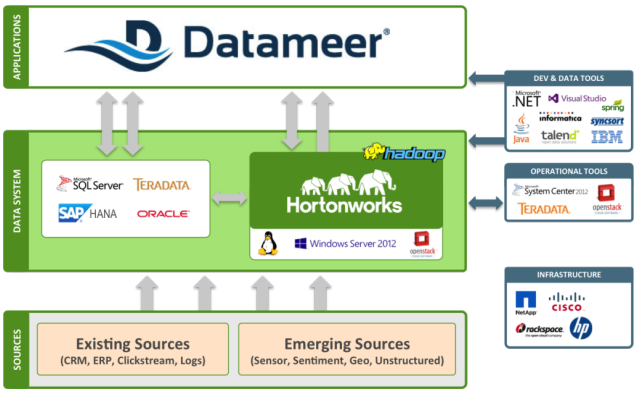


Abbildung 3.6: Einordnung von Datameer im Hadoop-Kontext ([Hor15a])

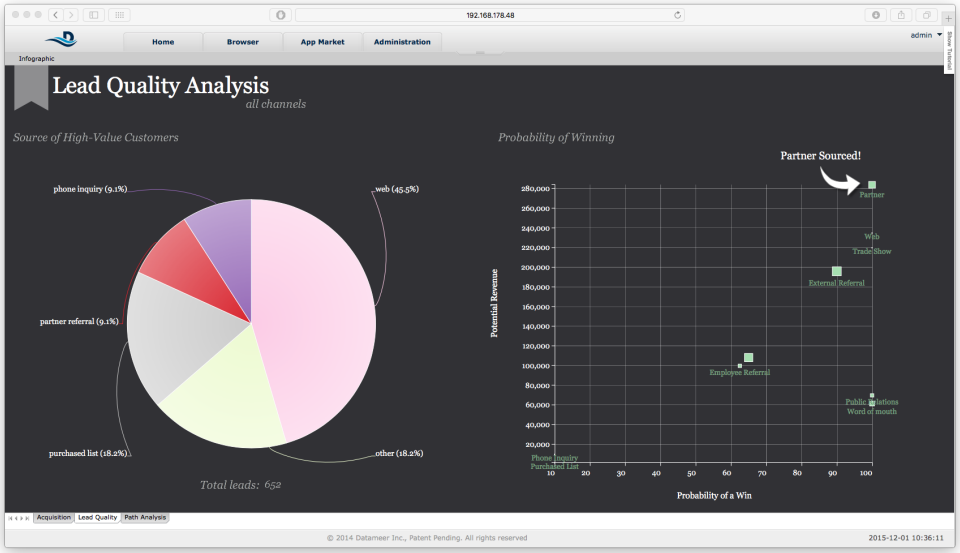


Abbildung 3.7: Ausschnitt der grafischen Oberfläche von Datameer (Eigene Darstellung)

4 Zusammenfassung & Ausblick

In dieser Seminararbeit wurden die Visualisierungsmöglichkeiten mittels dem Hadoop Ecosystem betrachtet. Dazu wurden zunächst die Grundlagen der Visualisierung näher erläutert, in dem der Begriff Datenvisualisierung eingeführt sowie die Herausforderungen, Ziele und Aufgaben der Visualisierung beschrieben wurden. Zudem wurden verschiedene Datentypen bzw. Diagrammtypen vorgestellt. Anschließend wurden vier verschiedene Visualisierungskomponenten vorgestellt und deren Anwendung bzw. Funktionsweise beschrieben. Die Seminararbeit wurde dabei auf die Lösungen Apache Zeppelin, Tableau, Kibana und Datameer beschränkt. Während der Anwendung bzw. Evaluation dieser Lösungen konnte festgestellt werden, dass das Apache Hadoop Ecosystem eine Vielzahl von Möglichkeiten bietet, verschiedene Lösungen anzubinden bzw. zu verwenden. Es muss jedoch beachtet werden, dass zwei der vorgestellten Lösungen (Tableau und Datameer) lizenzpflichtig sind und im Rahmen der Projektgruppe ggf. Kosten verursachen könnten. Diese Kosten sollten jedoch nicht als „KO-Kriterium“ gesehen werden diese Tools im Rahmen der Projektgruppe nicht näher zu betrachten, da gerade diese Lösungen sehr umfangreiche und unterschiedliche Visualisierungsmöglichkeiten bieten. Die Herausforderung für die Projektgruppe für die kommende Zeit wird in jedem Fall darin bestehen, aus den vielen Lösungen und Tools das Optimum für den jeweiligen Anwendungsfall zu ermitteln.

Literatur

[Apa15a] APACHE: *Apache Zeppelin*. http://zeppelin-project.org/. Version: 2015. – zuletzt abgerufen am 11.11.2015

[Apa15b] APACHE.ORG: *Hadoop Wiki*. http://wiki.apache.org/hadoop/. Version: 2015. – zuletzt abgerufen am 30.10.2015

[Bar13] BARON, Pavlo: *Big Data für IT-Entscheider: Riesige Datenmengen und moderne Technologien gewinnbringend nutzen*. 2013

[Bas13] BASKETBALL.DE: *Beispiel eines Netzdiagramms - NBA Finals 2013, Game 1*. http://basketball.de/wp-content/uploads/2013/06/nba_finals2013_game1_stats.gif. Version: 2013. – zuletzt abgerufen am 30.11.2015

[BI006] *BI-Spektrum: Fachzeitschrift für Business Intelligence und Data Warehousing*. Troisdorf : SIGS-DATACOM, 2006. – ISBN 1862–5789

[BIT14] BITKOM: *Big-Data-Technologien – Wissen für Entscheider: Leitfaden*. https://www.bitkom.org/Publikationen/2014/Leitfaden/Big-Data-Technologien-Wissen-fuer-Entscheider/140228_Big_Data_Technologien_Wissen_fuer_Entscheider.pdf. Version: 2014. – zuletzt abgerufen am 02.11.2015

[CHU07] CHEN, Chun-houh ; HÄRDLE, Wolfgang ; UNWIN, Antony: *Handbook of data visualization*. Berlin and London : Springer, 2007 (Springer handbooks of computational statistics). – ISBN 3540330364

[Deu15] DEUSSEN, Oliver: *Datenvisualisierung: Computergraphik II*. https://www.inf.tu-dresden.de/content/institutes/smt/cg/teaching/lectures/CG2WS0203/secure/datenvis_script.pdf. Version: 2015. – zuletzt abgerufen am 30.11.2015

[Die07] DIEHL, Stephan: *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2007. http://dx.doi.org/10.

1007/978-3-540-46505-8. http://dx.doi.org/10.1007/978-3-540-46505-8. – ISBN 9783540465041

[FH 15] FH NORDWESTSCHWEIZ: *Heatmap der Startseite der FH Nordwestschweiz*. http://www.fhnw.ch/technik/bachelor/informatik/usability-labor/heatmap-der-startseite-der-fhnw. Version: 2015. – zuletzt abgerufen am 30.11.2015

[Fre14] FREIKNECHT, Jonas: *Big Data in der Praxis: Lösungen mit Hadoop, HBase und Hive ; Daten speichern, aufbereiten, visualisieren*. München : Hanser, 2014. – ISBN 9783446439597

[Git15] GITHUB: *An Ambari Service for Zeppelin*. https://github.com/hortonworks-gallery/ambari-zeppelin-service. Version: 2015. – zuletzt abgerufen am 11.11.2015

[Hor14a] HORTONWORKS: *Combine Apache Hadoop and Elasticsearch to Get the Most of Your Big Data*. http://de.slideshare.net/hortonworks/hortonworks-elastic-searchfinal. Version: 2014. – zuletzt abgerufen am 15.11.2015

[Hor14b] HORTONWORKS: *Enable Real-Time Search and Analytics*. https://hortonworks.com/wp-content/uploads/2013/09/Elasticsearch-Hortonworks-Solutions-Brief.pdf. Version: 2014. – zuletzt abgerufen am 15.11.2015

[Hor14c] HORTONWORKS: *How To Configure Elasticsearch on Hadoop with HDP*. http://hortonworks.com/blog/configure-elastic-search-hadoop-hdp-2-0/. Version: 2014. – zuletzt abgerufen am 11.11.2015

[Hor15a] HORTONWORKS: *Datameer*. http://de.hortonworks.com/partner/datameer/. Version: 2015. – zuletzt abgerufen am 11.11.2015

[Hor15b] HORTONWORKS: *Hortonworks data platform*. http://hortonworks.com/. Version: 2015. – zuletzt abgerufen am 30.10.2015

[Hor15c] HORTONWORKS: *HOW TO: Connect Tableau to Hortonworks Sandbox*. http://de.hortonworks.com/kb/how-to-connect-tableau-to-hortonworks-sandbox/. Version: 2015. – zuletzt abgerufen am 11.11.2015

[IA15] ISHWARAPPA, J. ; ANURADHA, J.: A Brief Introduction on Big Data 5Vs Characteristics and Hadoop Technology. In: *Procedia Computer Science, 2015, Vol.48, pp.319-324* (2015)

[IBM15] IBM: *Analytics: Big Data in der Praxis: Wie innovative Unternehmen ihre Datenbestände effektiv nutzen*. `http://www-935.ibm.com/services/de/gbs/thoughtleadership/GBE03519-DEDE-00.pdf`. Version: 2015. – zuletzt abgerufen am 02.11.2015

[Kob15] KOBEK, Wolfgang: *Datenvisualisierung: Mehr als Balken und Torten: Datenvisualisierung gibt BI-Projekten den richtigen Schliff und liefert neue Einblicke*. `https://jaxenter.de/datenvisualisierung-bi-projekte-15797`. Version: 2015. – zuletzt abgerufen am 15.11.2015

[Mes14] MESNEY, James: *Datameer - May 2014 Hadoop MeetUp*. `http://de.slideshare.net/huguk/datameer-huguk-meetup-may-2014`. Version: 2014. – zuletzt abgerufen am 11.11.2015

[Nau12] NAUTH, D.: *Durch die Augen meines Kunden: Praxishandbuch für Usability Tests mit einem Eyetracking System*. Diplomica Verlag, 2012. – ISBN 9783842883642

[Neu15] NEUMANN, Alexander: *Big Data: Apache Spark tritt aus dem Schatten von Hadoop heraus*. `http://www.heise.de/developer/meldung/Big-Data-Apache-Spark-tritt-aus-dem-Schatten-von-Hadoop-heraus-284`
`html`. Version: 2015. – zuletzt abgerufen am 30.11.2015

[Sch12] SCHÖN, Dietmar: *Planung und Reporting im Mittelstand: Grundlagen, Business Intelligence und Mobile Computing*. Wiesbaden : Springer Gabler, 2012. – ISBN 9783834936042

[Sch15] SCHWARTZ, Dieter: *Mehr Information durch Visualisierung von Daten? Konventionelle und innovative Visualisierungstechniken*. `http://www.b-i-t-online.de/archiv/1999-04/nach1.htm`. Version: 2015. – zuletzt abgerufen am 15.11.2015

[Shn96] SHNEIDERMAN, B.: *The eyes have it: a task by data type taxonomy for information visualizations: Visual Languages, 1996. Proceedings., IEEE Symposium on*

[STE13] STEPMAP: *Beispiel einer Choroplethenkarte - Staatsverschuldung der Staaten Europas in % BIP*. `https://www.stepmap.de/landkarte/`

staatsverschuldung-der-staaten-europas-in-bip-151765. Version: 2013. – zuletzt abgerufen am 30.11.2015

[Whi09] WHITE, T.: *Hadoop: The Definitive Guide: The Definitive Guide*. O'Reilly Media, 2009. – ISBN 9780596551360

[Wik15] WIKIMEDIA.ORG: *Beispiel eines Streudiagramms*. https://upload.wikimedia.org/wikipedia/de/thumb/e/ef/Lang-breit.svg/300px-Lang-breit.svg.png. Version: 2015. – zuletzt abgerufen am 30.11.2015

[Wit15] WITHANAWASAM, J.: *Apache Mahout Essentials*. Packt Publishing, 2015. – ISBN 9781783555000

Abschließende Erklärung

Ich versichere hiermit, dass ich meine Seminararbeit „Visualisierungsmöglichkeiten mittels dem Apache Hadoop Ecosystem“ selbständig und ohne fremde Hilfe angefertigt habe, und dass ich alle von anderen Autoren wörtlich übernommenen Stellen wie auch die sich an die Gedankengänge anderer Autoren eng anlegenden Ausführungen meiner Arbeit besonders gekennzeichnet und die Quellen zitiert habe.

Oldenburg, den 2. Dezember 2015

Klassische vs. Agile Softwareentwicklung: Einsatz von SCRUM in der Projektgruppe

Seminararbeit

Themenersteller:	Prof. Dr.-Ing. habil. Jorge Marx Gómez
Betreuer:	M. Eng. & Tech. Viktor Dmitriyev Dr.-Ing. Andreas Solsbach
Vorgelegt von:	Felix Schoelzel Felix.Ivo.Schoelzel@uni-oldenburg.de
Abgabetermin:	2. Dezember 2015

Inhaltsverzeichnis

Abkürzungen

HDFS Hadoop Distributed Filesystem

Abbildungen

1 Einleitung

Ein Projekt bezeichnet ein Vorhaben, das durch verschieden Variablen wie zeitliche, finanzielle oder personelle Beschränkungen und die Zieldefinition in seiner Gesamtheit einmalig ist und sich von anderen Vorhaben unterscheidet. (vgl. DIN 69901) Somit sind die Aufgaben und Probleme, die es zu lösen gilt, in ihrer Form neuartig und erfordern eine strukturierte Planung und Steuerung. Projekte besitzen in der Regel eine hohe Komplexität. Zusammen mit der Neuartigkeit und den mitunter fehlenden Erfahrungswerten und Know-How ergibt sich ein gewisses Risiko bezüglich des erfolgreichen Abschließens eines Projektes. Dies gilt auch für IT-Projekte wie beispielsweise die Entwicklung einer neuen Software. Dabei erreichen Projekte in der IT oftmals nicht die geplanten Zielvorgaben. Gründe können die Überschreitung des Budgets, die Nichteinhaltung zeitlichen Fristen oder auch Abweichungen der geplanten Resultate beziehungsweise Funktionen sein. Die meist interdisziplinären Herausforderungen erfordern ein Arbeitsteam, das sich aus Mitgliedern mit unterschiedlichen Fachkompetenzen zusammensetzt oder Mitgliedern, die innerhalb des Projektes verschiedene Schwerpunkte bearbeiten. Dabei müssen die Arbeitsabläufe und –ergebnisse der einzelnen Individuen vereinheitlicht und transparent gemacht werden, um kollaborative Prozesse ermöglichen und unterstützen zu können.

