CARL VON OSSIETZKY universität OLDENBURG

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik
Abteilung Systemsoftware und verteilte Systeme

– Seminar –

Fehlertolerante Systeme

24.07.2012

Inhaltsverzeichnis

Fehlertoleranz im Network Time Protocol

Jan Steffen Becker
Department für Informatik
Universität Oldenburg
eMail: jan.steffen.becker@uni-oldenburg.de

***Zusammenfassung*—Dieses Paper beschäftigt sich mit der Synchronisation von Uhren über Netzwerke mit besonderem Hinblick auf dabei auftretende Probleme.**

Zunächst werden die zu berücksichtigenden Fehler analysiert. Darauf aufbauend wird ein von K. Marzullo und S. Owicki beschriebener Algorithmus und das in Teilen darauf basierende Network Time Protocol in der aktuellen Version 4 dargestellt. Dabei werden Lösungen für die erkannten Probleme aufgezeigt.

Abschließend wird anhand eines Experiments gezeigt, welchen Gewinn das Network Time Protocol dadurch gegenüber einfacheren Zeitdiensten erreicht.

***Keywords*-Fehlertoleranz, Zeitsynchronisation, Network Time Protocol, Marzullo-Algorithmus, Byzantinische Fehler**

I. Einleitung

Eine genau gehende Systemuhr ist für die meisten Benutzer im Internetzeitalter schon längst eine Selbstverständlichkeit. Die gemeinsame Benutzung von Dateien innerhalb eines Netzwerks ist ein einfaches Beispiel, wo synchronisierte Uhren nötig sind. Hier muss sichergestellt werden, dass beim Austausch von Dateien die Zeitstempel konsistent bleiben, beispielsweise um zu erkennen, welche Version einer Datei aktueller ist.

Das Network Time Protocol (NTP) ermöglicht diese Synchronisation von Uhren mit einer Genauigkeit von unter einer Millisekunde in schnellen lokalen Netzen und wenigen hundertstel Sekunden über das Internet. Durch Open Source Software und hunderte öffentlich zugänglicher Zeitserver im Internet findet es vor allem in der Unix-Welt weite Verbreitung. Aber auch Microsoft stellt seinen Nutzern ab Windows XP standardmäßig einen auf NTP basierten Zeitdienst zur Verfügung. Das NTP ist von David L. Mills 1985 in RFC 958 erstmals spezifiziert worden. Die aktuelle Version ist NTPv4, wie sie in RFC 9505 definiert ist.

Im Folgenden wird zunächst einmal ein Algorithmus zur Zeitsynchronisation von Keith Marzullo und Susan Owicki [1] vorgestellt. Dieser kann gewissermaßen als Grundlage für das NTP verstanden werden. Das NTP erweitert den Marzullo-Algorithmus um weitere Algorithmen, um eine größere Genauigkeit zu erzielen.

II. Generelles

A. Aufbau des Netzwerks

Das Network Time Protocol [2] spezifiziert einen Netzwerkdienst zur Synchronisation von (Computer-)Uhren in einem Netzwerk. Da das Protokoll vor allem von Millionen Internetnutzern verwendet wird, haben wir es hier mit großen Entfernungen und vielen miteinander vernetzten Servern zu tun.

Um das Problem genau betrachten zu können, muss zunächst ein Überblick über die Struktur dieses Netzwerks gegeben werden:

Das NTP geht von beliebig vielen Zeitservern aus, die in Schichten organisiert sind. Nur Zeitserver der ersten Schicht, als *stratum 1* bezeichnet, sind direkt an einen Zeitgeber, beispielsweise einen GPS-Empfänger[1], angeschlossen. Die Server in den darüber gelegenen Schichten synchronisieren sich jeweils über das Netz mit einem oder mehreren Servern einer darunter gelegenen Schicht.

Keith Marzullo und Susan Owicki gehen in ihrer Arbeit [1], die dem Synchronisationsalgorithmus im NTP zu Grunde liegt, hingegen von einem einfacheren voll vermaschten Netzwerk aus, in dem sich jeder Server mit jedem synchronisieren kann.

B. Betrachtung des Fehlermodells

1) Netzwerkfehler: Im Gegensatz zu anderen Anwendungen in Rechnernetzen, sind hier auftretende Netzwerkfehler nicht so sehr der Verlust oder die Beschädigung von Paketen. Die Datenintegrität ist durch die Fehlererkennung in den darunterliegenden Netzwerkprotokollen IP und UDP gegeben [2]. Ausbleibende Antworten, sowie Pakete mit inkonsistenten Zeitstempeln, werden erkannt.

Vielmehr sind das hier die Fehler, die durch die Latenzzeit des Netzwerks entstehen [3]. Sie verursachen den Großteil der bei der Zeitsynchronisation auftretenden Fehler. Sie sind auch nur sehr schwer abzuschätzen, besonders wenn keine genauen Kenntnisse über das Netzwerk vorliegen, in dem das System eingesetzt wird.

2) Messfehler: Computeruhren bestehen aus einer Taktquelle, meist einem Uhrenquarz, dessen Taktfrequenz durch einen *Prescaler* auf 100Hz oder 1000Hz herunterdividiert wird, der dann in Hardware oder Software (durch einen Interrupt) einen Zähler inkrementiert, der vom Prozessor ausgelesen werden kann [3].

[1]Das *Global Positioning System* arbeitet mittels relativer Zeitstempel. Einem Empfänger steht dadurch automatisch ein hoch präzises Zeitsignal zur Verfügung.

Die Uhren des Clients und des Servers laufen nur mit einer begrenzten Genauigkeit. So ist eine Computeruhr von Natur aus diskret, die Reale Zeit aber stetig, was beim Lesen der Uhr zu einem Fehler im Intervall $[-\rho, 0]$ führt, wobei ρ die Präzision *(precision)* oder *max reading error* einer Uhr genannt wird. Dazu unterliegt die Frequenz des Impulsgebers der Uhr einer gewissen Abweichung $f \in [-\varphi, \varphi]$, wobei φ der *max frequency error* genannt wird. Er kommt unter anderem daher, dass die Frequenz, mit der der Uhrenquarz schwingt, durch die Umgebungstemperatur beeinflusst wird. Dies wird von D. Mills in [3] genauer dargestellt.

3) Ererbte Fehler: Das weltweite Netzwerk, in dem das Network Time Protocol betrieben wird, ist hierarchisch aufgebaut. Genaue Zeitgeber synchronisieren *stratum 1*-Server, diese synchronisieren wiederum *stratum 2*-Server und Clients. Diese Schichtenstruktur bewirkt natürlich, dass sich die Fehler zu den Clients hin vergrößern. Jeder Server gibt seine Fehler an seine Clients weiter. Diese werden *inherited errors* – ererbte Fehler – genannt.

4) Byzantinische Fehler: Das NTP soll im Internet arbeiten. Es kann nicht davon ausgegangen werden, dass alle Server, mit denen sich ein Client oder Server synchronisiert, richtig arbeiten. Dadurch entstehen die schon genannten *Byzantinischen Fehler*. Unter den Servern können sich einige „Verräter“ befinden – im NTP *falsetickers* – die ein völlig undefiniertes Verhalten zeigen. Dies kann bis hin zu „two faced clocks“ gehen, die jedem Client eine andere, zufällige Zeit nennen [4]. Wenige solcher Server dürfen das gesamte System nicht gefährden.

Dies Fehler haben ihren Namen nach einem Lösungsansatz, der schon im alten Byzanz zum Entlarven von Verrätern in der Armee angewandt worden sein soll [5]. Dieser allgemeine Lösungsansatz kann ein einheitliches Verhalten aller „loyalen“ Server sicherstellen, wenn weniger als ein Drittel aller Server „Verräter“ sind, – das ist die maximale Anzahl, bei der das Problem in der allgemeinen Form lösbar ist – selbst wenn der primäre Server, dessen Daten alle anderen verwenden, darunter ist.

Dieser Algorithmus arbeitet aber auf diskreten Daten und bewirkt einen Nachrichtenaustausch, der exponentiell zur erlaubten Anzahl an Verrätern ist.

Für die Synchronisation von Zeitservern dienen andere Algorithmen, wie sie in [4] beschrieben sind. Diese finden aber im NTP keine direkte Anwendung.

Das NTP ist dennoch so entworfen, dass Byzantinische Fehler berücksichtigt werden [2]. In den *clock-select-* und *cluster*-Algorithmen (siehe IV-B) wird versucht, *falsetickers* auszusortieren.

In [6] wird gezeigt, dass die Voraussetzung von weniger als einem Drittel „Verrätern“ auch für die Zeitsynchronisation gilt.

C. Der Synchronisationsvorgang

Um sich mit einem Server zu synchronisieren, sendet ein Client eine Nachricht an den Server und erhält eine Nachricht mit einem Zeitstempel zurück. In der Annahme, dass der Zeitstempel korrekt ist, gibt er aus Sicht des Clients die Zeit zu einem beliebigen, aber nicht näher bestimmbaren Zeitpunkt zwischen dem Absenden der Anfrage und dem Erhalt der Antwort an.

Wenn eine Anfrage nicht nur an einen, sondern zugleich an mehrere Server geschickt wird, ist es wahrscheinlich, dass der Client so von jedem Server eine etwas anderen Zeitstempel bekommt, da die Anfragen zu unterschiedlichen Zeiten beim Server eingehen.