Die Koordination beziehungsweise Organisation von Arbeitsgruppen und deren Leistungserbringung wird durch Projektmanagement-Methoden bewerkstelligt. Bei der Softwareentwicklung haben sich eine Reihe einsetzbarer Methoden entwickelt, die sich in Aufbau und Prioritäten unterscheiden. Allgemein kann zwischen klassischen und agilen Ansätzen unterschieden werden.

Bis Ende der sechziger Jahre gab es bei der Softwareentwicklung keine standardisierten Abläufe. Die immer komplexeren Anforderungen wurden durch sogenanntes „Code and fix“-Vorgehen abgearbeitet, bis eine zufriedenstellende Lösung erreicht wurde. Dies ist in vielerlei Hinsicht problematisch, da die Arbeit für außenstehende kaum nachvollziehbar war, der Code sich nur sehr schwer Erweitern lies und sich die hohe Komplexität nur durch wenige, einst als „Cowboy-Programmers“ bezeichnete Fachkräfte bewältigen lies.

Durch die Formalisierung der Abläufe wurde Anfang der siebziger Jahre versucht, diese Komplexität entgegenzuwirken. So wurden verschiedene Modelle wie

beispielsweise das Wasserfallmodell von Royce entwickelt, um die Softwareentwicklung strukturierter und planbarer zu gestalten. [Gre10] Ursprung dieser Überlegungen kamen aus dem herkömmlichen Projektmanagement und legten ein hohes Augenmerk auf die Prozesse. Die Prioritäten liegen dabei in einer ausführlichen Planung und konsequente Dokumentation der einzelnen Entwicklungsphasen.

Durch neue informationstechnische Konzepte wie beispielsweise die Objektorientierung, erhöhte Anforderungen an Qualität oder Sicherheit und größere Projektdimensionen entstanden weiterentwickelte Ansätze zur formalisierten Projektdurchführung. Ende der achtziger Jahre stellte Boehm sein Spiralmodells vor, bei dem die einzelnen Phasen des Wasserfallmodells nicht iterativ sondern in Form einer Spirale mit wiederkehrenden Elementen aufeinander folgen. Dabei wurde unter anderem das Risikomanagement stärker hervorgehoben und die Planung der einzelnen Phasen nicht an den Projekt- sondern den jeweiligen Phasenbeginn gelegt. [Eck] Das V-Modell mit seiner Heraushebung von Qualitätsmanagement und intensiven Testabläufen stellt einen weiteren Nachfolger des Wasserfalls dar.

Diesen herkömmlichen Konzepten stehen seit den neunziger Jahren agile Projektmanagement-Methoden gegenüber. Diese legen ihre Schwerpunkte im Kontrast zu den ingenieurswissenschaftlichen Ansätzen nicht auf eine prozessorientierte Betrachtung mit möglichst exakter Planung und anschließender Implementierung. [FST14] Vielmehr steht der Entwicklungsgedanke als kreativer schöpferischer Akt im Mittelpunkt. Das vorrangige Ziel von agilen Verfahren ist die Erstellung einer bedarfserfüllenden Software, nicht einer möglichst prozesskonformen Abarbeitung der einzelnen Entwicklungsschritte und deren Dokumentation. Der agile Ansatz richtet sich flexibel an sich schnell verändernde Aufgabenstellungen und bietet die Möglichkeit, sich zur Laufzeit an sich verändernde Anforderungen und Rahmenbedingungen anzupassen. So werden zwar wie bei den klassischen Methoden im Vorfeld die Ziele definiert und festgehalten, Abweichungen aber nicht ausgeschlossen. Um dies zu gewährleisten besteht ein regelmäßiger Austausch zwischen Auftraggeber und dem Entwicklungsteam.

Da sowohl klassische, als auch agile Methoden Vor- und Nachteil mit sich bringen, lässt sich keine generelle Aussage darüber treffen, welche Vorgehensweise geeigneter ist. Vielmehr muss in Einzelfall unter Berücksichtigung der Projektbedingungen abgewägt werden, welche der beiden Richtungen tendenziell zielführender ist.

1.1 Motivation

Innerhalb der Projektgruppe muss ebenfalls eine Entscheidung über die Wahl der eingesetzten Projektmanagement-Methodik getroffen werden. Hier fließen unterschiedliche Faktoren in den Entscheidungsprozess ein wie z.B. die Gruppengröße, die gesetzten Anforderungen und Ziele, Erfahrungswerte der Mitglieder mit bereits eingesetzten Methoden oder die Laufzeit des Vorhabens. Da sich die Projektmanagement-Methodik während der Laufzeit des Projekts nur mit großem Aufwand umstellen lässt, hat die Entscheidung weitreichenden Einfluss auf die Durchführung des Projekts. Diese Seminararbeit soll im ersten Zuge einen Überblick über die häufigsten Vertreter beider Ansätze geben und anschließend auf die Eignung für die Projektgruppe eingehen.

2 Grundlagen

Um einen Überblick über die klassischen und agilen Projektmanagementmethoden zu gewinnen, werden nachfolgend je drei Vertreter beider Kategorien vorgestellt.

2.1 Klassische Methoden

Die klassischen Methoden, wie beispielsweise das Wasserfall-, das V- oder das Spiral-Modell, werden bereits seit Anfang der siebziger Jahre zur Unterstützung der Planung und Steuerung von IT-Projekten eingesetzt. Durch ingenieurswissenschaftliche Prozessorientierung bei der Softwareentwicklung werden eine einfachere Ablaufplanung und somit auch eine bessere Planungssicherheit ermöglicht. Jedoch machen die weit im Voraus geplanten einzelnen formalisierten Prozesse oder auch Phasen die Methoden unflexibel für unstabile Rahmenbedingungen oder Chance-Requests, also sich ändernde Anforderungen während der Projektdurchführung. Zudem besteht das Risiko, konzeptionelle Fehler erst gegen Ende des Projekts festzustellen was deutlich aufwendigere Korrektureingriffe erfordert. [Riea]

2.1.1 Wasserfall-Modell

Wie bereits in der Einleitung erwähnt, ist das Wasserfall-Modell von Winston Royce der älteste und bekannteste Vertreter der Projektmanagement-Methoden mit dem Ziel, die unkoordinierten Arbeitsweisen der damaligen Softwareentwicklung durch eine formalisierte Modellierung einen geeigneten Rahmen zu geben. Dieser Rahmen sollte die einzelnen Prozessschritte von Softwareprojekten in eine einheitliche Form bringen, um die Fehleranzahl und somit Qualitätseinbußen, wie auch die Gefahr des Scheiterns von solchen Vorhaben zu reduzieren. Dabei griff er das, von Herbert Benington bereits 1956 veröffentlichte, Nine Phase Stage-Wise Model auf und erweiterte es um Rückführungsschritte. [The04] Diese beschreiben, dass eine Fehleranalyse und Optimierung im Fall von auftretenden Fehlern und Problemen in einer der vorgelagerten Entwicklungsphasen ihren Ursprung haben kann und dort eliminiert werden sollten, falls dies nicht in der aktuellen Phase mög-

lich ist. Beide Modelle versuchen auf diese Weise, die damals übliche „Code and Fix“-Mentalität zu unterbinden, da sich die Programmcodestruktur im Laufe der Projektdurchführung zusehends verkompliziert und eine Erweiterung, Verbesserung oder Korrektur des Codes nur schwer umsetzen lässt. [DME05]

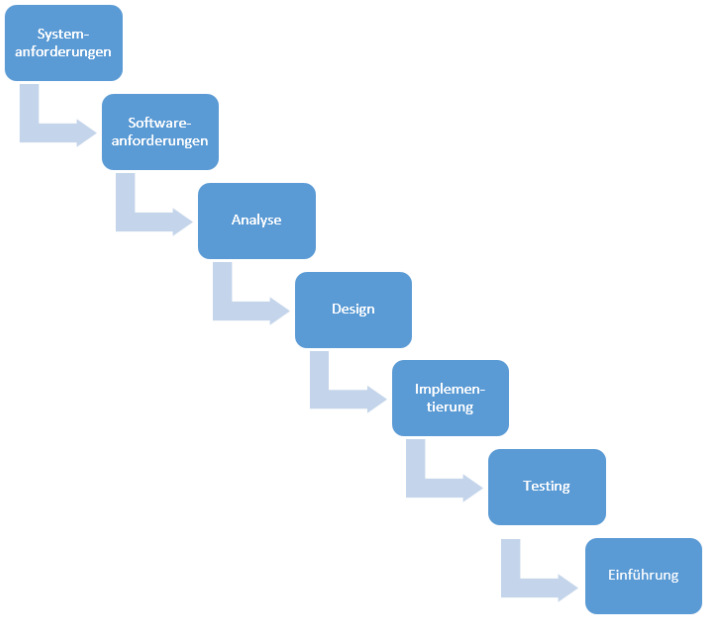


Abbildung 2.1: Wasserfall-Modell nach W. Royce

Anders als Benington unterteilt Royce sein Modell in sieben aufeinander folgende Phasen (vgl. Abb. 2.1), weshalb sein Modell auch als Phasenmodell kategorisiert wird. Diese werden nacheinander durchlaufen wobei in die jeweils nächste Phase erst übergegangen werden darf, wenn Tests und Qualitätsprüfung in ausreichendem Maße durchgeführt wurden und alle zuvor definierten Ergebnisse erfüllt sind. In der ersten Phase Systemanforderungen werden die nichtfunktionalen Anforderungen mit dem Auftraggeber erarbeitet und im Lastenheft festgehalten. Hierbei werden Aspekte wie Kosten, Sicherheit, Laufzeit oder die Dokumentation festgehalten. In der Phase Softwareanforderungen wurden anschließend die funktionalen Anforderungen an die eigentliche Software erarbeitet. Dabei werden die zu erreichenden Leistungen und Funktionen definiert und ebenfalls im Lastenheft festgehalten. Die nächste Phase Anforderungsanalyse dient zur Überprüfung der zuvor erhobenen Anforderungen auf Machbarkeit, Notwendigkeit und Wichtigkeit. Am Ende dieses Entwicklungsschritts wird meist ein Pflichtenheft formuliert, dass sowohl für den Arbeitgeber, als auch das Entwicklungsteam bindend ist. Es beschreibt zusätzlich in welchem Umfang Anforderungsänderungen zulässig sind. Bei Konflikten nach Projektabschluss können sich beide Par-

teien auf dieses Dokument berufen. Die Design-Phase dient zur Bestimmung der eingesetzten Technologien, der Software- und gegebenenfalls der Hardwarearchitektur. Hierbei werden üblicherweise verschiedene Diagramme zur Beschreibung eingesetzt. Auf dieser Grundlage wird anschließend mit der Implementierung der Software begonnen. Anschießend beginnt die Testing-Phase, die zur Fehlereliminierung und Qualitätssteigerung Komponenten-, Integrations- und Systemtests durchführt. Nach Abschluss aller Tests und Abnahme der entwickelten Software durch den Kunden startet die Einführungs-Phase. In dieser letzten Phase wird das fertige IT-Produkt bei dem Auftraggeber installiert beziehungsweise in das Unternehmen integriert und lauffähig gemacht. Vertraglich vereinbarte Wartungsaufgaben sind ein zusätzlicher Bestandteil dieser Phase. [Riea] Das Wasserfall-Modell ist bis heute eine der beliebtesten Projekt-management-Methoden.

2.1.2 Spiral-Modell

Das von Barry W. Boehm im Jahr 1988 veröffentlichte Spiralmodell ist ein weiteres Vorgehensmodell für die Softwareentwicklung. Dabei wird eine hohe Priorität auf die Reduzierung der Risiken während der Projektabwicklung gelegt. Das Modell besteht dabei aus iterativen Zyklen mit wiederkehrenden Schritten. [Bau13] Bei jedem Entwicklungszyklus werden die vier Aktivitäten „Definierung der Ziele, Findung von Alternativen“, „Bewerten der Alternativen, Minimierung von Risiken“, „Erstellung eines Prototypen und Überprüfung“ und „Planung der Projektfortsetzung“ durchlaufen (vgl. Abb. 2.2). Jeder Zyklus ist dabei vergleichbar mit einer der aufeinander folgenden Phasen des Wasserfall-Modells. [INf15]

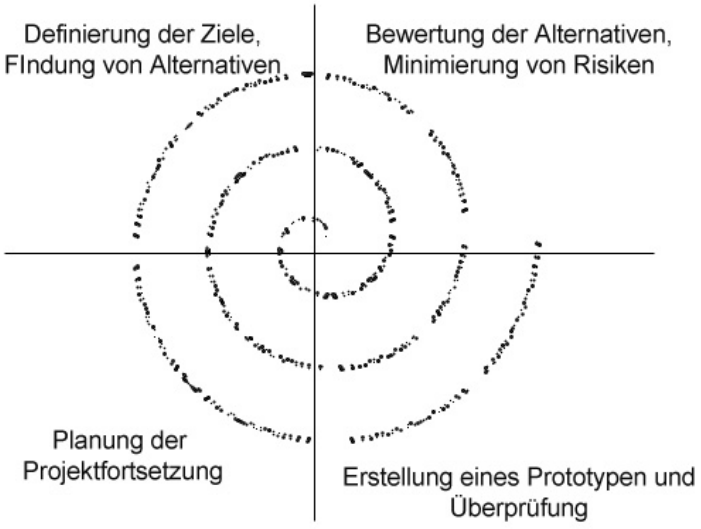


Abbildung 2.2: Spiralmodell nach Boehm [Wir]

In dem ersten Schritt werden jeweils die Anforderung und Ziele sowie die Rahmenbedingungen erarbeitet. Dabei werden Lösungsalternativen entwickelt. Diese werden im nachfolgenden Schritt analysiert und bewertet. Auf dieser Grundlage werden vom Risikomanagement Analysen zur Erkennung, Abschätzung und Reduzierung potenzieller Risiken durchgeführt. Im dritten Schritt findet die eigentliche Entwicklungsarbeit statt. Hierbei wird ein Zwischenprodukt nach dem aktuellen Stand der Entwicklung realisiert und überprüft. Hierzu zählen auch eine Reihe an Tests beziehungsweise Qualitätsprüfungen. Abgeschlossen wird jeder Zyklus mit der Überprüfung der vorangegangenen drei Schritte und der Planung der Projektweiterführung. Das Spiralmodell bietet im Vergleich zu den meisten anderen Vorgehensmodellen durch die Möglichkeit der Anpassung von Anforderungen während der Projektlaufzeit eine größere Flexibilität. Durch regelmäßige Entwicklung von Prototypen kann zudem in Kommunikation mit dem Auftraggeber die Zielerreichung und Änderungsbedarf zeitnah erörtert werden, wodurch Fehlentwicklung und Ressourcenvergeudung geringer gehalten werden kann. Die vollständige Abarbeitung aller Schritte je Zyklus ist bei dem Spiralmodell obligatorisch, jedoch in der Realität nicht immer sinnvoll, beispielsweise wenn nur kleine Änderungen bis zum Erreichen eines Entwicklungsmeilensteins durchzuführen sind. Der hohe Selbstverwaltungsaufwand durch Dokumentation und Sicherstellung eines korrekten Ablaufs verringert außerdem die Eignung für kleinere Projektgruppen. Durch eine kontinuierliche Erweiterung und Konkretisierung der Anforderung ist sowohl die zeitliche, als auch die finanzielle Planbarkeit des IT-Projekts nur schwer vorhersagbar. [eWo] Der entscheidende Vorteil des Spiralmodells ist nach Boehm, dass die Gefahr von Projektfehlschlägen bei Umfangreichen Softwareentwicklungen durch die Risikoorientierung seines Modells erheblich reduziert wird. [Mat14]

2.1.3 V-Modell XT

Das V-Modell XT ist die zweite Revision des bereits 1992 unter dem Namen V-Modell (zwischenzeitig V-Modell 97) entwickelten konkreten Vorgehensmodells zur Definition des Entwicklungsstandards für IT-Systeme staatlicher Einrichtungen. „XT“ steht dabei für „extreme tailoring“ und soll die hohe Anpassbarkeit des Standards an die jeweiligen Projektspezifikationen unterstreichen. Es beschreibt das Vorgehen bei der Zusammenarbeit zwischen Behörden und der Privatwirtschaft bei der gemeinsamen Planung und Entwicklung von IT-Systemen. Seit Anfang 2005 ist die Nutzung des Modells Pflicht für alle, von den Bundesbehörden, durchgeführten IT-Projekte. [HH08] Zur Umsetzung eines Projekts unter Nutzung des dokumentationslastigen V-Modells stehen kostenlose Programme zur Verfü-

gung.

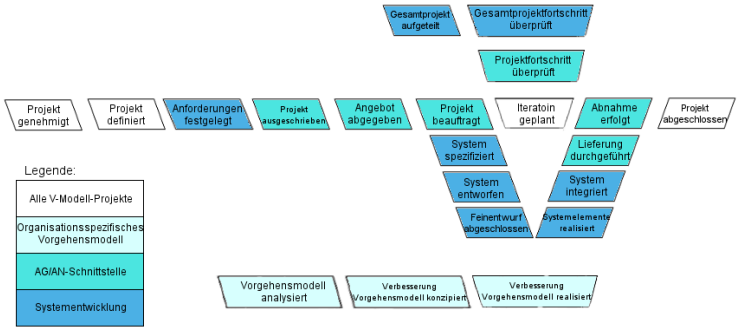


Abbildung 2.3: V-Modell XT [Win]

Das Vorgehensmodell ist ergebnis- und zielorientiert und definiert zu jedem Projektabschnitt, welche Aufgaben von welchen Personen beziehungsweise Personengruppen erledigt werden sollen. Hierzu wird der Projektverlauf, wie in Abbildung 3 zu sehen, in bis zu 21 Tätigkeitsfelder aufgeteilt. Diese werden dem Projektplan je nach Charakter des Vorhabens hinzugefügt. Der zeitliche Ablauf wird in Abbildung 2.3 durch die horizontale von links nach rechts dargestellt. Vier Grundpfeiler bilden hierbei die Basis: Projektmanagement, Qualitätssicherung, Konfigurationsmanagement sowie Problem- und Änderungsmanagement. Das Projektmanagement ist bei dem V-Modell lediglich ein Teil des gesamten Vorgehensmodells. Unter diesem Punkt werden alle Prozesse zur Planung, Steuerung und Kontrolle definiert. So zum Beispiel die Erstellung eines Projekthandbuchs zur Festlegung der organisatorischen Rahmenbedingungen, einen Projektplan der alle Aktivitäten zuzüglich Bearbeiter beschreibt oder das Erarbeiten einer Risikoliste, die alle potenziell auftretenden Risiken enthalten soll, um bereits im Vorfeld Lösungsansätze schaffen zu können. Auch die Ausführung eines umfangreichen Berichtswesens wird hier vorgegeben. Die Qualitätssicherung listet alle Aufgaben zur Planung, Vorbereitung, Durchführung und Dokumentation von Test- und Prüfaktivitäten auf und beinhaltet alle Soll-Parameter, die das Produkt zu Projektende erreichen muss. Das Konfigurationsmanagement protokolliert die im Projekt eingesetzten Mittel wie beispielsweise die Software-Entwicklungs- und Hardwareumgebung im Verlauf des Systemlebenszyklus. So lässt sich zu jedem Zeitpunkt eine Aussage über die Anforderungserfüllung unter Berücksichtigung der jeweiligen Produktkonfiguration treffen. Das Problem- und Änderungsmanagement schließlich dient nach Fertigstellung des Produkts der Steuerung von Änderungs- oder Wartungstätigkeiten. Im Projekthandbuch wird hierzu für jedes Projekt spezifisch definiert, bei welcher Art von Veränderung an dem System das Problem- und Änderungsmanagement eingesetzt werden soll. Alle durch Anfor-

derungserweiterungen, Fehler oder Probleme vollzogenen Modifikationen sollen genau dokumentiert werden. [VMo][HH08] Die Nutzung des V-Modells eignet sich besonders für große und klar strukturierte Projekte mit geringen Anforderungsanpassungen und genauen Zielvorgaben. Es lässt sich für verschiedenen Einsatzfelder anpassen, ist jedoch mit sehr hohem Initialisierungs- und Dokumentationsaufwand verbunden und somit für kleine und mittlere Projekte nur bedingt sinnvoll. [Win]

2.2 Agile Methoden

Die agilen Projektmanagement-Methoden entsprangen seit den neunziger Jahren aus der kritischen Hinterfragung der bis dahin eingesetzten klassischen Methoden, wie dem Wasserfall-Modell. Das agile Manifest beschreibt dabei die Maxime des agilen Ansatzes. Diese besagen [agi]:

- Individuen und Interaktionen haben Vorrang vor Prozessen und Werkzeugen.
- Funktionsfähige Produkte haben Vorrang vor ausgedehnter Dokumentation.
- Zusammenarbeit mit dem Kunden hat Vorrang vor Vertragsverhandlungen.
- Das Eingehen auf Änderungen hat Vorrang vor strikter Planverfolgung.

2.2.1 XP - eXtreme Programming

Extreme Programming (XP) ist einer der bekanntesten Vertreter der agilen Projektmanagement-Methoden. Hauptsächlicher Begründer dieses Softwareentwicklungsprozesses ist Kent Beck, der 1995 erste Veröffentlichungen zu dem Themenfeld machte. Im Laufe der Folgejahre stieg der Bekanntheitsgrad aufgrund der neuartigen Ausrichtung. Im Mittelpunkt der Projektarbeit stehen hierbei nicht mehr eine möglichst plankonforme Abarbeitung statische Entwicklungsphasen wie beispielsweise bei dem Wasserfall-Modell. Auch eine möglichst umfassende Anforderungserhebung vor Beginn der Softwareerstellung wird nicht zwingend vorausgesetzt, wie es bei den klassischen Methoden der Regelfall ist. Viel mehr steht bei XP das Programmieren selbst als schöpferische und kreative Disziplin an erstere Stelle. Die Anwendungsorientierung dieses Ansatzes richtet sich danach, was wirklich gefordert ist und versuch dabei die Komplexität der Implementierung so gering wie möglich zu gestallten. Zusätzlich soll XP dynamisch auf sich ändernde oder zu Projektbeginn nicht vollständig abschätzbare Anforderungen reagieren können, um den neuen Herausforderungen wie verkürzten Produktlebenszyklen am Markt gerecht zu werden. [Zei04] Hierdurch entsteht eine hohe Flexibilität, die die Änderungskosten bei laufenden Projekten moderat halten kann.

Grundsätzlich setzt sich XP aus den Basiskomponenten Werte, Prinzipien und Techniken zusammen. [BA04] Auf den Werten bauen die Prinzipien und Techniken auf. Dabei ist es wichtig, dass keine dieser missachtet wird, um den Erfolg des Projekts nicht zu gefährden, da sie mit einander verknüpft sind. Zu den Werten zählt die Einfachheit, nach der immer die einfachste Lösung gesucht werden soll, um Komplexität zu reduzieren. Außerdem die Kommunikation, die zwischen den Teammitgliedern einen hohen Stellenwert haben soll, da diese Irrtümer und Fragen schnell lösen kann und zudem Teile der Dokumentation weggelassen werden kann. Feedback ist ebenfalls von hoher Relevanz, da durch kontinuierliche Rückmeldung durch den Auftraggeber Fehlentwicklungen und Missverständnisse schnell aufgedeckt und spätere aufwändige Korrekturen vermieden werden können. Courage (zu Deutsch Mut), die zuvor genannten Werte konsequent umzusetzen. [Scr]

Die Prinzipien und Techniken sind dabei nicht neu erfunden, sondern vielmehr eine Sammlung an Good-Practice-Erfahrungswerten aus der Softwareentwicklung. Neu ist hingegen die konsequente – extreme – Umsetzung all dieser in einem Prozessmodell. Zu den Wichtigsten gehören unter anderem:

- *Kurze Release-Zyklen:* schon nach der Entwicklung des Kernsystems beginnende regelmäßige Releases, um in enger Absprache mit dem Auftraggeber weitere Anforderungen, Abweichungen und Wünsche festzustellen
- *Planungsspiel:* sowohl das Entwicklungsteam, als auch der Arbeitgeber planen die Inhalte der nächsten Iteration, bevor diese beginnt. Anforderungen werden durch grobe Anwendungsfälle beschrieben und mit einer Priorität bewertet. Anschließend wird der Arbeitsaufwand geschätzt und bei die Menge akzeptierter Anwendungsfälle an der Kapazität des Teams ausgerichtet
- *Testen:* bei XP werden automatisierte Tests und Test-First-Ansatz vorausgesetzt, die Entwickler testet seine Klassen und der Kunde formuliert Testfälle für die zuvor im Planungsspiel erarbeiteten Anwendungsfälle
- *Pair-Programming:* sowohl Entwurf, Kodierung als auch das Testen wird immer von zwei Programmierern übernommen. Einer programmiert, der andere prüft dessen Arbeit auf Schreib- und Logikfehler und achtet zusätzlich auf vollständige Testabdeckung und Entwurfskonformität. Die Rollen zur Paarzusammensetzung wechseln hierbei regelmäßig, wodurch jeder Entwickler einen besseren Gesamtüberblick erhält. Untersuchungen ergaben, dass durch diese Praktik eine höhere Code-Qualität und erhöhte Mitarbeitermotivation bei nur geringem zusätzlichen Arbeitsaufwand generiert wird
- *On-Site Customer:* neben dem Entwicklerteam ist auch immer ein Ansprechpartner des Auftraggebers vor Ort, um Anforderungen und Fragen ohne

Umwege und Verzögerungen klären zu können
- *Systemmetapher:* soll die Systemarchitektur für Entwickler und Arbeitgeber bildlich vor Augen halten und sie als Orientierungshilfe nutzen [Scr]

Um die Vorteile von XP nutzen zu können, müssen einige Bedingungen erfüllt sein. So sollte wie bereit beschrieben kein Aspekt der drei Basiskomponenten missachtet werden. Zudem muss die Akzeptanz für diese Projektmanagement-Methode sowohl von allen Teammitgliedern und dem Projektleiter auf der einen, als auch von Seiten des Auftragsgebers auf der anderen Seite gegeben sein. Das Entwicklungsteam sollte maximal aus zehn bis 15 Personen bestehen. Der Kunde muss des Weiteren einen Mitarbeiter als Ansprechpartner während der Projektlaufzeit freistellen können.