Die Frage, welche Zeit nun vom Client übernommen werden soll, behandeln Marzullo und Owicki in ihrer Arbeit [1].

III. Der Marzullo-Algorithmus

Marzullo und Owicki betrachten eine Uhr als stetige Funktion $t \mapsto C(t)$, die der realen Zeit t *(real time)* eine Uhrzeit $C(t)$ *(clock time)* zuordnet. Es wird im Folgenden angenommen, dass für jede Uhr C_i eine obere Schranke φ_i für ihre mit der Zeit wachsende Abweichung *(maximum drift rate)* bekannt ist, so dass, wenn $C_i(t_0) = t_0$ gilt, zu einem beliebigen späteren Zeitpunkt $t > t_0$ dann

$$|C_i(t) - t| < (t - t_0)\varphi_i \tag{1}$$

angenommen werden kann.

A. Synchronisation bei Marzullo

Ein Zeitserver S_i hat eine Uhr C_i mit bekanntem φ_i. Ebenso ist der Zeitpunkt r_i sowie der maximale Fehler ϵ_i bei der letzten Synchronisation mit anderen Servern bekannt. Daraus lässt sich zu einem beliebigen späteren Zeitpunkt t ein maximaler Fehler $E_i(t) = \epsilon_i + (t - r_i)\varphi_i$ berechnen. Die Bestimmung von ϵ_i wird später erklärt.

Davon ausgehend, dass ein Server korrekt arbeitet, muss also die reale Zeit immer im Intervall $[C_i(t) - E_i(t), C_i(t) + E_i(t)]$ liegen.

Diesen Umstand, dass zwar die reale Zeit nicht bekannt ist, aber dafür ein Intervall, in dem sie liegt, lässt sich gut zur Synchronisation nutzen.

Marzullo und Owicki schlagen hier zwei Protokolle vor. In beiden Fällen schickt ein Client S_i – wie bereits zuvor beschrieben – eine Anfrage an mehrere Server S_j. Jeder Server antwortet mit seiner Uhrzeit C_j und dem maximalen Fehler E_i.

Es ist zwar nicht bekannt, wann ein Server eine Anfrage bearbeitet, der Zeitraum wird aber durch das Abschicken der Anfrage und das Eingehen der Antwort begrenzt. Diese Zeitspanne kann mit der Uhr des Clients gemessen werden und sei ξ_j^i. Unter Berücksichtigung von φ_i des Clients muss also die reale Zeit beim Eintreffen der Antwort im Intervall

$$[C_j - E_j, C_j + E_j + (1 + \varphi_i)\xi_j^i] \tag{2}$$

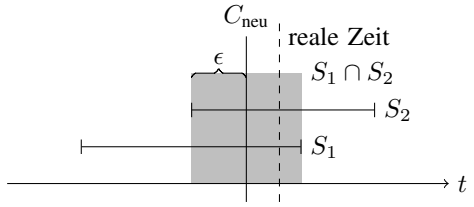

Abbildung 1. Synchronisation bei Marzullo

liegen. Die untere Schranke ergibt sich aus der Annahme, dass die Antwort unmittelbar eingeht (der Zeitstempel aktuell ist) und die Uhr des Servers „vor geht“. Die Obere Schranke aus der entgegengesetzten Annahme, dass die Anfrage ohne Verzögerung beim Server eintrifft und bearbeitet wird (seit dem Erstellen des Zeitstempels also $(1+\varphi_i)\xi_j^i$ Zeit vergangen ist – solange hat dann die Antwort gebraucht) und der Server „nach geht“.

Die erste Möglichkeit ist, die Uhr mit dem Server zu synchronisieren, für den sich das kleinste Intervall ergibt.

Die Zweite Möglichkeit ist die Konsequenz aus folgender Überlegung: Wenn alle Server korrekt arbeiten, muss die reale Zeit auch in allen berechneten Intervallen liegen.

Dazu wird für jeden antwortenden Server S_j das obige Intervall *relativ* zu C_i bestimmt. Es ergibt sich für jede Antwort ein Zeitintervall $[T_j, L_j]$ mit

$$T_j = C_j - E_j - C_i \tag{3}$$

und

$$L_j = C_j + E_j + (1 + \varphi_i)\xi_j^i - C_i \tag{4}$$

.

Der Schnitt der Intervalle ergibt sich aus $[\max T_j, \min L_j]$. Die Zeit des Clients wird dann durch die Mitte des Schnittintervalls angeglichen und ϵ_i auf die Hälfte der Breite des Intervalls gesetzt. Abbildung 1 verdeutlicht dies anhand der Zeitintervalle für zwei Server S_1 und S_2.

IV. Synchronisation im NTP

Dieser Weg der Synchronisation erfordert aber die Existenz eines Schnittintervalls.

Da aber davon ausgegangen werden muss, dass nicht alle Server richtig arbeiten, ist dies so nicht immer gegeben. Der Algorithmus muss demnach so verändert werden, dass mit dieser Fehlerklasse, sogenannten Byzantinischen Fehlern, umgegangen werden kann.

Das Network Time Protocol berücksichtigt dies.

Obwohl der Algorithmus der aktuellen Version vier des NTP mit dem zuletzt vorgestellten bis zu diesem Punkt übereinstimmt, sollte er hier der Vollständigkeit halber ganz umrissen werden.

Die Synchronisation läuft im NTP in vier Schritten ab. An erster Stelle steht der *peer process*, der für jeden Server eingehende Antworten sammelt. Diese werden in den *clock select*, *cluster* und *clock combine* Algorithmen verarbeitet, bis man am Ende einen genauen Schätzwert für die Abweichung der lokalen Uhr von der realen Zeit erhält.

A. Der peer-Prozess

Das NTP arbeitet mit vier Zeitstempeln t_1 bis t_4 pro Server, mit dem synchronisiert wird, die gemäß ihrer zeitlichen Reihenfolge nummeriert sind – dem Zeitpunkt des Absendens der Anfrage, dem Eingehen der Anfrage, dem Absenden der Antwort und schließlich dem Empfangen der Antwort. Sie werden in den Nachrichten nacheinander ergänzt. Auch t_1 wird zum Server und zurück gesendet, sodass der Client ihn zur Kontrolle überprüfen kann.

Daraus wird im Client für jede Verbindung zu einem Server S_i der *offset*

$$\theta_i := \frac{(t_2 - t_1) + (t_3 - t_4)}{2} \tag{5}$$

und der *round-trip delay*

$$\delta_i = (t_4 - t_1) - (t_3 - t_2) \tag{6}$$

berechnet. Der *offset* ist die Differenz der Uhren vom Client und Server, wobei die einfache Annahme getroffen wird, dass die Anfrage vom Client zum Server und die Antwort vom Server zum Client gleich lange unterwegs sind. Der *round trip delay* gibt an, wie lange das Senden dieser beiden Nachrichten insgesamt gedauert hat, und entspricht ξ_i^j bei Marzullo und Owicki. Das Fehlen des Fehlerkorrekturterms wird in späteren Berechnungen kompensiert.

Pro Server – im NTP hier *peer* genannt – werden die letzten acht Zeitstempel in einem Shift-Register gesammelt. Es wird immer die Antwort zur Synchronisation benutzt, die die größte Genauigkeit verspricht. Untersuchungen haben gezeigt, dass der Offset genauer ist, je kleiner δ_i ist. Es werden daher immer die Werte für den *clock-select*-Algorithmus verwendet, die aus der Antwort im Shiftregister mit dem kleinsten δ_i entstehen. Um die Werte aktuell zu halten, wird das Shift-Register bei ausbleibenden Antworten mit Dummy-Werten gefüllt. So verbleibt keine Antwort mehr als acht Synchronisationszyklen im Register.

B. Der clock-select-Algorithmus

Als nächstes geht es darum mögliche fehlerhafte Server, *falsetickers* genannt, auszusortieren. Die Übrigen, hoffentlich richtig arbeitenden, werden *truechimers* genannt.

Die Gesamtzahl der zur Verfügung stehenden Server sei n. Für jeden Server wird ein Intervall $[\theta_i - \lambda_i, \theta_i + \lambda_i]$ errechnet. λ_i ist dabei die *root synchronization distance* $\lambda_i := \frac{\delta}{2} + E_i$. Diese gibt an wie weit θ_i vom wahren Offset – das ist die Differenz zwischen der lokalen Uhr und der realen Zeit – maximal abweicht. Dabei E_i ein Korrekturterm, der unter anderem aus dem geerbten Fehler besteht, den ein Client von einem Server übernimmt. Dieser wird in der Antwort des Servers mit übergeben. Zudem fließen der *reading error*

ρ und *frequency error* φ der lokalen Uhr mit ein. Für die genaue Berechnung sei auf [2] verwiesen.

Dieses Intervall gibt damit den Bereich an, in dem der wahre Offset liegen muss, wenn der Server korrekt arbeitet.

Ein Einsetzen und Umstellen der Formeln (5) und (6) zeigt, dass das Intervall äquivalent zu $[t_3 - t_4 - E_i, t_3 - t_4 + \delta_i + E_i]$ ist. Damit entsprechen die Grenzen dem bei Marzullo in (3) und (4) bestimmten Intervall, wenn man $t_3 = C_j$, $t_4 = C_i$ und $\delta_i = (1 + \varphi_i)\xi_j^i$ setzt und die Korrekturterme beider Algorithmen gegeneinander austauscht. Diese Ausdrücke entsprechen einander, weshalb das zum Vergleich der Algorithmen möglich ist.