Kritisch ist das Fehlen einer vollständigen Dokumentation, die bei XP fast ausschließlich in dem Software-Code vorgesehen ist, wodurch das Know-How und wichtiges Wissen nur in den Köpfen der Projektmitglieder vorhanden ist, was bei späteren Änderungswünschen, auftretenden Fehlern oder Wartungsarbeiten Probleme verursachen kann. Das Qualitätsmanagement beläuft sich lediglich auf die im Pair-Programming geschriebenen Tests und muss somit hinterfragt werden. Das Verlassen auf die Systemmetapher ist ebenfalls kaum ein ausreichender Ersatz für eine Dokumentation der Systemarchitektur und kann leicht zu Missverständnissen und Verwirrungen führen. Trotz dieser Schwächen gibt es viele positive Erfahrungsberichte von erfolgreichen Projekten unter dem Einsatz von XP. [Rei]

2.2.2 Kanban

Kanban (japanisch für „Signalkarte“) wurde in den fünfziger Jahren von dem japanischen Fahrzeughersteller Toyota zur Steuerung der Produktionsprozesse entwickelt. Das damals als Planungssystem eingesetzte Verfahren sollte alle Teile der Fertigung durch die Vermeidung von Engpässen die Durchlaufzeiten optimieren. Im Jahr 2007 wurde diese Methode von David Anderson für den Einsatz in Projekten, insbesondere für die Softwareentwicklung, transformiert. Seine Intention war, in diesem Kontext ebenfalls die Effizienz und Bearbeitungsgeschwindigkeit zu erhöhen. [Pro]

Im Mittelpunkt dieses Ansatzes steht die Erzielung eines durchgängig hohen Arbeitsflusses, unterstützt durch eine Visualisierung. Dazu soll den Projektmitgliedern ermöglicht werden, während der Entwicklung Veränderungen und Verbesserungen einfließen zu lassen, um den Fluss der Projektbearbeitung zu optimieren.

Die Visualisierung wird mittels einem Board generiert, bei dem mehrere Spalten jeweils einen Bearbeitungsstand repräsentieren. Karten stellen einzelne Aufgaben

dar und werden im Verlauf der Bearbeitung von der ersten Spalte „Unerledigt“ bei Beginn zu „In Bearbeitung“ und nach Abschluss aller weiteren Schritte in die Spalte „Erledigt“ verschoben. Auf diese Weise wird auf einen Blick sichtbar, wie der Fortschritt aller auf dem Board befindlichen Arbeitsaufträge ist. Je nach Bedarf lassen sich weitere Spalten wie beispielsweise „Testen“ hinzufügen. Um das Entwicklerteam nicht zu überlasten, einen konstanten Arbeitsfluss und die Übersichtlichkeit zu wahren, wird eine maximale Anzahl, zeitgleich in Bearbeitung befindlicher Aufgaben, festgesetzt. Durch die Ausspaltung komplexer Prozesse in kleinere Aufgabenbereiche wird zudem der agile Aspekt gefördert, da durch kleinere Veränderungen die Gefahr durch Fehler und Probleme reduziert und eine Korrektur vereinfacht wird. Die unerledigten Aufgaben werden von den Mitarbeitern geholt (Pull-Prinzip), statt von Vorgesetzten delegiert, was die Arbeitsqualität, Motivation und Zufriedenheit der Projektgruppe erhöht und den Leistungsdruck verringert. [WSHS]

Da sich Kanban auf die Aufgabenverteilung und Bearbeitung konzentriert, können weitere Projektentscheidungen wie Rollenverteilung, Dokumentationsaufwand oder Prototypenentwicklung individuell und projektspezifisch von der Arbeitsgruppe beziehungsweise dem Projektleiter treffen.

Die Einführung und Umsetzung von Kanban kann in sieben Schritten beschrieben werden: [Leo15]

1. Aufgaben definieren: Zu Beginn muss auf Grundlage der Anforderungen und Realisierbarkeit ermittelt werden, welche Aufgaben zu erledigen sind.

2. Visualisieren: Anschließend werden alle Aufgaben auf das Kanban-Board übertragen. Hierbei werden die Aufgaben im Detail durch Eigenschaften wie notwendige Informationsflüsse, Aufgabenfeld und Aufwand beschrieben

3. WiP-Beschränkung: Die Anzahl maximal gleichzeitig in der Spalte „Bearbeitung“ befindlichen Aufgaben (engl. Work in Progress, kurz WiP) ist Basis für die Analyse von Engpässen, da durch die Betrachtung des Boards auf diese Weise leicht zu erkennen ist, an welcher Stelle der Fluss gebremst wird.

4. Serviceklassen: Neben der allgemeinen Wertzuweisung einzelner Aufgaben im ersten Schritt werden die Aufgaben zusätzlich separaten Serviceklassen zugeordnet. So kann auf einfache Weise sichergestellt werden, dass Aufgaben aller Aufgabenbereiche die entsprechende Aufmerksamkeit durch das Entwicklerteam bekommen. Bei der Definition von Serviceklassen sollte im Idealfall auch der Auftraggeber Einfluss nehmen, um seine Wünsche und Anforderungen möglichst klar darstellen zu können.

5. Regeln: Die im vorherigen Schritt definierten Serviceklassen werden nun mit Regeln und Kapazitäten ausgestattet. So wird vorab bestimmt, wie viele Aufgaben je Serviceklasse zur gleichen Zeit in umgesetzt werden dürfen und welche Aktionen bei Bearbeitungsbeginn Standardmäßig durchgeführt werden. Eine Regel ist beispielsweise die sofortige Behebung von Fehlern. Alle Regeln sollten dabei stets eingehalten werden und auf ihre Notwendigkeit hin untersucht werden.

6. Leistungsanalysen: Vor und nach Veränderungen am System soll die Zielerreichung gemessen werden, um die Veränderung bewerten zu können. Dabei ist eine ungefähre Aussage als ausreichend zu betrachten, da lediglich die Tendenz von Interesse ist. Wichtige Größen sind unter anderem die durchschnittliche Durchlaufzeit von Aufgaben oder die Dauer der Wartezeiten in der ersten Spalte das Kanban-Boards.

7. Kanban ausführen: Um das Projekt zu starten, muss das Board mit Aufgaben gefüllt werden. Die Input-Koordination definiert den Aufgabenerstellungsprozess unter Berücksichtigung der Interessen des Auftraggebers und der Empfehlungen des Entwicklungsteams. Bei der Auswahl der nächsten Tätigkeiten ist die Angemessenheit und Zügelführung ausschlaggebend. Die Output-Koordination plant zu welchem Zeitpunkt ein Produktrelease an den Kunden ausgeliefert werden soll, zum Beispiel nach Erreichen bestimmter Meilensteine.

   Das Daily Standup, einem täglichen Treffen der Projektmitarbeiter, dient zur Lösung von Problemen und Fragen und kann durch regelmäßige, beispielsweise wöchentliche, Team-Retrospektiven ergänzt werden. Diese reflektieren die wichtigsten Ereignisse seit der vorherigen Retrospektive, um Veränderungs- bzw. Optimierungsbedarfe zu erkennen.

Kanban als Projektmanagementmethode legt den Fokus auf den Arbeitsfluss des Entwicklungsteams. Hierbei bietet die Visualisierung der einzelnen Aufgaben eine schnelle und einfache Übersicht. Engpässe, Probleme und die Überlastung einzelner Mitarbeiter lassen sich auf diese Weise gut identifizieren und optimieren. Ab einer bestimmten Projektgröße und entsprechend vielen parallel ausgeführten Aufgaben kann diese Übersicht jedoch verloren gehen. Das Einsatzpotenzial ist somit auf weniger komplexe, kleine bis mittlere Projekte beschränkt.

2.2.3 SCRUM

Die Projektmanagement-Methode SCRUM ist, wie das zuvor vorgestellte Kanban, inspiriert durch das Konzept des „Lean Managements“, also des schlanken

Managements, dessen Entwicklung maßgeblich durch Toyotas „Lean Production“ vorangetrieben wurde. Der Grundgedanke beschreibt die Konzentration auf eine schlanke oder auch agile Organisation statt auf Produktionsprozesse. [Wik15]

In der Softwareentwicklung zählt SCRUM somit zu den agilen Projektmanagement-Methoden, unterscheidet sich jedoch von Prozessmodellen wie ‚extreme Programming‘, da der Projektablauf und –aufbau, nicht die Arbeitsweise des Entwicklungsteams im Mittelpunkt steht. Es definiert den Organisatorischen Rahmen, die Wahl der Entwicklungsart wird der jeweiligen Projektgruppe überlassen. Entwickelt wurde die Methode Anfang der neunziger Jahre von Jeff Sutherland und Ken Schwaber. Seitdem ist die Beliebtheit und Verbreitung der Methode bis heute stark gestiegen. [Eck10][Glo10]

SCRUM teilt die Projektbeteiligten in die Rollen Product Owner, Team und Scrum-Master auf. Ein direkter Projektmanager wie in den meisten Ansätzen ist nicht vorgesehen. Die einzelnen Entwicklungszyklen im Projektverlauf werden Sprints genannt und erstrecken sich über einen Zeitraum von wenigen Wochen. [Eck10] Abbildung 4 illustriert den Aufbau und Ablauf der Management-Methode.

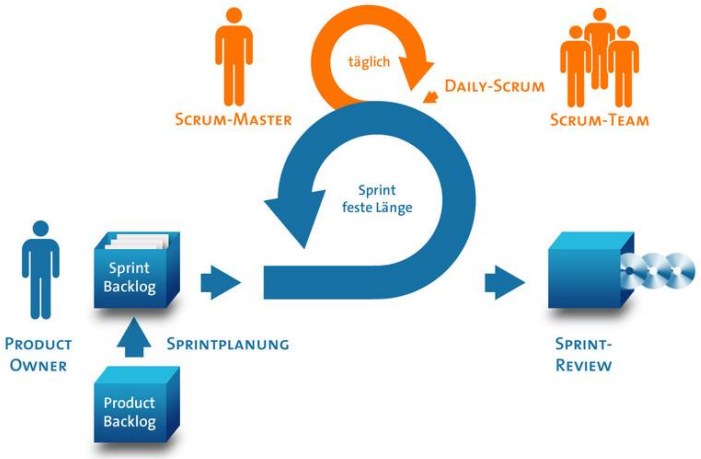


Abbildung 2.4: SCRUM-Modell [3m5]

Der Product Owner (sinngemäß übersetzt „Produktverantwortlicher“) vertritt die Interessen des Auftraggebers oder Anwenders des zu erarbeitenden Produkts. Er formuliert alle Anforderungen und Tätigkeiten zur Entwicklung der Software und bündelt sie in dem Product Backlog. Dieser Prozess begleitet den gesamten Projektverlauf und wird fortwährend aktualisiert. Zudem plant er die Termine für die Auslieferung von Prototypen an den Kunden auf Grundlage der Aufwandsschätzungen des Projektteams. Der Product Owner ist zudem das Bindeglied zwischen den Entwicklern und dem Auftraggeber. Er ist Ansprechpartner bei Fragen

und Problemen auf der einen und „Informant“ für den Kunden bei allen Veränderungen des Produkts auf der anderen Seite. Er vermittelt auf diese Weise auch Rückmeldungen des Kunden bei neu implementierten Softwarefeatures.

Das Team übernimmt die Entwicklung des Produkts. Wie bereits beschrieben ist dabei das Vorgehen nicht vorgegeben. Es sollte im Normalfall aus etwa fünf bis zehn Mitgliedern bestehen und ohne eine Leitung auskommen. Stattdessen soll es sich selbst organisieren, d.h. eigenständig und zielgerichtet Aufgaben aus dem Product Backlog entnehmen und je nach Kompetenzen auf die Mitglieder aufteilen. Ein Sprint wird durch die Erreichung bestimmter Etappenziele und Meilensteine definiert. Alle hierzu notwendigen Aufgaben werden ein einem Sprint Planning-Meeting aus dem Product Backlog ausgewählt in im Sprint Backlog gesammelt. Anschließend wird der mehrwöchige Sprint gestartet. Bei dem Daily-Scrum-Meeting berichten die Teammitglieder, was sie seit dem letzten Daily-Scrum erreicht haben, welche Probleme und Hindernisse sie dabei hatten und was sie bis zum nächsten Meeting umsetzen wollen. Dieser Vorgang sollte dabei möglichst knapp ausfallen, damit die Arbeitsgeschwindigkeit nicht negativ beeinflusst wird. Wie auch bei Kanban wird ein Board zur Visualisierung und Verwaltung der Sprint-Aufgaben (Tasks genannt) eingesetzt. Die einzelnen Tasks werden dabei je nach Bearbeitungsstatus zwischen den einzelnen Spalten verschoben wobei in der ersten Spalte alle zu erledigenden Aufgaben gesammelt werden. Während des Sprints sind keine Anforderungsänderungen zulässig, um den Arbeitsprozess nicht zu stören.

Während der Bearbeitung der Sprints versucht der ScrumMaster, die Einhaltung der Scrum-Prozesse sicherzustellen und bei Abweichungen einzuschreiten. Je weniger er eingreifen muss, desto besser ist die Vorgehensweise durch das Team verinnerlicht worden. Zusätzlich soll er Product Owner und das Team bei der Umsetzung von SCRUM unterstützen, Prozessbezogene Fragen beantworten und den Arbeitsfluss des Teams stabil halten. So soll er zwar auch den Product Owner bei der Nutzung von Scrum helfen, jedoch gleichzeitig dafür sorgen, dass sich dieser nicht zum Projektleiter ernennt.

Nach Abschluss eines Sprints wird die, vom ScrumMaster moderierte Sprint-Review durchgeführt. Der Product Owner analysiert die erreichten Ergebnisse, wobei nur vollständig und korrekt erfüllte Aufgaben als erledigt angesehen werden. In einer Live-Demo soll diesem der aktuelle Entwicklungsstand und die erreichten Ziele gezeigt werden.

Im Anschluss an die Sprint-Review wird die Sprint-Retrospektive durchgeführt. Sie dient der Reflektion des abgeschlossenen Sprints , um Verbesserungen im Arbeitsablauf erarbeiten zu können. Diese sollen in die Arbeitspraxis des nächsten Sprints einfließen. [Eck10]

2.3 Exkurs: CRISP DM

CRISP-DM (Abkürzung für CRoss-Industry Standard Process for Data Mining) ist ein Prozessmodell für die Abwicklung von Data Mining (DM) Projekten. Es wurde 1996 unter anderem von DaimlerChrysler innerhalb eines EU-Förderprojekts entwickelt. Es teilt den DM-Prozess in mehrere Phasen auf, die vielfach zu unregelmäßigen Zeiten angewendet werden müssen wie in Abbildung 2.5 zu sehen:

Abbildung 2.5: CRISP-DM-Modell [Rieb]

1. Geschäftsverständnis: Definition der Anforderungen und Ziel und hieraus resultierender Aufgaben

2. Datenverständnis: erste Begutachtung und Potenzialanalyse der Verwendbaren Daten feststellen, Hindernisse durch unzureichende Datenqualität abschätzen

3. Datenvorbereitung: Aufbereiten und Transformieren der Datensätze um eine Modellierung durchführen zu können

4. Modellierung: Anwendung von Data Mining-Analysen und -Modellen, Optimierungen an den Parametern vornehmen

5. Evaluation: Ergebnisse Bewerten und zielorientierte Auswahl des optimalen Modells

6. Bereitstellung: dem Kunden die Ergebnisse in geeigneter Form, beispielsweise über ein Dashboard, zur Verfügung stellen

Eine Umfrage der Website „KDnuggets“ aus dem Jahr 2007 über die Nutzung von Data Minig Methoden ergab, dass 42% der Befragten CRISP-DM einsetzen, während nur 19% die zweithäufigste Methode einsetzen. [Pol] Nach dieser Quelle ist der Standard damit das beliebteste Prozessmodell.

3 Einsatzentscheidung der Projektgruppe

Innerhalb der Projektgruppe DASH (Data Analytics with Hadoop) musste ebenfalls über den Einsatz einer geeigneten Projektmanagement-Methode zur Unterstützung der Softwareentwicklungsprozesse entschieden werden. Es wurde sich auf eine individualisierte Projektmanagement-Methode geeinigt, die sich an den Prinzipien von SCRUM orientiert. Zusätzlich werden Elemente von CRISP-DM in den Projektablauf integriert. Der agile Ansatz eignet sich für das Projekt, da sich zu Beginn noch nicht alle Anforderung bis ins Detail definieren lassen, unter anderem weil der Praxispartner nur sehr grobe Wünsche und Ziele genannt hat. Das Potenzial durch Data Mining kann aufgrund fehlender Fachkompetenzen erst im Laufe des Projekts vollständig erkannt werden. Ein weiterer weicher Entscheidungsfaktor ist, dass ein Großteil der Gruppe bereits positive Erfahrungen mit dem Einsatz von SCRUM gemacht hat und die Vorgehensweisen so schon bekannt waren. Die Akzeptanz der management-Methode ist ein wichtiger oftmals unterschätzter Aspekt bei der Auswahl.
So werden Sprints nicht durch Daily- sondern Weekly-Scrum-Meetings unterstützt, da die Projektgruppe (PG) nur in Teilzeit von den Mitgliedern wahrgenommen werden kann. Zudem werden, anders als bei SCRUM, zwei Projektmanager die Abläufe koordinieren und zugleich in Verbindung mit dem Ansprechpartner des Praxispartners stehen. Dieser ist der Vertreter des Auftraggebers des Projekts. Die universitären Betreuer sind ebenfalls in einer Art Auftragsgeberrolle, da sie die erbrachte Leistung in Form einer Benotung „finanzieren“. Die PG muss also „zwei Herren dienen“ und zugleich die Anforderungen möglichst zielgerichtet umsetzen.

Die Sprint-Laufzeit wird den aktuellen Bedürfnissen des Projekts angepasst, soll aber im späteren Verlauf auf zwei Wochen festgesetzt werden. Die Steuerung und Kontrolle der Aufgabenbearbeitung wird wie in SCRUM mittels Taskboard durchgeführt. Unterstützt wird dies durch die kollaborativen Web-Tools Confluence und JIRA. Wie auch bei SCRUM soll es zu definierten Zeitpunkten ein Alpha- beziehungsweise Beta-Release geben, bei denen der Aufraggeber durch eine Live-Demonstration der Entwicklungsfortschritte informiert werden soll, um Feedback

über den Grad der Zielerreichung zu diesem Zeitpunkt, Änderungswünsche und Kritik zu erlangen.

4 Fazit

Da die meisten Projektmanagement-Methoden für Softwareentwicklungen auf Vollzeitprojektarbeit ausgerichtet sind und die Bedingungen somit nicht der universitären Projektbearbeitung entsprechen, ist eine Individualisierung des Projektmanagement innerhalb der PG naheliegend. Die zusätzliche Integrierung des CRISP – Prozessmodells ist durch den Themenschwerpunkt Data Mining begründet. In dieser Konfiguration hat bis jetzt kein Teammitglied Erfahrungswerte sammeln können, sodass der Projekterfolg nicht garantiert werden kann. Jedoch kann abschließend gesagt werden, dass die die Kombination von SCRUM und CRISP-DM ein hohes Potenzial bietet und eine robuste Orientierungshilfe für die Projektdurchführung bietet.

Literatur

[3m5] *3m5. - Scrum*. https://www.3m5.de/scrum/

[agi] *Agiles Manifest*. http://scrum-master.de/Scrum-Glossar/Agiles_Manifest

[BA04] BECK, Kent ; ANDRES, Cynthia: *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 2004. – ISBN 978–0–13–405199–4

[Bau13] BAUER, Roman: *Hard- und Software im Entwicklungsprozeß integrierter Produkte: Die Anwendung des Just-in-Time-Konzeptes in Forschung & Entwicklung*. Springer-Verlag, 2013. – ISBN 978–3–663–08539–3

[DME05] DEEK, Fadi P. ; MCHUGH, James A. M. ; ELJABIRI, Osama M.: *Strategic Software Engineering: An Interdisciplinary Approach*. CRC Press, 2005. – ISBN 978–1–4200–3111–9

[Eck] ECKHART HANSER, EMEL TAN: *Klassisch vs. Agil*. http://www.codesco.com/fileadmin/bilder_grafiken/Sonstiges/KlassischVSagil_Ausarbeitung.pdf

[Eck10] ECKHART HANSER: *Agile Prozesse: Von XP über Scrum bis MAP*. Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2010 (eXamen.press 0). – ISBN 978–3–642–12313–9

[eWo] *Usability Engineering Hypermedialer Benutzungsoberflächen*. http://www.e-work.ethz.ch/praesentationen/ss_04/gruppe_3/Homepage%20Telepraktikum%20SS04%20Gruppe%203/index.php?left=k2&body=k2_spiral&head=h2_spiral

[FST14] FUCHS, Dipl-Wirtsch-Inf A. ; STOLZE, Prof Dr C. ; THOMAS, Prof Dr O.: Von der klassischen zur agilen Softwareentwicklung Evolution der Methoden am Beispiel eines Anwendungssystems. In: *HMD Praxis der Wirtschaftsinformatik* 50 (2014), Januar, Nr. 2, 17–26. http://dx.doi.org/10.1007/BF03340792. – DOI 10.1007/BF03340792. – ISSN 1436–3011, 2198–2775

[Glo10] GLOGER, Boris: Scrum. In: *Informatik-Spektrum* 33 (2010), März, Nr. 2, 195–200. http://dx.doi.org/10.1007/s00287-010-0426-6. – DOI 10.1007/s00287–010–0426–6. – ISSN 0170–6012, 1432–122X

[Gre10] GRECHENIG, Thomas: *Softwaretechnik: Mit Fallbeispielen aus realen Entwicklungsprojekten*. Pearson Deutschland GmbH, 2010. – ISBN 978–3–86894–007–7

[HH08] HÖHN, Reinhard ; HÖPPNER, Stephan: *Das V-Modell XT: Grundlagen, Methodik und Anwendungen*. Springer-Verlag, 2008. – ISBN 978–3–540–30250–6

[INf15] *Anwendungsentwicklung - Vorgehensmodell - Spiral-Modell*. http://www.infforum.de/themen/anwendungsentwicklung/se-spiral-modell.htm. Version: November 2015

[Leo15] LEOPOLD, Klaus: *Kanban richtig einführen*. http://www.heise.de/developer/artikel/Kanban-richtig-einfuehren-1344554.html?artikelseite=2. Version: Dezember 2015

[Mat14] MATTERN, Matz: *Critical Chain Project Management: Ein Ansatz zur Durchführung von Projekten in der Softwareentwicklung*. disserta Verlag, 2014. – ISBN 978–3–95425–786–7

[Pol] *Poll: Data Mining Methodology*. http://kdnuggets.com/polls/2007/data_mining_methodology.htm

[Pro] *Kanban — Projektmanagement: Definitionen, Einführungen und Vorlagen*. http://projektmanagement-definitionen.de/glossar/kanban/

[Rei] REISSLING, Ralf: *Extremes Programmieren (XP) - GI - Gesellschaft für Informatik e.V.* https://www.gi.de/index.php?id=647&tx_ttnews%5Btt_news%5D=40&cHash=48ac03061448868836b77b9474023d4d

[Riea] RIEHLE, Jan-Philip: *Projektmanagement: Wasserfall-Modell vs. agiles Vorgehen*. https://www.pinuts.de/blog/webstrategie/projektmanagement-wasserfall-gegen-scrum

[Rieb] RIEPL, Wolf: *CRISP-DM: Ein Standard-Prozess-Modell für Data Mining*. http://statistik-dresden.de/archives/1128

[Scr] *Extreme Programming (XP) – Scrum Kompakt*. http://www.scrum-kompakt.de/grundlagen-des-projektmanagements/extreme-programming-xp/

[The04] THEUERKORN, Fenix: *Lightweight Enterprise Architectures*. CRC Press, 2004. – ISBN 978–0–203–50531–1

[VMo] *Grundlagen des V-Modells*. http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt/Releases/1.4/V-Modell-XT-Gesamt.pdf

[Wik15] *Lean Management*. https://de.wikipedia.org/w/index.php?title=Lean_Management&oldid=143183397. Version: Juni 2015. – Page Version ID: 143183397

[Win] *Überblick Vorgehensmodelle im Projektmanagement – Winfwiki*. http://winfwiki.wi-fom.de/index.php/%C3%9Cberblick_Vorgehensmodelle_im_Projektmanagement

[Wir] *Spiralmodell | Software Technik*. http://www.wirtschaftsinformatik-24.de/software-engineering/spiralmodell.php

[WSHS] WILLKOMMER, Josef ; STORZ, Sacha ; HALLER, Dominik ; SCHUBART, Michael: *Agiles Projektmanagement – Projektentwicklung mit Scrum, Kanban & Co*. https://www.techdivision.com/_Resources/Persistent/a90c984a454ba0b8478694b83f7a8822514b8fc8/Agiles-PM-Whitepaper0502.pdf

[Zei04] ZEIGER, Björn: *eXtreme Programming - Konzepte, Ziele und Methoden*. GRIN Verlag, 2004. – ISBN 978–3–638–25266–9

Abschließende Erklärung

Ich versichere hiermit, dass ich meine Seminararbeit „Klassische vs. Agile Softwareentwicklung: Einsatz von SCRUM in der Projektgruppe“ selbständig und ohne fremde Hilfe angefertigt habe, und dass ich alle von anderen Autoren wörtlich übernommenen Stellen wie auch die sich an die Gedankengänge anderer Autoren eng anlegenden Ausführungen meiner Arbeit besonders gekennzeichnet und die Quellen zitiert habe.

Oldenburg, den 26. September 2016

Design Thinking / Design Science

Seminararbeit

Themenersteller:	Prof. Dr.-Ing. habil. Jorge Marx Gómez
Betreuer:	M. Eng. & Tech. Viktor Dmitriyev Dr.-Ing. Andreas Solsbach
Vorgelegt von:	Hauke Precht hauke.precht@uni-oldenburg.de
Abgabetermin:	2. Dezember 2015

Inhaltsverzeichnis

Abkürzungen

DT Design Thinking

DS Design Science

Abbildungen

1 Einleitung

Die Globalisierung, der technischen Fortschritt und die damit einhergehende Vernetzung erzeugt eine nie dagewesene Komplexität in unserer Zeit. Dies stellt insbesondere die Geschäftswelt vor großen Herausforderungen. Um dieser Komplexität angemessen zu begegnen, entwickelt sich seit einigen Jahren das Thema Design Thinking. Durch die stetige Entwicklung und neuen Anforderungen, wachsen auch zunehmend Informationssysteme. Um diese komplexen Informationssysteme entwickeln und analysieren zu können, wird zunehmend das Feld des Design Science hinzugezogen.

Im Rahmen dieser Seminarbeit, sollen nun die beiden Themen, Design Thinking sowie Design Science, vorgestellt und beschrieben werden. Dazu wird zunächst der Begriff „Design“ definiert. In Kapitel 3 wird ein Überblick über Design Thinking gegeben. Dabei wird der Begriff definiert. Weiter werden Voraussetzungen aus organisatorischer und personeller Sicht identifiziert. Schließlich werden verschiedenen Phasen (Modelle) vorgestellt, nach denen Design Thinking betrieben werden kann.

In Kapitel 4 wird zunächst der Begriff „Design Science“ zusammen mit Design Science Research definiert. Weiter werden artverwandte Begriffe abgegrenzt, um das Verständnis von Design Science zu verbessern. Im Anschluss daran wird das Design Science Reserach Framework nach Hevner et. al. vorgestellt sowie Kritik an diesem Framework behandelt. Das letzte Kapitel schließt diese Arbeit im Rahmen einer Zusammenfassung der gerade genannten Themen ab.

2 Design

In diesem Kapitel wird der Begriff Design als Einleitung in die Themengebiete des Design Thinking und Design Science definiert.