In der Bestimmung eines gemeinsamen Intervalls ändert sich die Berechnung gegenüber Marzullo und Owicki, um *falsetickers* auszusortieren. Während bei Marzullo noch ein Intervall gefordert war, in dem alle Antworten liegen, wird hier versucht, ein Intervall zu finden, in dem mindestens die Werte der Hälfte der Server liegen. Auf diese Weise kann auch mit Ausreißern umgegangen werden.

1) Beginne mit der Annahme, dass die Anzahl der *falsetickers* $f = 0$ sei.
2) Berechne
   - $l :=$ der *kleinste* Wert, der in mindestens $n - f$ Intervallen liegt, oder $+\infty$, falls keiner existiert, und
   - $u :=$ der *größte* Wert, der in mindestens $n - f$ Intervallen liegt, oder $-\infty$, falls keiner existiert.
3) Erhöhe f solange, bis $l \leq u$ gilt und mindestens $n - f$ Mittelpunkte θ_i in $[l, u]$ liegen.

Falls $f < \frac{n}{2}$ gilt, verwerfe alle S_i für die θ_i nicht in $[l, u]$ liegt. Diese sind *falsetickers*. Alle anderen bilden die Menge der *truechimers* S.

Falls kein Intervall mit $f < \frac{n}{2}$ gefunden werden konnte, ist eine Synchronisation nicht möglich. Diese Forderung für f ergibt sich aus der angewandten Strategie, eine Übereinkunft aus der Mehrheit der Server zu finden und ist nicht in der theoretischen Voraussetzung für ein funktionierendes System aus Abschnitt II-B4 begründet. Bei $f \geq \frac{n}{2}$ existiert kein Server mehr, dessen Intervall die Mittelpunkte aller anderen noch verbliebenen Server überdeckt.

Es ist ersichtlich, dass durch die Voraussetzung $\theta_i \in [l, u]$ für *truechimers* ein größeres Schnittintervall übrig bleiben kann als im zuvor vorgestellten Algorithmus, selbst wenn alle Server korrekt sind. Unter den übrigen Servern wird im *cluster*-Algorithmus noch eine weitere Auswahl getroffen. Diese beruht auf Heuristiken und soll hier nicht weiter erklärt werden.

C. Der clock-combine-Algorithmus

Aus den Antworten der nach dem *cluster*-Algorithmus noch enthaltenen $S_i \in S$ wird schließlich

$$\Theta := \frac{\sum\limits_{S_i \in S} \theta_i \lambda_i^{-1}}{\sum\limits_{S_i \in S} \lambda_i^{-1}} \tag{7}$$

berechnet. Dies ist der aktuelle Offset, um den die Systemuhr schließlich angeglichen wird. Er soll einen genauen Schätzwert für die Differenz zwischen der realen Zeit und der lokalen Uhr liefern. Statt der einfachen Mitte des Schnittintervalls wie bei Marzullo und Owicki wird im NTP die Systemuhr um das gewichtete Mittel Θ der einzelnen Offsets der Server korrigiert. Der Server mit kleinster *root synchronization distance* – also der scheinbar genaueste Server – wird dabei am meisten berücksichtigt.

Grund dafür ist wieder eine ähnliche Beobachtung wie die, die in Abschnitt IV-A beschrieben ist: Bei kleinerer *root synchronization distance* ist ein Zeitstempel genauer.

D. Abschließende Betrachtung

Um diese Fehler behandeln zu können, setzt das NTP auf strukturelle Redundanz. Zum einen wird mit mehreren Servern synchronisiert. Die Wahrscheinlichkeit, dass alle Server in die gleiche Richtung abweichen, nimmt bei einer größeren Anzahl Servern ab und man hat eine größere Chance, einen genauen Wert zu ermitteln. Zudem werden im Shiftregister des *peer*-Prozess meist mehr Zeitstempel gesammelt, als letztendlich zur Synchronisation herangezogen werden. An mehreren Stellen wird die Verlässlichkeit der empfangenen Zeitstempel bewertet und entsprechend selektiert.

Abschließend soll betrachtet werden, Gewinn an Genauigkeit ein solch komplexes Protokoll gegenüber weitaus simpleren Verfahren hat. David L. Mills hat 1989 die Ergebnisse eines Experiments veröffentlicht, bei dem an über 100 000 Server Anfragen über die Protokolle TIME (RFC 868), ICMP (RFC 867) und NTPv2 (RFC 1119) gesendet wurden [7].

Die beiden ersteren sind sehr simple Protokolle, über die Zeitstempel ausgetauscht werden können. Im TIME-Protokoll wird beispielsweise nur ein einzelner Zeitstempel gesendet, der die vergangenen Sekunden seit dem 01.01.1900, 0:00 Uhr, enthält.

Von 20 000 Servern erhielt man bei mindestens einem Protokoll eine Antwort, die mit einem Lokalen Radioempfänger mit der Genauigkeit einiger Millisekunden verglichen wurde. Dabei ergab sich, dass bei 30% der Antworten die Genauigkeit mit NTP über 30ms, mit TIME bei einer Minute und mit ICMP bei einigen Minuten lag. Die genauen Ergebnisse sind in Abbildung 2 dargestellt. Auf der x-Achse ist ein Offset x in Sekunden dargestellt, auf der y-Achse der Anteil der Antworten in Prozent dargestellt, die um einen Offset $\theta > x$ von der lokalen Referenzuhr abweichen. Die

obere Kurve ist für ICMP, die mittlere für TIME und die Untere für NTPv2.

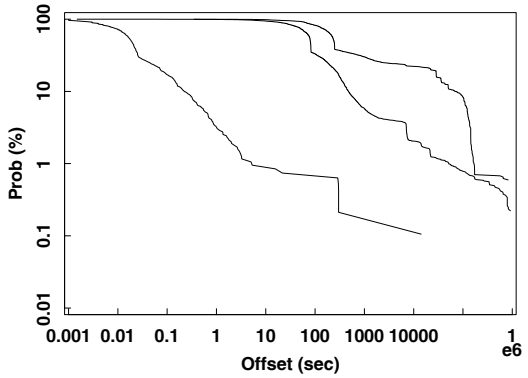

Abbildung 2. Vergleich der Protokolle NTP, ICMP und TIME

Auch wenn diese Daten mit Vorsicht zu genießen sind – Server die auf TIME und ICMP antworten, sind nicht unbedingt als Zeitserver ausgelegt – geben sie dennoch eine Vorstellung davon, welche Genauigkeit schon mit der veralteten Version des NTP erreicht werden konnte.

Leider sind trotz intensiver Recherche keine aktuelleren Ergebnisse vergleichbarer Experimente zu finden.

V. Ausblick

Man sieht nun, dass eine gute Synchronisation von Uhren den Einsatz des Network Time Protocols rechtfertigt. Es gibt durchaus einfachere Lösungsansätze, die auch mit weniger als den vom NTP geforderten Ressourcen an Datenverkehr und Rechenleistung auskommen. Das TIME-Protokoll und ICMP-Zeitstempel sind nur zwei Beispiele. Aber schon die Fehlerbetrachtung zeigt, dass der Einsatz von mehreren Servern und Anfragen für genaue Werte nötig ist. Die Algorithmen von Marzullo und Owicki sowie im Network Time Protocol geben ein Beispiel dafür, wie diese Redundanz effektiv genutzt werden kann. Der Erfolg des NTP im Internet beweist dies wohl.

Das NTP lebt seit der ersten Version von 1985 mittlerweile in der vierten Version und die Nutzer- und Servergemeinde wächst zusehends. Auch wenn die Referenzimplementation ständig erweitert wird, ist eine Version 5 des Protokolls noch nicht in Sicht. Es gibt aber Pläne, die Algorithmen des NTP zukünftig sogar in der Raumfahrt einzusetzen, wo die Anforderungen an ein solches Protokoll in mancher Hinsicht noch weitaus größer als im Internet sind [8].

Literatur

[1] K. Marzullo and S. Owicki, "Maintaining the time in a distributed system," *SIGOPS Oper. Syst. Rev.*, vol. 19, no. 3, pp. 44–54, Jul. 1985. [Online]. Available: http://doi.acm.org/10.1145/850776.850780

[2] D. L. Mills *et al.*, "Network time protocol (NTP)," IETF Request for Comments: RFC5905, Jun. 2010. [Online]. Available: http://tools.ietf.org/html/rfc5905

[3] D. L. Mills, "Modelling and analysis of computer network clocks," University Of Delvare, Tech. Rep. 92-5-2, may 1992.

[4] L. Lamport and P. M. Melliar-Smith, "Byzantine clock synchronization." in *Proceedings of the Symposium on Principles of Distributed Computing*. ACM SIGPLAN/SIGOPS, 1984, pp. 68–74.

[5] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, pp. 382–401, 1982.