Die Herkunft des Wortes Design lässt sich auf das lateinische Wort *designare* zurückführen, was als erläutern, beschreiben oder markieren übersetzt werden kann. Im Verlauf der geschichtliche Entwicklung änderte sich die allgemeine Bedeutung hin zu einer „professionellen Praxis und des spezifischen Denkens"([EM08] S. 88).

Zunächst ist zu sagen, dass es keine eindeutige und allgemeine Definition für den Begriff Design gibt. Vielmehr gibt es in den unterschiedlichsten Bereichen von Wissenschaft, Technik und Wirtschaft, verschiedene Ausprägungen und Definitionen von Design. Zudem ist in ([EM08] S. 87) hervorgehoben, dass schon das historische und damit kulturelle Verständnis von Design auseinander geht. So wird in der deutschen Sprache beschrieben, dass sich das Verständnis von Design vorrangig auf *„die Umsetzbarkeit gestalterischer Ideen im Rahmen industrieller Produktion oder von Zeichen-Systemen (und neuerdings auch von Dienstleistung, Forschung und dergleichen)*" ([EM08] S. 87) bezieht. Der Ansatz innerhalb der englischen Sprache bezieht den Begriff *Design* dagegen auf den mentalen Plan bzw. auf die Konstruktion eines Objektes. Weiter bezeichnet Design hier alles, was gestaltbar ist ([EM08] S. 87). Somit ist dieser Ansatz weitaus pragmatischer als der Ansatz innerhalb der deutschen Sprache.

Während die ersten Ansätze und Beschreibungen von Design sich auf eine gestalterische Ebene beziehen, ist zu beobachten, dass Design zunehmend als Prozess verstanden wird, der in unterschiedlichen Kontexten mit unterschiedlichen Bedingungen und Komplexitäten eingesetzt wird. Damit wird deutlich, dass Design interdisziplinär ist.

So wurden auch in ([GH10] S. 97-103 verschiedene Sichten des Design identifiziert. Da eine vollständige Auflistung aller dort identifizierten Sichten den Rahmen dieser Arbeit überschreiten würde, wird nur auf die für diese Arbeit, aus Sicht des Autors, passendste eingegangen. Diese ist nach ([GH10] S. 98) „Design as problem solving", also Design zur Problemlösung. Dies deckt sich mit der zuvor erwähnten Entwicklung des Begriffs. Dabei wird beschrieben, das Design lösungsorientiert vorgeht, um Probleme zu bearbeiten. Probleme, die mit Hilfe von Design gelöst werden können bzw. gelöst werden sollen, können in zwei Kategorien ge-

teilt werden, „Wissens-Problem“ (Knowledge Problem) und „Domänen-Problem“ (Field Problem).

In ([GM13] S. 6) haben Gürtler und Meyer diesen Sachverhalt zusammengefasst, in dem Sie schreiben, dass Design nicht nur ein Prozess ist um Dinge schöner zu machen, sondern um kreative Lösungen für komplexe Probleme zu finden.

3 Design Thinking

In diesem Kapitel wird der Begriff *Design Thinking* näher betrachtet. Abschnitt 3.1 beinhaltet die für die vorliegende Arbeit wichtige Definition des Begriffs. In den darauf folgenden Abschnitten werden Voraussetzungen für erfolgreiches Design Thinking sowie Phasen und Modelle des Design Thinkings vorgestellt.

3.1 Definition - Design Thinking

Durch die Definition des Begriffs *Design* in Kapitel 2 wurde bereits dargestellt, dass Design als Möglichkeit betrachtet werden kann, kreative Lösungen zu komplexen Problemen zu finden. Davon ausgehend, definieren Gürtler und Meyer Design Thinking als *„[...]Arbeitsmethode, die verschiedene Werkzeuge verbindet, um Innovation und Ideenfindung zu unterstützen“* ([GM13] S. 7). Tim Brown, Chief executive officer (CEO) des Unternehmens IDEO, definiert Design Thinking als „[...] human-centered approach to innovation that draws from the designer's toolkit to integrate the needs of people, the possibilities of technology, and the requirements for business success“ ([IDE15]). Nach Brown handelt es sich bei Design Thinking also um einen Ansatz, bei dem der Mensch im Mittelpunkt steht und Innovationen mithilfe von verschiedenen Designwerkzeugen und technologischen Möglichkeiten entwickelt werden, die den Bedürfnissen des Menschen gerecht werden.

Besonders der Fokus um den Nutzer herum wird auch von Denning hervorgehoben ([Den13] S. 31). Aus den dargestellten Definitionen, die bereits nahezu homogen erscheinen, lässt sich nun die folgenden Definition ableiten: *Design Thinking wird als Methodik verstanden, die auf den Nutzer fokussiert, mit Hilfe von Tools und Technik gestellte Probleme zu lösen versucht sowie Innovationen vorantreibt und entwickelt.* Dazu muss ergänzt werden, dass Design Thinking als Methodik im Team durchgeführt wird.

3.2 Voraussetzungen für erfolgreiches Design Thinking

Dieser Abschnitt stellt verschiedene Voraussetzungen vor, die benötigt werden um Design Thinking erfolgreich zu betreiben. Die hier identifizierten Faktoren, beziehen sich auf die Organisation und Struktur, in der Design Thinking betrieben

werden soll. Neben der Organisationssicht beinhaltet Abschnitt 3.3 Charakterzüge eines typischen Design Thinkers. Für Voraussetzungen aus Organisationssicht hat Dennings ([Den13] S. 31) drei Prinzipien erläutert, wie sie auch von dem Unternehmen IDEO gelebt werden, die in folgender Auflistung dargestellt sind:

1. Das *Viele-Augen-Prinzip (Many Eyes)* wird als erstes Prinzip genannt. Ziel ist es, Teammitglieder aus möglichst vielen unterschiedlichen Bereichen zusammenzubringen. Durch die vielen unterschiedlichen Kenntnissen und Qualifikationen der Teammitglieder kann ein Problem von vielen Seiten betrachtet und bearbeitet werden. Somit ist eine umfangreiche Analyse möglich.

2. Die *Nutzersicht (Customer view point)* wurde bereits in der Definition stark hervorgehoben. Somit ist es auch unter den drei Prinzipien zu finden. Hier ist zu sagen, dass das Desing Thinking Team den Nutzer vor Ort besuchen muss, Interviews führt und verschiedene Szenarien (z. B. Stresssituationen) beobachtet / erfragt. Alle Schritte müssen sorgfältig dokumentiert werden, um die Ergebnisse in das weitere Verfahren integrieren zu können und dem Nutzer ein maßgeschneidertes Produkt bieten zu können.

3. Als drittes Prinzip wurde die *Greifbarkeit (Tangibility)* angeführt. Dahinter verbirgt sich eine Kultur des schnellen Prototypings und des „Fast Failures“. Ziel ist es, möglichst schnell Prototypen zu erstellen, die ausprobiert und evaluiert werden können. Hier kann und sollte der Nutzer miteinbezogen werden.

Durch das sehr frühe Erstellen von Prototypen ist das Desing Thinking Team in der Lage, Dinge auszuprobieren, zu verbessern oder zu verwerfen. Es handelt sich somit um einen iterativen Prozess, in dem auch der Nutzer involviert ist. Am Ende dieses Prozesses können Aussagen getroffen werden, welche Funktionen den Nutzer unterstützt haben und wo noch Potentiale identifiziert worden sind. Dies spielt wieder in das erstgenannte Prinzip „Nutzersicht“ hinein.

Ergänzend zu den drei vorgestellten Prinzipien, haben Gürtler und Meyer in ihrem Buch weitere Grundwerte identifiziert. Diese werden in den Abschnitten 3.2.1 bis 3.2.3 näher betrachtet. Dabei wird gezeigt, dass die hier identifizierten Grundwerte zu den zuvor aufgezeigten Prinzipien passen. Zu erwähnen ist, das sich die beschriebenen Werte auf die Organisation und Struktur beziehen, und nicht direkt auf den Design Thinker als Person. Es gilt also gewissen Rahmenbedingungen und Räume zu bilden, um Design Thinking zu unterstützen.

3.2.1 Das Team

Im vorangestellten Abschnitt wurde bereits das *Viele-Augen-Prinzip* erläutert. Dies wird auch von Gürtler und Meyer aufgegriffen. Sie beschreiben das Prinzip als „die passenden Menschen“. Das Ziel dieses Prinzips ist, eine Gruppe von Menschen zusammen zu bringen, die sich als Team gegenseitig fördern, fordern und inspirieren, um ein gestelltes Problem zu bearbeiten. Ebenfalls wird empfohlen, nach Möglichkeit, Menschen aus verschiedenen Gebieten zusammenzubringen, um das interdisziplinäre Arbeiten zu ermöglichen und zu fördern. (Vgl. [GM13] S.18)

3.2.2 Die Räumlichkeiten

Durch das Bereitstellen von Räumlichkeiten, die nach Möglichkeit flexibel gestaltet werden können, wird der Prozess des Design Thinkings unterstützt. Empfohlen wird ein sogenannter Team Space, in dem das Design Thinking Team zusammen diskutieren und arbeiten kann. Es ist explizit darauf hinzuweisen, dass dieser Team Space nicht die Ablösung der „traditionellen Büros“ bedeutet. Diese bleiben auch im Design Thinking ein fester Bestandteil und dienen als Rückzugsraum für die einzelnen Mitglieder, die dort die Möglichkeit haben in einer ruhigen Umgebung weiterzuarbeiten. Auch herkömmliche Besprechungsräume, beispielsweise für Kundenpräsentationen, sollten erhalten bleiben. (Vgl. [GM13] S. 21) In ([GM13] S. 22f) gegebenen Anregungen konkretisieren die abstrakte Beschreibung eines flexiblen Raums als Team Space.

Durch das Zusammenwirken der Teammitglieder mit ihren jeweiligen verschiedenen Hintergründen und des Prinzips der Greifbarkeit werden oft viele neue Ideen und Konzepte generiert. Damit keine dieser Ideen verloren geht, wird empfohlen, im Team Space stets eine ausreichende Anzahl an Materialien wie Papier, Post-its, Stifte etc. bereitzuhalten. Die so generierten, physischen Informationen müssen festgehalten werden. Dazu können sämtliche vertikalen Flächen (also z.B. Wände, Whiteboards, Tafeln, Flip-Chats, ...) benutzt werden, um die Möglichkeit zu bieten, jede Information / Idee zu sammeln und ggf. weiter zu entwickeln. Durch den Einsatz von mobilem Mobiliar (bewegliche Trennwände, Whiteboards usw.) ist es zudem möglich, den Raum für verschiedene Arbeitssituationen anzupassen. (Vgl. [GM13] S. 22)

Neben den Materialien und der Möglichkeit zur flexiblen Raumaufteilung führen Gürtler und Meyer einen weiteren unterstützenden Faktor an: Durch Musik könne die Stimmung und Atmosphäre beeinflusst werden. Dies kann als weiter Katalysator für kreatives Arbeiten dienen. (Vgl. [GM13] S.23)

3.2.3 Die Teamkultur

Die Teamkultur beschreibt die Art und Weise der Zusammenarbeit der Teammitglieder. Dazu müssen die einzelnen Teammitglieder Fähigkeiten und Kompetenzen mitbringen, die allgemein als *Soft Skills* bezeichnet werden. Diese Soft Skills sind zum Teil auch Charaktereigenschaften eines Design Thinkers. Darauf wird in Abschnitt 3.3 näher eingegangen.

Die Mitglieder müssen vor allem sehr gute Kommunikationsfähigkeiten besitzen sowie jedes andere Mitglied mit Respekt behandeln. Dies ist besonders dann wichtig, wenn die Feedback-Kultur angewandt wird, um regelmäßig das Vorgehen und die Arbeitsweise innerhalb des Teams reflektieren zu können. Als weitere Eigenschaft wird der Mut zum Scheitern beschrieben. (Vgl. [GM13] S.25)

Die hier aufgeführten Eigenschaften werden allesamt für die das dritte Prinzip der Greifbarkeit benötigt. Dadurch dass in kurzer Zeit viele Prototypen entwickelt werden, müssen diese auch innerhalb des Teams diskutiert werden. Dazu wird die zuvor genannte Feedback-Kultur benötigt und den Mut, auch auf den ersten Blick „unnütze“ Prototypen zu testen. Im schlechtesten Fall wird dann die Befürchtung, eine schlechte Idee gehabt zu haben, bestätigt. Im besten Fall werden neue Erkenntnisse gewonnen.

3.3 Charakter eines Design Thinker

Thema dieses Abschnitts sind der Charakter und die damit verbundenen Eigenschaften eines Design Thinkers. Ein Design Thinker, ist eine Person, die die Prinzipien des Design Thinking verinnerlicht hat und in der Lage ist diese anzuwenden. Dazu benötigt der Design Thinker verschiedene Eigenschaften, die im folgenden als Ausschnitt beschrieben werden.

In seinem Artikel im Harvard Business Review hat Tim Brown ([Bro08]) fünf Charaktereigenschaften beschrieben, die ein Design Thinker aus seiner Sicht mitbringen sollte. Die erste ausgewählte Eigenschaft ist *Empathie*. Ein Design Thinker soll in der Lage sein, sich in verschiedene Rollen und Perspektiven zu versetzen (z.B. in die Rolle eines Kollegen, eines Entscheiders oder eines Kunden), um ein gestelltes Problem umfassend betrachten zu können. Ein Design Thinker soll somit ein genauer Beobachter sein.

Als zweite Eigenschaft beschreibt Brown das *Integrative thinking*. Hinter diesem Begriff verbirgt sich die Eigenschaft, nicht nur analytische Prozesse zu nutzen, die eine Ja oder Nein Entscheidung herbeiführen, sondern über alternative Ansätze nachzudenken.