[6] D. Dolev, J. Halpern, and H. R. Strong, "On the possibility and impossibility of achieving clock synchronization," in *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, ser. STOC '84. New York, NY, USA: ACM, 1984, pp. 504–511. [Online]. Available: http://doi.acm.org/10.1145/800057.808720

[7] D. L. Mills, "Measured performance of the network time protocol in the internet system," Network Working Group Request for Comments: 1128, Tech. Rep., Oct. 1989.

[8] Network time synchronization research project. Zuletzt abgerufen am 15.07.2012. [Online]. Available: http://www.eecis.udel.edu/~mills/ntp.html#status

Fault tolerant transmission of data within the Internet

Connor Fibich
Department of Computing Science
Oldenburg, Niedersachsen, 26131

Abstract

The Internet is one of the biggest fault tolerant systems. Fault tolerance is mainly based on the availability of redundant paths through the network. This redundancy only guarantees that there is some way between source and destination. The fault tolerance itself is achieved by routing protocols. These protocols use current information about the topology of the network. With this information the protocols are able to determine the desired path through the network. The redundancy of the paths creates disadvantages as well. Due to the dynamics of the network new information will constantly be processed, as well as it has to be ensured that a loop-free path will be calculated.

Keywords: routing protocols, BGP, Dijkstra, fault isolation, fault domain, loop avoidance

1. Introduction

This paper will set its main focus on fault tolerance achieved by routing protocols. In order to understand these protocols a basic set of information about the general network topology and the parts where the redundancy is used is needed.

1.1. Redundancy of the global Internet

Redundancy is one of the main ways to achieve fault tolerance in the Internet. The redundancy begins in the routers itself. Routers are designed to remain operational even if a part of it is faulty. To achieve this fault tolerance a router has a redundant backup of all the vital parts (like the routing processor, cooling, power supply, etc.).[6] If there is an error the backup will take over until the fault is repaired.

Following the principle of 'all roads lead to Rome', there are many redundant paths through the network from source to the destination (route). With these paths redundancy increases the time a network operates without failing (Mean time between failures (MTBF)), since most of the times there is a path to fall back onto when the desired one has a failure. Like navigating though the traffic in an unknown city, having many routes to choose from makes it necessary to have some kind of map or navigation system. Navigating the data through the network is the job of routing protocols.

1.2. Network topology

The global Internet consists of many interconnected Autonomous Systems (AS) (41,434 AS in the routing system in June 2012 [2]). An AS usually consists of a core network and the edges (where the actual entities sending the data are located).

While reliably providing a possible route, redundancy increases the complexity of the network, which has a direct impact on the complexity of the calculations needed to reorganize. Therefore redundancy does not only increase the MTBF but the time the network needs to restore its functionality (Mean time to recover (MTTR)) as well. To decrease the complexity networks are split into smaller segments (fault domains), in which a failure would be handled. While leaving the MTBF of the entire network untouched, the fault domains, with their decreased complexity, decrease the MTBF and the MTTR of the domain, since there are less paths to take into consideration when recalculating the routes. This concept of splitting the network into fault domains is called fault isolation. By isolating the faults in fault domains the complete network (for example an AS) would only need to repair itself if a failure can not be repaired within the domain it is located in.

2. Routing Protocols

There are two types of routing protocols: Link-State-Protocols and Distance- / Path-Vector-Protocols.

2.1. Link-State-Protocols

Link-State-Protocols need a complete view of the network in order to determine the route between source and destination. Because an AS has up to 1,000 routers and a

relatively static layout most AS use Link-State-Protocols as the internal routing protocol.

Every router within a network using Link-State-Protocols floods the information about his connections to every router within the routing domain (the area of the network using the protocol).[1] Once the information about the whole network is gathered Link-State-Protocols use the Dijkstra Algorithm to calculate the shortest path within the network.

2.1.1. Dijkstra Algorithm

The Dijkstra Algorithm is a graph search algorithm which was developed by Edsger Dijkstra in 1956. It is used to determine the shortest paths from a source to all other reachable nodes. The algorithm can be described in the following six steps:

1. Each node gets a distance value (with the standard value 0 for the source node and infinity for all other nodes) and a 'previous' field containing the node from which we reached the current.
2. If there are unvisited nodes choose the one with the lowest distance value.
3. If the distance value of the chosen node is infinity stop the algorithm.
4. Mark the node as visited.
5. Calculate the distance for all neighbor nodes.
6. If the calculated distance is lower then the saved distance value of a neighbor, save the calculated distance as the new distance value and set the current node as 'previous' for that neighbor. Go back to step 2.

The protocol using the Dijkstra Algorithm only needs to backtrack the 'previous' field from the destination node to find the shortest path between source and destination.[5]

2.1.2. Loop avoidance

Link-State-Protocols using the Dijkstra Algorithm do not need to use special rules to achieve loop free routes, since the algorithm was proven to calculate only loop free paths. But even the fact that the algorithm is loop free does not prevent loops from occurring. If a path within the network has a failure, it takes some time until the information about the failure reaches every router within the network. During this time the information of each router may not be the same resulting in path decisions which have loops.

For example there are 4 routers (A, B, C, D) connected as a Ring (A-B, B-C, C-D, A-D). The path between A and D has a much higher weight then the other paths (Example network as seen in figure 1). If a package from A is sent to D it would take the path A-B-C-D. If the connection between C and D now has a failure, C has the information and changes the route to next best available path, which would be C-B-A-D, while B is still unaware of the change and still routes packages to D through C. Therefore B routes a package with D as destination to C and C sends it back to B because of the different set of information. These so called micro loops only occur while the network converges.[1]

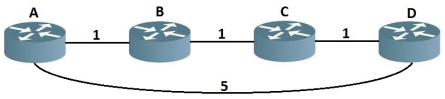

Figure 1. Example network

2.2. Distance- / Path-Vector-Protocols

Unlike Link-State-Protocols Distance- / Path-Vector-Protocols have a aggregated view of the network. These protocols are mainly used to route between the AS. Routers using such protocols gain all their routing information from their direct neighbors.

A router advertises to his neighbors all the networks he knows and a metric as aggregated as possible. The neighbors will then determine if the path advertised is better then the one that might already be in his routing table. If the router changes his routing table he re-advertises the new or changed paths to his neighbors.

2.2.1. Border-Gateway-Protocol

The protocol which is used to determine the routes between the AS is the Border-Gateway-Protocol (BGP). In BGP part of the metric which is advertised with the route is the AS_PATH. The AS_PATH contains the path through the different AS connecting the current AS with the advertised destination. When an AS receives a route it will save the route and the next hop (the router at the edge of the AS, which received the route) in its routing table (the routing table of an AS is saved within every router connecting it to other AS). After a change in the best paths the AS will advertise the changed best paths.

Since BGP routes use a more complex metric, which given the routing-policy will be used to determine the best path, the following (shortened) decision process is needed[3]:

1 The border gateway marked as next hop must be reachable within the AS.

2 Select the path with the highest local preference value. (The local preference is a value influenced by company

policies)

4 Select the route with the shortest AS-path length.

8 Select the path with the easiest to reach next hop.

With each step of the process the possible routes are narrowed down until there is just one route left.

2.2.2. Loop avoidance

Since company policies have a high importance in the decision process, BGP needs a couple of techniques in order to keep the routes determined loop free.

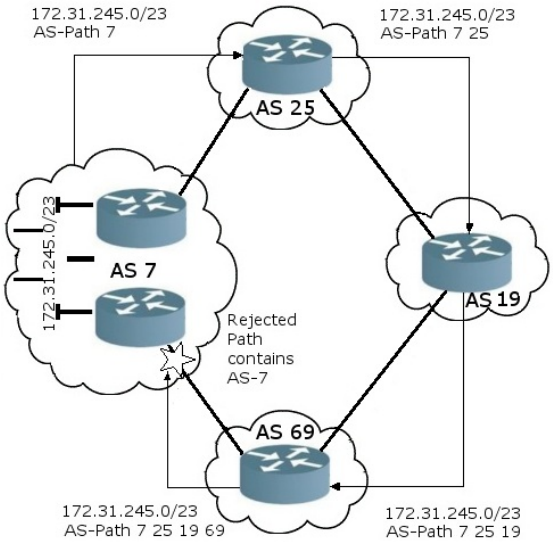

Figure 2. Loop Detection Using AS-Path [7]

One of those techniques is Poison Reverse. Poison Reverse is a mechanism where a BGP router that accepts an advertised route as a new best path, will not re-advertise it to the router this route originates from.

Another technique is the AS-Path-Check. Since BGP route information save the entire path a router will check every route received from an external AS if his AS is already part of the path, in which case it will dismiss the advertised route (as seen in the example in figure 2).[1]

2.2.3. Route-flap dampening

When a connection between two routers is going up and down (flapping) the change would always be sent through the entire routing domain, causing the routing table of the routers within the domain to be constantly recalculated. Route-flap dampening is a technique that decreases the amount of recalculations needed in case that a route is flapping. Each time a route flaps a timer is added. Once a route's timer is above a certain limit it will not be advertised to the neighbors until the timer decreases below the limit again.

With the development of more advanced routing processors today's processors are able to do calculations much faster. This is why a study of RIPE advices that with the increased available resources the advantage of Route-flap damping is not worth the chance that side effects may occur.[4]

3. Summary and Conclusions

The fault tolerance achieved by the redundant routes through the global Internet combined with the routing protocols allow the network to dynamically change, have faults, etc. Each fault will only influence the fault domain it is located in. Depending on the kind of the failure, this influence can reach from not being noticeable to a total failure of the network. But the network in whole will adapt to the change by reorganizing the routes in order to bypass the faulty part. This makes the global network a self stabilizing system.