Die dritte Eigenschaft die beschrieben wird, ist *Optimismus*. Scheint die Aufga-

be noch so schwer und komplex, so muss der Design Thinker optimistisch bleiben und überzeugt sein, dass es dennoch eine adäquate Lösung gibt. Dies spielt zusammen mit der in Abschnitt 3.2.3 erwähnten Eigenschaft des Mutes.

Als vierte Eigenschaft wird *Experimentalism* angeführt. Ein Design Thinker hinterfragt Probleme und führt zu neuen Diskussionen, die neues Potential führen können.

Die letzte ausgewählte Eigenschaft ist die *Zusammenarbeit*. Ein Design Thinker versteift sich nicht auf ein einzelnes Thema, sondern hat bzw. baut umfassendes Wissen aus anderen Bereichen auf. Er wird somit interdisziplinär.

3.4 Phasen des Design Thinking

In diesem Abschnitt werden Phasen eines Design Thinking Prozesses dargestellt. Wie bereits bei der Definition von Design in Abschnitt 2 dargestellt wurde, gibt es viele verschiedene Definitionen von Design. Ähnlich verhält es sich mit Modellen und Phasen für Design Thinking. Es werden nun verschiedene Ansätze vorgestellt und deren Gemeinsamkeiten bzw. Unterschiede herausgestellt.

In dem zuvor in Abschnitt 3.3 zitierten Artikel aus dem Harvard Business Review findet sich ebenfalls ein Modell für den Ablauf von Design Thinking. Dieses Modell ist dabei in drei Hauptphasen eingeteilt: Inspiration, Ideengenerierung (Ideation) und Implementierung (Implementation) (Vgl. [Bro08] S.88f). Sämtliche dieser Phasen stehen in Beziehungen zueinander. Es ist möglich, von jeder Phase zur nächsten zu schreiten bzw. wieder eine Phase zurück zu gehen. Die drei Phasen sind in Abbildung 3.1 grafisch dargestellt.

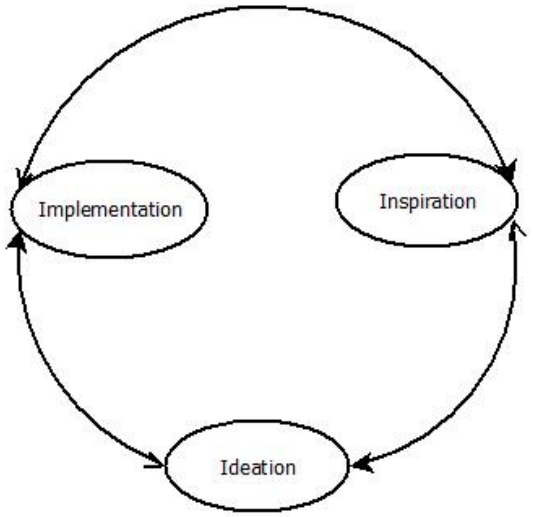


Abbildung 3.1: Der vereinfachte Kreislauf nach Brown (Vgl. [Bro08] S.88f)

Wie in ([Bro08] S.88f) zu sehen ist, sind jeder der drei Phasen verschiedenen Tätigkeiten bzw. Fragen sowie ein „Oberpunkt“ zugeordnet. Der Oberpunkt der ersten Frage lautet *Expect success*, gehe von Erfolg aus. Dazu müssen die Probleme und möglichen Chancen identifiziert werden, indem die Umwelt untersucht wird. Was tun Menschen, wie denken sie, was wollen oder brauchen Sie in Bezug auf das Problem? Wirtschaftliche Aspekte müssen hierbei ebenfalls berücksichtigt werden. Fragen, die dabei geklärt werden müssen sind z.B. die Fragen nach dem Budget oder der Zeit. (Vgl. [Bro08] S.89)

In der zweiten Phase, der Ideengenerierung, lautet der Oberbegriff *Brainstorm*. Nachdem aus der ersten Phase die Rahmenbedingungen aufgenommen worden sind, kann begonnen werden, Ideen in Form von Skizzen und Szenarien zu entwickeln. Dabei wird aus einem oberflächlichen Chaos, bedingt durch eine Vielzahl von Ideen, mit Hilfe von creative frameworks und integrative thinking der Grundstein für die Prototypen Entwicklung gesetzt. Dabei ist hervorzuheben, dass nicht nur ein Prototyp entwickelt und getestet wird, sondern mehrere. Wichtig dabei ist eine stete Kommunikation aller Beteiligten. (Vgl. [Bro08] S.88)

Die Phase der Ideengenerierung geht fließend in die dritte Phase der Implementierung über. Nachdem in der zweiten Phase durch mehrere Iterationen des Prototypings eine Vision entstanden ist, wird diese nun in der dritten Phase implementiert. Das Motto der dritten Phase lautet demnach *Execute the Vision*, "Verwirkliche die Vision". Neben der eigentlichen Implementierung gehören zu dieser Phase auch Bereiche des Marketing. Ist diese Phase abgeschlossen, wird auch das Projekt abgeschlossen und der Kreislauf beginnt mit dem nächsten Projekt. (Vgl. [Bro08] S.88)

Ein weiteres Modell für Design Thinking wird in ([MLP11] S. xiv) dargestellt. Im Gegensatz zu dem oben dargestellten Modell, wurden hier fünf Phasen identifiziert: Define the Problem (Problemdefinition), Needfinding and Benchmarking (Anfoderungsanalyse), Bodystorm, Prototype sowie Test. Besonders wird in ([MLP11] S. xiv) darauf hingewiesen, dass es sich keineswegs um einen geordneten Kreislauf handelt, sondern das von jeder Phase in eine andere gesprungen werden kann. Damit ist die Verknüpfung der Phasen noch flexibler als in dem zuvor beschriebenen Modell von Brown. Dieser Sachverhalt ist in nachfolgender Abbildung 3.2 grafisch dargestellt worden.

Links ist der theoretische Ablauf dargestellt. Der rechte „Kreislauf“ entspricht nach ([MLP11]) eher der Realität. Sämtliche hier definierten Phasen sind auch in dem Modell von Brown, zum Teil als Unterpunkte, enthalten. Ein Unterschied zeigt sich zwischen Brainstorming, das Brown als Oberpunkt seiner Phase der Ideengenerierung aufgeführt hat, und dem von [MLP11] aufgeführten Bodystorming. Der Begriff Bodystorming beschreibt ein Vorgehen, in dem eine Situation physisch

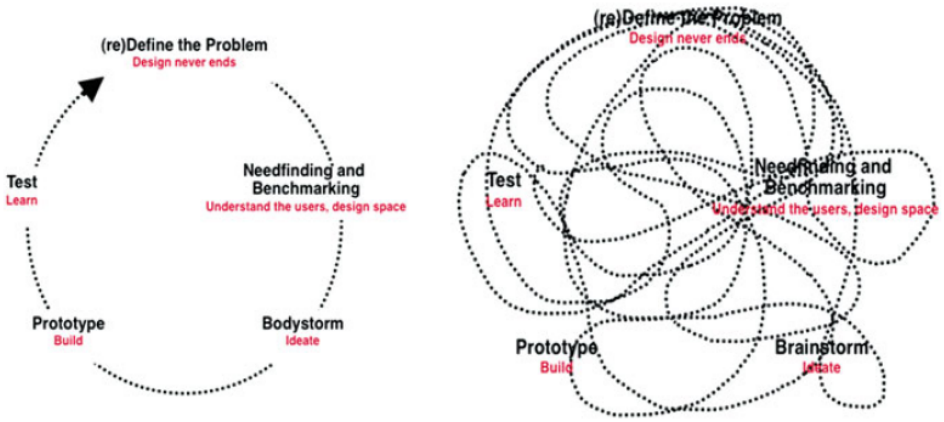


Abbildung 3.2: Design Thinking Phasen. Bildquelle: ([MLP11] S. xiv)

erlebt wird (z.B. mit Hilfe von Prototypen), um neue Ideen und Ansätze zu generieren (Vgl. [WG14]). Es geht also über das „herkömmliche“ Brainstorming hinaus.

Stellt man nun die beiden Ansätze gegenüber, so ist zu sagen, dass beide Ansätze ähnliche Ziele mit ähnlichen Methoden verfolgen. Beide Kreisläufe definieren zunächst das Problem und identifizieren Rahmenbedingungen aus der Anwendungsdomäne. Wichtig dabei ist, wie auch die Definition aus Abschnitt 3.1 zeigt, den Fokus auf den Menschen bzw. den Nutzer zu legen. Wurden möglichst alle relevanten Informationen gesammelt, geht der Kreislauf in die Ideenfindung über. Mögliche Techniken sind dabei z.B. das Bodystorming. Daraus werden entsprechende Prototypen entwickelt, die zusammen mit dem Nutzer evaluiert werden. Hat sich eine Richtung herauskristallisiert wird diese weiter verfolgt und schließlich implementiert und getestet.

4 Design Science

In diesem Abschnitt wird das Themenfeld des Design Science und zusammenhängend damit auch Design Science Research behandelt. Es wird zunächst der Begriff Design Science (Research) definiert. Dazu werden artverwandte Begriffe abgegrenzt. Im Anschluss daran wird das von Hevner et. al. entwickelte Design Science Research Framework vorgestellt. Zu diesem Framework geäußerte Anmerkungen und Kritik werden ebenfalls beleuchtet.

4.1 Definition - Design Science (Research)

Hevner et. al. beschreiben Design Science als fundamentales Paradigma mit dem Ursprung in den Ingenieurwissenschaften (Engineering). Dieses Paradigma nutzt nach Hevner et. al. Innovationen, Ideen und Praktiken, um den Erstellungsprozess eines Informationssystems zu optimieren. Optimiert wird dabei z.B. Design, Implementierung oder Management des Informationssystems. (Vlg. [HMPR04] S. 76) Weiter beschreiben Hevner et. al. (Vgl. [HMPR04] S.76), dass Design Science innerhalb des Information System Research Cycle IT Artefakte[1] generiert und evaluiert. Wird von Design Science gesprochen, so fällt auch der Begriff des *Design Science Research*. Desing Science Research ergänzt den Fokus des Design Science auf IT Artefakt um den starken Fokus zum Anwendungsfeld, in dem das Informationssystem eingesetzt werden soll, um Probleme zu lösen. Zudem wird durch Design Science Research die Effektivität und Benutzbarkeit eines generierten IT Artefakts für eine Anwendungsdomäne untersucht und optimiert.(Vgl. [HC10] S. 9f) Weiter wird auch der Prozess, der hinter der Erstellung des IT Artefakts steht, untersucht und festgehalten.

Nach der Definition von Johannesson und Perjons, ist Design Science die wissenschaftliche Untersuchung von Artefakten, die von Menschen erstellt und genutzt werden. Außerdem definieren sie das Ziel von Design Science als das Lösen von Problemen. (Vgl. [JP14] S. 7) Somit sind die Definitionen von Hevner et. al. und Johannesson und Perjons ähnlich.

[1] Der Begriff *Artefakt* dient zur Beschreibung sämtlicher gestalteter Dinge und wird weitläufig als ein von Menschenhand gefertigtes Produkt verstanden (Vgl. [EM08] S. 19). Ein IT Artefakt bewegt sich dabei in dem Kontext der IT.

Mit Hilfe der generierten IT Artefakte wird es Menschen ermöglicht, weitere existierende Probleme zu lösen. Die Grenzen, die anzeigen welche Probleme gelöst werden können und welche nicht, werden somit verschoben. Auch die organisatorischen Möglichkeiten können durch entsprechende Tools erweitert und verbessert werden. Wird nun Desing Science Research hinzugezogen, so sehen sich Design Science Researcher in der Lage, den Konstruktions- und Evaluationsprozess des Artefakts für ein konkretes Problem nachzuvollziehen und die Methodik vom Ansatz bis zu der Lösung zu untersuchen (Vgl. [HMPR04] S.77). Somit kann gesagt werden, dass Design Science sich vorrangig damit beschäftigt, Lösungen für Probleme im Informationssystemumfeld zu finden und Design Science Research den Findungsprozess unterstützt und anschließend evaluiert, um ggf. neue Ansätze und Methodiken zu identifizieren, die wiederverwendet werden können. Für den weiteren Verlauf dieser Arbeit, werden beide Praktiken betrachtet, da sie eng miteinander verzahnt sind.

Um mit Hilfe von Design Science (research) die zuvor genannten Grenzen umfassend zu erweitern, argumentieren Hevner et. al. ([HMPR04] S. 76f), dass eine Verknüpfung der disjunkten, aber kompatiblen Ansätze des Design Science und *Behavioural Science* nötig ist. Behavioural Science, dessen Wurzeln in den Naturwissenschaften liegt, versucht Theorien und Regeln zu identifizieren und zu beweisen, die zeigen oder vorhersagen, wie Menschen und Organisationen auf Informationssysteme und mit ihnen reagieren. Dies umfasst ebenfalls die Analyse, das Design, die Implementierung und das Management von Informationssystemen. (Vgl [HMPR04] S.76)

Dadurch, dass der Ansatz des Design Science den Fokus auf die Technik legt, und das Behavioural Science den Fokus auf das Verhalten des Systems und des Menschen hat, schafft die Kombination der beiden Ansätze eine umfassende Betrachtung und Analyse eines Informationssystems. Dies ist ein iterativer Ablauf. Begonnen wird mit der Entwicklung von Theorien und Artefakten im Bereich des Design Science. Die entwickelten Theorien und Artefakte werden dann durch den Bereich des Behavioural Science bewiesen bzw. evaluiert. Dies wird auch als *Design Cycle* bezeichnet. Das gerade Beschriebene kann als Kreislauf verstanden werden und wird weiter ergänzt um den Aspekt des Umfelds (Environment) und der Wissensbasis (Knowledge Base). Der daraus resultierende Ansatz wird als Information System Research Framework bezeichnet und ist in Abschnitt 4.3 näher beschrieben.

4.2 Abgrenzungen zu artverwandten Themen

Um das Verständnis von Design Science (Research) zu schärfen, werden in diesem Abschnitt weitere Begriffe aus der Thematik identifiziert und abgegrenzt. Dazu zählt die Unterscheidung von Design Science Research und Professional Design sowie die Klärung von Design as Research und Researching Design.

Design Science Research und Professional Design

Der größte Unterschied zwischen Design Science Research und Professional Design ist die Art des Problems, welches untersucht wird. Professional Design löst bekannte Probleme mit bekannten Artefakten. Ein Beispiel dafür ist die Entwicklung eines Informationssystems für die Finanzwelt. Bekannte Artefakte beinhalten dabei vor allem Best Practices Ansätze sowie Modelle und Methoden. Im Gegensatz zum Professional Design beschäftigt sich Design Science Research, wie schon in der Definition in 4.1 erwähnt, mit komplexen ungelösten Problemen. Dabei wird nach innovativen Lösungsansätzen gesucht, um das Problem zu lösen. Zu erwähnen ist, dass Design Science Research sich auch mit schon gelösten Problemen befassen kann. Hier wird versucht, der Weg zu der Lösung, bzw. die Methodik, zu verbessern und zu optimieren. (Vgl. [HC10] S.15)

Design as Research und Researching Desing

Design as Research hat zum Ziel, durch innovatives Design einen Beitrag zur bestehenden Knowledge base zu leisten. Dieser Beitrag kann z.B. in Form von Modellen und Methodiken vorliegen. Der Fokus liegt auf der Erstellung und Verbesserung von Artefakten, die zur Lösung eines Problems beitragen. Dabei ist zu beachten, dass Design as Research oftmals auf eine spezifische Anwendungsdomäne begrenzt ist. Ausgehend von diesen Artefakten wird dann versucht eine allgemeinere und weiter gefasste Methodik zu identifizieren. Dies bedeutet, dass eventuell weitere Schritte nötig sind, um aus den identifizierten Modellen oder Methodiken allgemeingültige Verfahren zu extrahieren. Im Gegensatz dazu untersucht Researching Design voranging den Prozess des Designs und den Designer selbst. Ziel ist es, domänenübergreifende Methoden und Modelle zu entwickeln, die für einen Designprozess genutzt werden können. (Vgl. [HC10] S.15f)

4.3 Information System Research Framework nach Hevner et. al.

In diesem Abschnitt wird das von Hevner et. al. entwickelte Information System Research Framework vorgestellt. Zunächst wird ein Überblick über das Framework gegeben. Dabei wird vor allem das Zusammenspiel der einzelnen Kompo-

nenten in den Vordergrund gehoben. In den Unterabschnitten dieses Abschnitts werden die einzelnen Komponenten näher betrachtet.

Wie in Abbildung 4.1 zu sehen ist, besteht das Framework aus drei Hauptkomponenten. Auf der linken Seite ist die Komponente Environment (Umgebung) zu sehen. Sie beinhaltet sämtliche Elemente aus der Anwendungsdomäne, in dem das zu entwickelnde Informationssystem eingesetzt werden soll. Auf der rechten Seite ist die Knowledge Base (Wissensbasis) abgebildet. Die Wissensbasis enthält verschiedene Methoden und Modelle, um z.B. Artefakte zu konstruieren. Der Mittelteil des Frameworks besteht aus dem Informationssystem Research Cycle, der in Abschnitt 4.1 bereits eingeführt wurde.

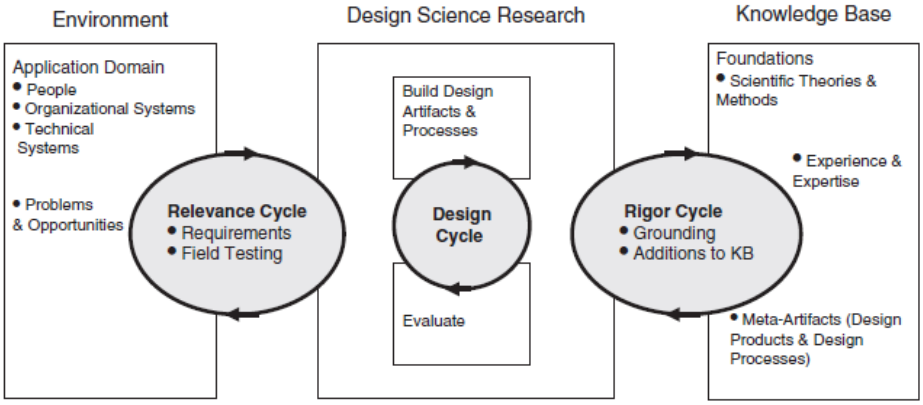


Abbildung 4.1: Design Science Framework mit Research Circles, Bildquelle: ([HC10] S. 16)

Wie zu sehen ist, wirken sowohl die linke als auch die rechte Seite auf den Bereich des Design Science Research ein. Dabei stellt die Anwendungsdomäne die Anforderungen, Bedürfnisse und bestehende Informationen in Form von Business Needs / Requirements (Anforderungen) zur Verfügung. Durch die Wissensbasis fließen Theorien und Modelle in den Research Cycle. Innerhalb des Design Cycle werden nun die Information, die durch beide Seiten bereitgestellt werden aufgenommen, um über Desing Science Artefakte und Theorien zu erzeugen. Dies ist in der Abbildung als „Building Design Artifacts & Processes“ bezeichnet. Nach der Konstruktion, wird mit Hilfe der Behavioural Science das Artefakt überprüft und evaluiert. Zwischen diesen Phasen besteht eine stetige Iteration, um nach Möglichkeit das Artefakt weiter zu verbessern. Hevner et. al. bezeichnen das Ziel des Develop / Build Prozesses als utility (Nutzen). Als Ziel für den Prozess des Justify / Evaluate nennen sie truth (Wahrheit). Beide Eigenschaften, Nutzen und Wahrheit, sind nach Hevner et. al. nicht trennbar. (Vgl. [HMPR04] S. 80) So ist zu sagen, dass ein erzeugtes Artefakt nur von Nutzen ist, wenn es wahr ist. Anders ausge-

drückt: ein Artefakt welches falsche Werte liefert ist auch nicht von Nutzen für den Anwender.

Aus dem Kreislauf können schließlich zwei Ergebnisse gewonnen werden. Das erste ist z.B. ein Artefakt, das als Anwendung zurück in die Anwendungsdomäne geführt wird, um die zuvor gestellten Business Needs oder Anforderungen zu befriedigen. Dies ist dann das Ergebnis des Design Science. Als weiteres Ergebnis kann z.b. eine neue oder verbesserte Methode zurück in die Wissensbasis geführt werden. Dies ist Ergebnis des Design Science Research.

4.4 Anmerkungen zu dem Framework nach Hevner et. al.

Dieser Abschnitt behandelt Anmerkungen und Kritiken an dem zuvor in Abschnitt 4.3 vorgestellten Information System Research Framework von Hevner et. al..

In ([GH10] S. 94) wird besonders die starke Fokussierung auf das Erstellen einer Infrastruktur für Informationssysteme kritisiert. Weiter wird dort argumentiert, dass das IT Artefakt im Mittelpunkt der Betrachtung steht. Der Design Science Research baut hier in erster Linie Wissen durch das Erstellen und Evaluieren dieses IT Artefakts auf. Erst danach wird das IT Artefakt in einem Organisation-/Anwendungskontext gesetzt, das in ([GH10] S. 95f) als aufteilen in zwei Kategorien von Design-Wissen aufgefasst wird. Dies wirkt dem weit gefassten Zielt entgegen, eine Wissenschaft für Design in Informationssystemen zu entwickeln.

Zusammengefasst wird an dem Framework von Hevner et. al. kritisiert, dass der Fokus zu technisch gehalten ist und damit eine Teilung des aufzubauenden Design-Wissens gefördert wird, das nach ([GH10] S.96) nicht im Sinne der „Information System academic community“ liegt.

5 Fazit

Im Rahmen dieser Arbeit wurde in die umfangreiche Thematik des Design Thinkings und des Design Science eingeführt. Dazu wurde zunächst dargestellt, dass der Begriff Design selbst nicht eindeutig zu definieren ist. Im Rahmen dieser Arbeit wurde Design als interdisziplinärer Prozess angesehen, mit dessen Hilfe komplexe Probleme gelöst werden können.

Im Anschluss an die Definition von Design wurde der Themenbereich des Design Thinking beschrieben. Desing Thinking wurde dabei als Methodik definiert, die auf den Nutzer zugeschnitten ist und mit Hilfe von Tools und Technik gestellte komplexe Probleme zu lösen versucht und somit Innovationen vorantreibt und entwickelt. Des Weiteren wurde gezeigt, dass die Prinzipien Viele-Augen-Prinzip (Many Eyes), Nutzersicht (Customer view point) sowie Greifbarkeit (Tangibillity) als Voraussetzungen für erfolgreiches Design Thinking gelten. Neben diesen Voraussetzungen sollte jedes Mitglied des Teams Charaktereigenschaften wie Empathie, Optimismus sowie Teamfähigkeit mitbringen. Im Anschluss wurde dargestellt, dass es für Design Thinking Modelle mit verschiedenen Phasen gibt. Im Kern beschreiben alle Modelle ein Vorgehen, in dem zunächst die Problemstellung zusammen mit der Anwendungsdomäne untersucht wird und der Nutzer im Fokus ist. Nach der Sammlung von Informationen wird über Brain und Bodystorming Ideen generiert, die jeweils prototypisch umgesetzt und zusammen mit dem Nutzer evaluiert werden. Daraus kristallisiert sich eine Lösung, die letztlich implementiert wird.

Das zweite in dieser Arbeit betrachtete Themenfeld ist Design Science. Design Science wurde zunächst als wissenschaftliche Untersuchung und Generierung von IT Artefakten zur Lösung von komplexen Problemen beschrieben. Wird der Prozess der Erstellung bzw. der Weg zum IT Artefakt untersucht, so wird weiter von Design Science Reserarch gesprochen. Es wurde zudem gezeigt, dass Design Science in Form eines Design Circle auftritt und in Wechselwirkung mit dem Behavioural Science steht. Aufgabe des Behavioural Science ist es, generierte Artefakte zu evaluieren bzw. zu beweisen. Weiter wurde das Design Science Research Framework nach Hevner et. al. vorgestellt welches den Design Circle um den Bereich des Environment und der Knowledge Base erweitert.

Im Rahmen der Projektgruppe kann vor allem Design Thinking eingesetzt wer-

den. Dies liegt besonders an der dynamischen und vergleichsweise einfachen Handhabung von Design Thinking. Darüber hinaus decken sich die Phasen des Design Thinking mit dem geplanten Vorgehen der Projektgruppe, sodass eine Verknüpfung sich hier anbietet. Die Erzeugung und Evaluierung von Artefakten kann im Rahmen der Projektgruppe durch aus interessant sein, kurz: Der Design Circle könnte zur Anwendung gebracht werden.

Literatur

[Bro08] BROWN, Tim: *Design Thinking: Thinking like a designer can transform the way you develop products, services, processes - and even strategy*. 2008

[Den13] DENNING, Peter J.: Design thinking. In: *Communications of the ACM* 56 (2013), Nr. 12, S. 29–31. http://dx.doi.org/10.1145/2535915. – DOI 10.1145/2535915. – ISSN 00010782

[EM08] ERLHOFF, Michael ; MARSHALL, Tim: *Wörterbuch Design: Begriffliche Perspektiven des Design*. Basel and Boston : Birkhäuser Verlag, 2008 (Board of International Research in Design). – ISBN 9783764377380

[GH10] GREGOR, Shirley D. ; HART, Dennis N.: *Information systems foundations: The role of design science*. Acton and A.C.T : ANU E Press, 2010. – ISBN 9781921666346

[GM13] GÜRTLER, Jochen ; MEYER, Johannes: *30 Minuten Design Thinking*. Offenbach am Main : GABAL, 2013 (30 Minuten-Reihe : in 30 Minuten wissen Sie mehr). – ISBN 9783869364865

[HC10] HEVNER, Alan R. ; CHATTERJEE, Samir: *Integrated series in information systems*. Bd. v. 22: *Design research in information systems: Theory and practice*. New York and London : Springer, 2010. – ISBN 9781441956521

[HMPR04] HEVNER, Alan R. ; MARCH, Salvatore T. ; PARK, Jinsoo ; RAM SUDHA: Design Science in Information Systems Research. In: *MIS Quarterly* (2004), Nr. 28, S. 75–105

[IDE15] IDEO: *About IDEO: What we do*. https://www.ideo.com/about/, 2015. – Zugegriffen am 31.10.2015

[JP14] JOHANNESSON, Paul ; PERJONS, Erik: *An introduction to design science*. New York and London : Springer, 2014. – ISBN 9783319106311

[MLP11] MEINEL, Christoph ; LEIFER, Larry ; PLATTNER, Hasso: *Design Thinking*. Berlin and Heidelberg : Springer Berlin Heidelberg, 2011. http://dx.doi.org/10.1007/978-3-642-13757-0. http://

dx.doi.org/10.1007/978-3-642-13757-0. – ISBN 978–3–642–13756–3

[WG14] WITTHOFT, Scott ; GEEHR, Carly: *Bodystorming*. https://dschool.stanford.edu/groups/k12/wiki/48c54/Bodystorming.html, 2014. – Zugegriffen am 28.11.2015

Abschließende Erklärung

Ich versichere hiermit, dass ich meine Seminararbeit Design Thinking / Design Science selbständig und ohne fremde Hilfe angefertigt habe, und dass ich alle von anderen Autoren wörtlich übernommenen Stellen wie auch die sich an die Gedankengänge anderer Autoren eng anlegenden Ausführungen meiner Arbeit besonders gekennzeichnet und die Quellen zitiert habe.

Oldenburg, den 26. September 2016 Hauke Precht