Acknowledgments

Here I would like to thank Oliver Böhmer for sharing his knowledge about loop free routing.

References

[1] Oliver Böhmer, AS Technical Leader, Cisco Systems GmbH, Personal contact 2012-06-02.

[2] Tony Bates, Philip Smith, Geoff Huston. http://www.cidr-report.org/as2.0/ , Retrieved 2012-06-17.

[3] B. Halabi, S. Halabi, D. McPherson, *Internet Routing Architectures*, ISBN 9781578702336, ICCN 00105166, Cisco Press core series, 2000, Cisco

[4] Philip Smith, Christian Panigl, "RIPE Routing Working Group Recommendations On Route-flap Damping", RIPE, May 2006

[5] http://www-b2.is.tokushima-u.ac.jp/ĩkeda/suuri/dijkstra/Dijkstra.shtml, Retrieved 2012-06-19

[6] http://archive.icann.org/en/tlds/org/applications/register/attachments/h Cisco_7200.pdf, Retrieved 2012-07-14

[7] http://etutorials.org/cert/ccnp+bsci/Part+VI+BGP/Chapter+14.+BGP+ Retrieved 2012-07-14

Decodierung von Faltungscodes in der Kanalcodierung
-
Der Viterbi-Algorithmus

Tobias Schüürhuis
Seminar Fehlertolerante Systeme
Karl von Ossietzky Universität
Oldenburg

Abstract

Ein wichtiger Aspekt in der modernen Nachrichtentechnik ist die möglichst fehlerfreie Rückgewinnung von Nachrichten während einer Datenübertragung. Diese Übertragungen können dabei über Funk- oder auch kabelgebundene Kanäle erfolgen. Basierend auf dem von Shannon 1948 gelegten theoretischen Grundstein, wurden mit der Zeit unterschiedliche Ansätze und Lösungen zur Problematik gestörter Kanäle entwickelt und vorgestellt. Unterschieden wird dabei zwischen der Fehlererkennung und der Fehlerkorrektur. Dabei handelt es sich um unterschiedliche Codierungs- und Decodierungsverfahren, mithilfe derer die Erkennung, Lokalisierung aber auch Korrektur bestimmter Fehlerarten ermöglicht wird. Dieser Beitrag wird sich mit einem wichtigen Bereich der digitalen Nachrichtentechnik, der Kanalcodierung beschäftigen. Schwerpunkt soll dabei insbesondere auf das Gebiet der sogenannten Faltungscodes gesetzt werden. Im weiteren Zusammenhang wird daraufhin eine effiziente Methode zur Decodierung von Faltungscodes mithilfe des Viterbi-Algorithmus' vorgestellt.

keywords: Kanalcodierung; Fehlerkorrektur; Faltungscodes; Viterbi-Algorithmus

1. Einleitung

In der modernen Datenübertragung werden Informationen über unterschiedliche Arten von Kanälen übertragen. Bei dem Begriff des *Kanals* handelt es sich um ein Medium zur Übermittlung von Informationen durch Signale. Dabei kann es sich um Übertragungswege unterschiedlichster Natur handeln. Dazu zählen Kabel und Glasfaserleitungen genauso wie auch drahtlose Verbindungen aus dem Mobilfunk oder der Satellitenkommunikation. Da es in der Praxis keine störungsfreien Kanäle gibt, muss die zu übertragende Information gegen Fehler geschützt werden. Dies erfolgt unter anderem durch die Methoden der im ersten Teil dieser Arbeit eingeführten Kanalcodierung. Nachrichten werden dabei im weitesten Sinne vor der Übertragung über einen Kanal codiert und nach der Übertragung decodiert. Der Begriff der Kanalcodierung selbst umfasst ein breites Spektrum an Codeklassen, die zur Behandlung von unterschiedlichen Fehlerarten eingesetzt werden. Im zweiten Abschnitt der Arbeit wird die Klasse der Faltungscodes, die aufgrund Ihrer vielfachen Anwendung in der Praxis einen hohen Stellenwert in der Kanalcodierung erlangt haben, näher beschrieben. Diese Art von Codes fand erst sehr spät nach ihrer Entdeckung Einzug in die Praxis, da Sie sich zwar sehr einfach implementieren ließen, die Decodierung nach einer Kanalübertragung sich jedoch als sehr komplex erwies. Diesem Problem nahm sich 1967 Viterbi mit dem nach ihm benannten Viterbi-Algorithmus an, durch dessen Hilfe sich eine effiziente Decodierung von Faltungscodierten Nachrichten praktisch realisieren lies. Diesem Algorithmus widmet sich somit der dritte Abschnitt dieser Arbeit. Abschliessend werden einige der häufigsten Einsatzgebiete der eben benannten Themengebiete sowie eine kurze Zusammenfassung gegeben.

1.1. Abgrenzung

Da es sich im Rahmen dieser Arbeit um einen Seminarbeitrag handelt, sollen dem Leser keine neuen Erkenntnisse, sondern vielmehr die nötigen Grundlagen zur Kanalcodierung nähergebracht werden. Eine Vertiefung erfolgt dabei in das Gebiet der Faltungscodes im Zusammenhang mit dem Viterbi-Algorithmus bzw. der Viterbi-Decodierung.

2. Kanalcodierung

Allgemein gefasst bedeutet Codierung eine eindeutige Zuordnung von Zeichen aus einer Quelle zu Zeichen, die durch einen Kanal übertragen werden. Diese Zuordnung soll dabei in der Hinsicht auf Störungen möglichst effektiv sowie Zuverlässig sein. Zur Lösung hat es sich in einem Großteil der Anwendungen als zweckmässig erwiesen, zwischen einer *Quellencodierung* und einer *Kanalcodierung* zu unterscheiden. Die Quellencodierung, die in der Regel einer Kanalcodierung vorangeht, zielt auf eine möglichst redundanzarme Darstellung der Nachricht ab, um Speicher- und Übertragungsaufwand so gering wie möglich zu halten. Man spricht dabei auch von Datenkompression. Diese Art der Codierung soll aber hier nicht weiter behandelt werden. Im Falle der Kanalcodierung wird den zu übertragenden Zeichen Redundanz hinzugefügt, um im Anschluss an die Übertragung durch Kanalstörungen verfälschte Zeichen erkennen und daraufhin korrigieren zu können. Der Bereich der Kanalcodierung umfasst dabei eine große Anzahl an unterschiedlichen Codes. Teil dieser Codes sind auch die in Kapitel 3 vorgestellten Faltungscodes. Im Folgenden werden nach einer kurzen Einordnung der Kanalcodierung in die Kette der Nachrichtenübertragung die Prinzipien der Fehlerkorrektur aufgezeigt. Im Anschluss wird ein Blick auf die dieser Codierungsart angehörigen, unterschiedlichen Codeklassen geworfen [4].

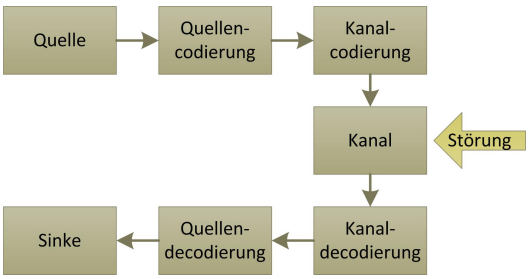

Abb. 1. Übertragungssystem

Abbildung 1 zeigt die Einordnung der Kanalcodierung in den Übertragungsverlauf einer Nachricht. Eine Nachricht, die einer Quelle entspringt, wird demnach im ersten Schritt quellencodiert. Bevor die Nachricht auf den Übertragungskanal gelangt erfolgt die Kanalcodierung. Ab diesem Punkt besteht die Möglichkeit, dass es unter Einfluss von Störungen auf der Übertragungsstrecke zu Verfälschungen der Nachricht kommt. Die Decodierung erfolgt nach der Übertragung, in umgekehrter Reihenfolge zur Codierung, im Empfänger. Im Anschluss wird die decodierte Nachricht ihrem Ziel (Senke) zugefhrt.

2.1. Prinzipien der Fehlerkorrektur

> ”Der Begriff der Redundanz in der Informationstheorie gibt an, wie viel Information im Mittel pro Zeichen in einer Informationsquelle mehrfach vorhanden ist. Eine Informationseinheit ist dann redundant, wenn sie ohne Informationsverlust weggelassen werden kann [5].“

Durch das Hinzufügen von Redundanz zur zu übertragenden Information wird die Korrektur von Fehlern durch Störungen auf dem Kanal ermöglicht. Abbildung 2 zeigt die unterschiedlichen Methoden der Fehlerkorrekturverfahren.

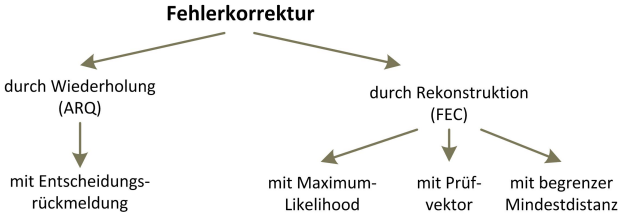

Abb. 2. Fehlerkorrekturverfahren

Im Falle der *Fehlerkorrektur durch Wiederholung* wird nach Feststellung einer fehlerhaft übertragenen Information durch die Auswertung redundanter Kontrollinformationen, der Sender über einen Rückkanal benachrichtigt, um eine nochmalige Übertragung einzuleiten. Die Umsetzung erfolgt mit dem ARQ[automatic repeat request]-Übertragungsverfahren.

Bei der *Fehlerkorrektur durch Rekonstruktion* kann, durch die der Quelle hinzugefügten redundanten Kontrollinformationen, ein Fehler erkannt und zusätzlich lokalisiert werden. Durch die Kenntnis der Fehlerposition ist es dem Kanaldecodierer möglich, den Fehler durch Rekonstruktion zu beseitigen. Die Umsetzung erfolgt mit dem FED[forward error correction]-Übertragungsverfahren. Im Kontext dieser Arbeit interessant ist die Maximum-Likelihood (ML) Rekonstruktions-Methode. Bei dieser Methode erfolgt die Korrektur durch Auswertung bedingter Wahrscheinlichkeiten. Die ML-Methode findet Ihren Einsatz unter anderem in der Decodierung von Faltungscodes.

2.2. Überblick Kanalcodes

Abbildung 3 soll lediglich die Menge an vorhandenen Codes zur Kanalcodierung aufzeigen. Die Einteilung von Kanalcodes kann in zwei große Gruppen erfolgen. Dabei handelt es sich um Blockcodes und die Blockfreien (sequentiellen) Codes.

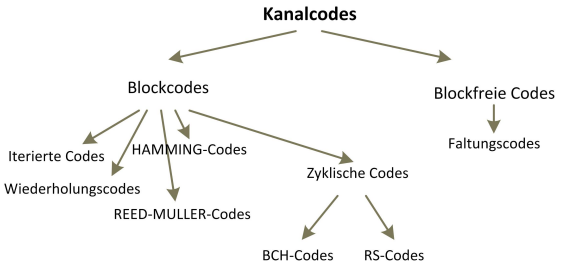

Abb. 3. Baum Kanalcodes

Im Rahmen dieser Arbeit werden die Faltungscodes aus dem Bereich der Blockfreien Codes behandelt.

3. Faltungscodes

Faltungscodes finden Ihren Einsatz in der aus Kapitel 2 bekannten Kanalcodierung. Durch hohe Bitraten, die durch die mathematische Operation der Faltung[1] erreicht werden, kann auch bei stark gestörten Übertragungskanälen ein möglichst fehlerfreier Empfang einer gesendeten Nachricht gewährleistet werden. Hauptmerkmal der Faltungscodes, welche diese von der Gruppe der Blockcodes unterscheidet, ist die nicht auf einer Prüfgleichung beruhende Decodierung. Statt einer algebraischen Decodierung wird dabei mit der optimalen Schätzung einer Empfangsfolge gearbeitet. Aufgrund der einfachen Implementierung sind Faltungscodes in der Kommunikationstechnik weit verbreitet.

Erste Arbeiten zum Gebiet der Faltungscodes publizierte Elias im Jahr 1955. Bei diesen Codiermethoden wurde einer Nachricht Redundanz hinzugefügt, indem die Eingangsbitfolge in einer faltungsähnlichen Operation verarbeitet wurde. Nach seiner Entdeckung wurde dem Konzept der Faltungscodes keine hohe Bedeutung zugeschrieben, da unter anderem effiziente Decodierverfahren fehlten, die auch in der Praxis eingesetzt werden konnten.

Nachdem jedoch in den Jahren 1963 und 1967 Fano und Viterbi Verfahren zur sequentiellen Decodierung entwickelten [6], konnte ein *Maximum-Likelihood-Decoder* [2] erstmals praktisch umgesetzt und dadurch die hinter den Faltungscodes stehende, hohe Leistungsfähigkeit genutzt werden. Daher sind Faltungscodes aus dem Gebiet der modernen nachrichtentechnischen Systeme nicht mehr wegzudenken. Die Struktur von Faltungscodes soll im Folgenden erklärt und durch einige Beispiele unterstützt werden [3], [2].

3.1. Struktur eines Faltungscodierers

Ein *Faltungscodierer* besteht, wie in Abbildung 4 gezeigt, in der praktischen Anwendung aus einem mehrstufigen Schieberegister sowie einem Multiplexer. Dabei werden die vorhergehende, sowie aktuelle Information gespeichert. Eine Eingangsbitfolge wird dabei zu einem Block zusammengefasst und schrittweise dem Schieberegister zugeführt. Nach dieser Eingabe werden in jedem Schritt Ausgabebits duch modulo-2-Addition geeignet gewählter Permutationen der Schieberegisterinhalte gebildet. Die einer Nachricht hinzugefügte Redundanz macht sich in Form eines höheren Taktes am Encoderausgang bemerkbar. Die Codierung geschieht dabei durch Faltung der Eingangsfolgen mit den Impulsantworten [3]

Im Folgenden soll an einem Beispiel die grundsätzliche Struktur der Codierung eines Bitstroms durch einen Faltungscodierer gezeigt werden.

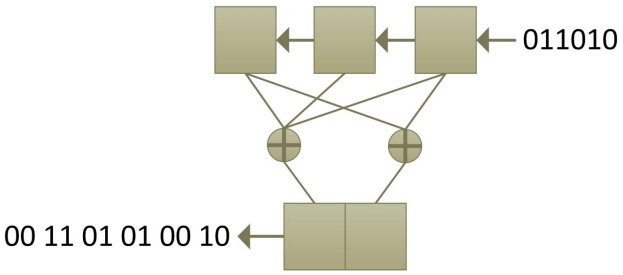

Abb. 4. Faltungscodierer der Rate = 1/2

Bei der Codierung eines Faltungscodes wird die Datenbitfolge, in diesem Fall 011010 schrittweise in ein Schieberegister mit $L = 3$ Zellen gegeben. In jedem Codierschritt werden $b = 1$ Datenbits in das Schieberegister weitergegeben. Im Anschluss werden $n = 2$ Ausgabebits durch die modulo-2-Addition (XOR / Exklusiv ODER) der Inhalte geeignet gewählter Zellen des Schieberegisters gebildet. Es resultiert eine Coderate $R = \frac{b}{n}$, welche das Verhältnis zwischen der Anzahl der Ein- und Ausgabebits in der Schieberegisterschaltung beschreibt.

3.2. Darstellungsformen von Faltungscodes

Neben der Darstellung mit Hilfe von Schieberegistern existieren drei weitere Methoden zur Visualisierung von

[1] Durch Summierung oder Integration zweier Funktionen wird eine weitere Funktion erzeugt.

[2] Maximum-Likelihood-Methode - Bezeichnet ein Schätzverfahren, bei dem derjenige Parameter als Schätzung gewählt wird, gemäßdessen Verteilung die Realisierung der beobachteten Daten am plausiblesten erscheint.

Faltungscodes, die im Folgenden kurz vorgestellt werden. Dabei besitzt jede Art der Darstellung Vor- und Nachteile. Alle Beispiele verwenden zur besseren Übersicht den in Abbildung 4 eingeführten Faltungscode [3].

Codebaum

In dieser Darstellung können erzeugte Faltungscodes als Baum (siehe Abbildung 5) dargestellt werden. Dabei besitzt jeder Knoten innerhalb einer Ebene zwei Äste zu den Knoten der nächsten Ebene. Die Ein- und Ausgabebits sind an den Ästen zu finden. Der Pfad der Eingangsbitfolge wird dabei markiert, somit ist die gesammte Information eines Faltungscodes im Codebaum wiederzufinden.

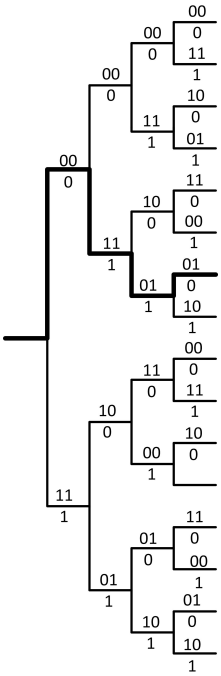

Abb. 5. Codebaum mit R = 1/2

Da jedoch die Anzahl der möglichen Pfade exponentiell mit der Sequenzlänge wächst, stösst man mit dieser Art der Darstellung schnell an seine räumlichen Grenzen.

Zustandsdiagramm

Ist ein Faltungscode als endlicher Automat beschrieben, so handelt es sich um ein Zustandsdiagramm. Dabei sind die von jedem gültigen Knoten ausgehenden Pfeile gleichbedeutend mit den verschiedenen Eingabemöglichkeiten im Schieberegister aus Abbildung 4. Nachteil an dieser Art der Darstellung ist der zeitliche Bezug, der im Gegensatz zur Binärbaumdarstellung in der Entwicklung des Faltungscodes verloren geht. Abbildung 6 zeigt das zugehörige Zustandsdiagramm.

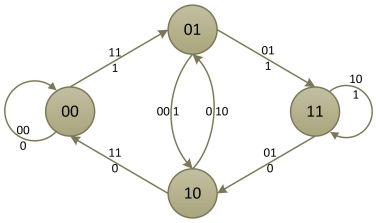

Abb. 6. Zustandsdiagramm mit R = 1/2

Trellisdiagramm

Beim Trellisdiagramm in Abbildung 7 handelt es sich um eine Art Kombination aus den Methoden des Codebaums und des Zustandsdiagramms. Das Trellisdiagramm zeigt alle Zustandsübergänge die bei der Codierung, ausgehend von einem Nullzustand, auftreten können. Ein Pfeil Symbolisiert dabei einen Zustandsübergang, wobei dieser jeweils mit der zugehörigen Eingabe (unten) sowie Ausgabe (oben) des Codierers beschriftet ist.

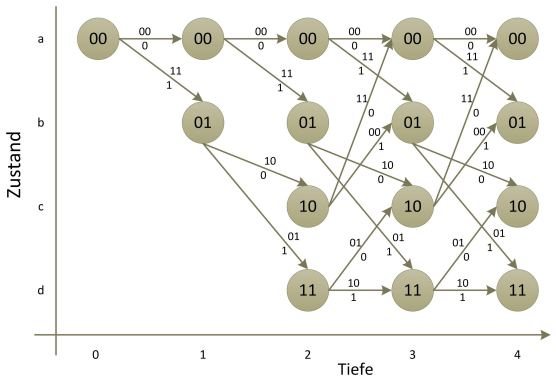

Abb. 7. Trellisdiagramm mit R = 1/2

In jedem weiteren Schritt werden alle der möglichen Folgezustände im Schieberegister betrachtet. Das Trellisdiagramm ist insbesondere für die Entwicklung von Decodieralgorithmen geeignet, da es eine praktikable Darstellung von Faltungscodes liefert.

3.3. Decodierung von Faltungscodes

Bei der Decodierung von Faltungscodes wird nicht ein einzelnes empfangenes Symbol, sondern eine gesamte Sequenz bearbeitet. Dabei werden alle möglichen Sendesym-

bolfolgen mit der jeweiligen aktuellen Empfangsfolge verglichen. Ausgewählt wird die Symbolfolge, die mit höchster Wahrscheinlichkeit gesendet wurde [3], [4]. Allerdings ist der resultierende Verarbeitungsaufwand für einen solchen Maximum-Likelihood-Empfänger in der oben beschriebenen Realisierung ungemein groß, weil alle möglichen Sendesymbolfolgen betrachtet und überprüft werden müssen. Erst durch die von Viterbi veröffentlichten Arbeiten ist ein rechenökonomisches Verfahren zur Realisierung eines ML-Empfängers bekannt geworden. In diesem Zusammenhang ist es wichtig, dass zwar alle möglichen Sendesymbolfolgen überprüft werden, aber bereits frühzeitig erkannt und entschieden wird, welche Folgen bis zum Schluss der Auswertung noch berücksichtigt werden müssen. Es erfolgt also eine Reduzierung der nötigen Arbeitsschritte.

Die Decodierung von Faltungscodes kann also mit dem im folgenden Kapitel 4 beschriebenen Viterbi-Algorithmus, nach dem obigen Maximum-Likelihood-Kriterium, durchgeführt werden. Der Decodieraufwand steigt dabei exponentiell mit der Länge des Schieberegisters L, da die Anzahl der Zustände im Trellisdiagramm wesentlich zunimmt. In der Praxis führen aber Schieberegisterlängen bis $L = 7$ noch zu einem vertretbaren Verarbeitungsaufwand und werden praktisch verwendet.

3.4. Hard/Soft-Decision Decodierung

Zur Entscheidung über die Sendesymbolfolge mit der höchsten Auftrittswahrscheinlichkeit werden abhängig von den erhaltenen Bitfolgen zwei unterschiedliche Decodierungs-Ansätze verfolgt [3], [4].

Eine Hard-Decision Decodierung ist erforderlich, wenn eine Folge von binären (0 oder 1) Eingangssequenzen vorliegt. Soft-Decision Decodierung wird eingesetzt, wenn dem Decoder eine kontinuierliche Folge von Bits nach vorangegangener Quantisierung[3] zugeführt wird.

4. Viterbi-Decodierung

Die Decodierung mit dem Viterbi-Algorithmus ist im Vergleich zur Decodierung von Blockcodes, sowie mit Hilfe sogenannter Syndromtabellen, wesentlich aufwendiger. Die deutlich höhere Leistungsfähigkeit, d.h. bei gleicher Redundanz können mehr Fehler korrigiert werden, machen diesen Algorithmus jedoch trotzdem äusserst interessant. Wie bereits in Kapitel 3.3 erwähnt, basiert der Viterbi-Algorithmus bzw. das Viterbi-Decodierverfahren von Faltungscodes auf der optimalen Schätzung einer Empfangsfolge. Die Realisierung erfolgt dabei mit Hilfe eines Maximum-Likelihood-Decoders. Im Prinzip müssen bei dieser Art der Decodierung sämtliche mögliche Sendefolgen überprüft werden, um eine Auswahl bezüglich der Empfangsfolge treffen zu können. Diese Überprüfung erfolgt durch die Berechnung der quadratischen Abstände, der Bestimmung einer minimalen Hamming-Distanz[4] oder auch der maximalen Wahrscheinlichkeitsdichte. Dieses Vorgehen ist jedoch aufgrund der großen Anzahl an möglichen Sendefolgen nicht praktisch anwendbar. Der Verarbeitungsaufwand eines ML-Decoders wächst in diesem Fall exponentiell mit der Sequenzlänge [1], [3].

Als Basis des Viterbi-Algorithmus dient immer ein Trellisdiagramm. Zur Beschreibung werden im Folgenden wiederum der bereits aus Kapitel 3 bekannte Faltungscode sowie das in Abbildung 7 dargestellte Trellisdiagramm genutzt. Das Beispiel zur Decodierung soll der Einfachheit halber für eine Hard-Decision Decodierung (siehe Kapitel 3.4) mit der darin enthaltenen Hamming-Abstandsfunktion entwickelt werden. Dazu betrachten wir im ersten Fall eine fehlerfreie Übertragung.

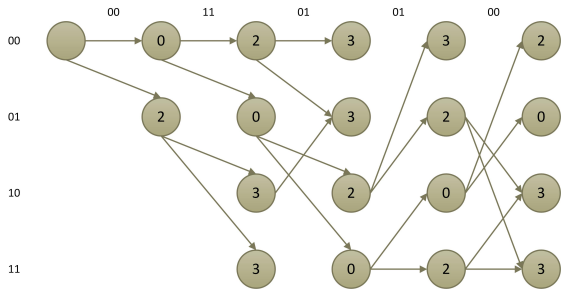

Abb. 8. Trellisdiagramm zur Decodierung bei fehlerfreier Übertragung

Im Gegensatz zur Codierung aus Abbildung 7 werden bei der Decodierung des Faltungscodes in den Knoten des Trellisdiagramms aus Abbildung 8 die Werte der Abstandsfunktion (auch Pfadmetrik), d.h. die jeweils akkumulierten Hamming- bzw quadratischen Abstände zwischen der Empfangs- und möglichen Sendefolge notiert. Es wird also bei einer Hard-Decision im Faltungsdecoder die Anzahl der Bitfehler zwischen den empfangenen Binärfolgen und den mit dem jeweiligen Faltungscode möglichen Sendesymbolfolgen berechnet.

Ab der dritten Stufe im Trellisdiagramm gibt es in dem beobachteten Beispiel bereits zwei unterschiedliche Pfade, die in einen Knoten münden, deren akkumulierter Abstand sich aber in der Metrik unterscheidet. Nach dem hier diskutierten ML-Ansatz wird die wahrscheinlichste Sendefolge

[3] Bei einer Quantisierung erfolgt die Einstufung eines analogen Signals in abzhlbar vielen Mglichkeiten. Das Signal wird somit wertdiskret.

[4] Als Hamming-Distanz wird die Anzahl unterschiedlicher stellen zweier Codewörter bezeichnet.
Bsp.: 01100 und 01000 ergibt eine Hamming-Distanz von 1.

gesucht, d.h. der Pfad mit dem kleinsten bisher berechneten Abstand wird weiterverfolgt und sämtliche anderen Pfade können demnach eliminiert werden, um Rechenzeit zu sparen. Trifft der Algorithmus auf zwei gleich akkumulierte Abstände, entscheidet das Los, welcher Pfad nun weiterverfolgt wird.

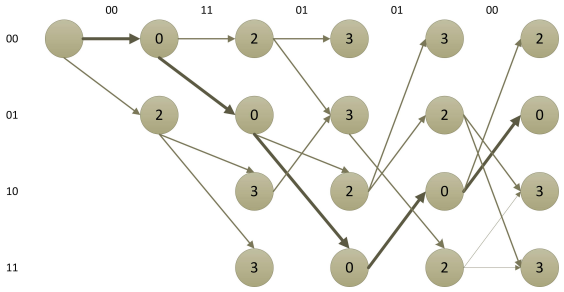

Abb. 9. Zurückverfolgen des wahrscheinlichsten Pfades

Im obigen Trellisdiagramm sind jeweils nur die nicht eliminierten Pfade aufgezeigt. Die in den einzelnen Knoten angegebenen Zahlen beschreiben den bis zu dieser Stelle akkumulierten und jeweils minimalen Hamming-Abstand. In dieser Stufe ist der kleinste berechnete Abstand ausschlaggebend für die Entscheidung des Folgezustands. Dadurch wird die Zurückverfolgung des wahrscheinlichsten Pfades ermöglicht.

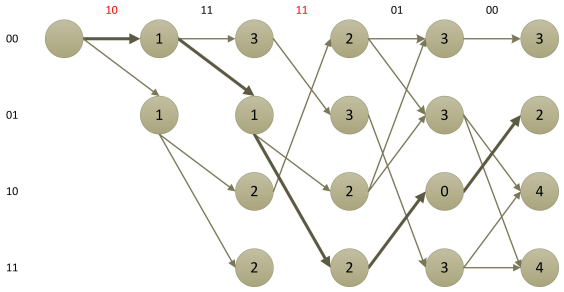

Abb. 10. Zurückverfolgen des wahrscheinlichsten Pfades / mit Übertragungsfehlern

Abbildung 10 zeigt, dass, auch bei einer Verfälschung des Codewortes bei der Übertragung, die korrekte Sendesequenz berechnet werden kann. Dies erfolgt durch das Zurückverfolgen des wahrscheinlichsten Pfades. Wie im Beispiel zuvor wird die Sendesequenz dabei aus der Abstandsfunktion der geringsten akkumulierten Hamming-Distanz wiederhergestellt.

5. Einsatz

Einsatz findet diese Methode der Nachrichtencodierung in weitreichenden Gebieten. Dazu zählen unter anderem nach dem GSM Standard arbeitende Handys aus der Mobilkommunikation, Satellitenfunk oder auch Verfahren in der Spracherkennung. Weitere Anwendungen gibt es in zahlreichen kabelgebundenen Systemen oder bei der Herstellung fehlertoleranter Speichermedien.

6. Zusammenfassung

Nachdem sich der Artikel im ersten Abschnitt mit der Kanalcodierung im allgemeinen beschäftigt hat, wurde das neben den Blockcodes wichtigste Gebiet der Faltungscodes hervorgehoben. Im komplexen Thema der Co- und Decodierung dieser Codeklasse, wurde anschlieend die Möglichkeit des Maximum-Likelihood Ansatzes am Beispiel eines Viterbi-Decoders aufgezeigt.

Zusammenfassend ist zu sagen, dass die Kanalcodierung und seit dem möglichen Einsatz des Viterbi-Decoders in der Praxis, speziell die Gruppe der Faltungscodes einen hohen Stellenwert in der Nachrichtentechnik eingenommen haben. Weiterhin existieren, auf Basis der vorgestellten Codes, bereits noch effizientere Methoden der Kanalcodierung, zu denen z.B. die Verkettung mehrerer Faltungscodes oder auch die Verkettung von Faltungscodes mit Anderen, in Abbildung 3 dargestellten Codeklassen zählen.

Literatur

[1] M. Werner, “Information und Codierung - Grundlagen und Anwendungen,” *Vieweg+Teubner*, Ausg.1, 2002.

[2] P. Rechenberg, “Informatik-Handbuch,” *Vieweg+Teubner*, Ausg.4, 2006.

[3] H. Rohling, “Einführung in die Informations- und Codierungstheorie,” *Teubner*, Aug.2, 1995.

[4] H. Klimant, R. Piotraschke, D. Schönfeld “Informations- und Codierungstheorie,” *Teubner*, Ausg.2, 2003.

[5] Wikipedia, “http://de.wikipedia.org/wiki/Redundanz,” *Wikipedia*, zuletzt abgerufenn am 20.06.2012.

[6] A. Viterbi, “Error bounds for convolutional codes and an asymptotically optimum decoding algorithm,” *Transactions on Information Theory. 13*, 1967.

QR-Code: Fehlererkennung und Fehlerkorrektur

Marius Hacker
Carl von Ossietzky Universität Oldenburg
Oldenburg

Abstract

Ursprünglich wurden QR-Codes(Quick Response Codes) nur im Logistikbereich eingesetzt und ersetzten dort den Barcode. Heute werden sie immer öfter auch in der Werbung eingesetzt. Da QR-Codes mit einer Kamera gelesen werden, ist das Einlesen sehr anfällig gegenüber Störungen (z.B. Verschmutzungen). Um die Nachricht trotzdem richtig einlesen zu können, wird die Reed-Solomon-Kodierung verwendet. Dieses Kodierungsverfahren kann Fehler erkennen und korrigieren. Dieses Paper behandelt die Reed-Solomon-Codes und ihre Anwendung bei QR-Codes.

Schlüsselwörter: QR-Code, Reed-Solomon-Codes, Fehlerkorrektur, Fehlererkennung

1. Einleitung

An der Bushaltestelle, auf Internetseiten, in Prospekten, sogar auf Grabsteinen sind sie mittlerweile zu finden: Quick Response Codes. Durch ihr ansprechendes Aussehen haben sie ihren Platz in der Werbung gefunden. Ein Smartphone kann den Code direkt auslesen und erspart das eintippen der Adresse einer Internetseite.

Im Folgenden soll gezeigt werden was hinter den schwarzen und weißen Pixeln eines QR-Codes steckt. Es wird erklärt, wie ein QR-Code aufgebaut ist und welche Verfahren notwendig sind, um ihn zu erstellen. Auf die zur Fehlerkorrektur verwendeten Reed-Solomon Codes soll besonders eingegangen werden. Dazu wird das Kodieren und Dekodieren an einem vereinfachten Beispiel ausführlich erklärt. Die Kodierung und Dekodierung von RS-Codes ist im Vergleich zu anderen Codierungsverfahren (z.B. Hammingcodes) sehr rechenintensiv, dafür haben RS-Codes aber sehr gute Fehlerkorrektureigenschaften.

2 Vom Barcode zum QR-Code

Die Idee vom Barcode ist aus dem Jahr 1948 und stammt von N.J. Woodland und B. Silver. Als Vorlage diente der Morsecode. Die Punkte des Morsecodes wurden nach oben verlängert, um das maschinelle Einlesen zu erleichtern. Da es lange keine günstigen Lesegeräte gab, wurde der Barcode erst 1969 kommerziell genutzt. Erstmals wurde er von einer Motorenfabrik und einem Logistikunternehmen benutzt. [2]

Hauptsächlich wurde der Barcode für die Warenkennzeichnung im Logistikbereich eingesetzt. Die Begrenzung auf 20 Zeichen beim normalen Barcode führten zur Weiterentwicklung dieses 1-D Codes hin zum 2-D Code. Die Idee bei 2-D Codes ist, wie der Name vermuten lässt, die Daten zweidimensional zu speichern. Dadurch können deutlich mehr Informationen gespeichert werden.

Der QR-Code ist ein solcher 2-D Code. [6]

3 QR-Codes

QR-Codes gehören zu der Gruppe der Matrix-Codes und wurden 1994 von der Firma Denso entwickelt. Denso setzte die Codes zur Verfolgung der Bauteile während der Produktion ein. [6]

Die schwarzen und weißen Pixel des QR-Codes werden Module genannt. Ein dunkles Modul steht für eine binäre 1, ein helles für eine binäre 0. Je nach Zeichensatz werden die binären Zahlen unterschiedlichen Zeichen zugeordnet. Es gibt vier verschiedene Zeichensätze. Es gibt einen numerischen (Zahlen 0-9), einen alphanumerischen (Zahlen 0-9, Großbuchstaben A-Z und space, $, %, *, +, -, ., /, :), einen für 8-bit Daten und einen Zeichensatz für Kanji Zeichen (chinesische Schriftzeichen). Beim numerischen Zeichensatz kann der Code bis zu 7089 Zeichen speichern. Beim alphanumerischen Code sind es nur 4296. [3]

Die Version gibt an, wie groß der Code ist. Version 1 ist 21x21 Module groß. Jede Version hat an der Seite 4 Module mehr als sein Vorgänger. Version 40 ist die höchste Version, der Code misst dann 177x177 Module. Jede Version kann eine von vier Stufen der Fehlerkorrektur benutzen. Die Stufen werden mit L, M, Q und H (Low, Medium, Quartile und High) bezeichnet. Die Stufe L kann (je nach Version) etwa 7% der Daten wieder herstellen. Bei der höchsten Stufe H sind es etwa 30%. [3] [6]

Abbildung 1. Aufbau eines QR-Codes

3.1 Aufbau des QR-Codes

Jeder QR-Code hat drei Hauptpositionsmarker (siehe Abblidung 1). Diese Marker dienen der Bestimmung der Lage. Die Lagebestimmung des QR-Codes ist wichtig, da er zum Einlesen auch von der Seite fotografiert werden kann. Die Hauptpositionsmarker können aus verschiedenen Winkeln leicht erkannt werden. Je nach Größe besitzt ein QR-Code kein, ein, oder mehrere Ausrichtungsmuster. Bei größeren Codes erleichtern diese die Lagebestimmung. Die drei Hauptpositionsmarker sind mit einem Zeitraster verbunden. Es besteht aus sich abwechselnden schwarzen und weißen Modulen und erleichtert ebenfalls das Auslesen. Kann der Code mithilfe der eben beschriebenen Merkmale erkannt werden, so werden als nächstes die Versions- und Formatinformationen ausgelesen. Mit diesen Informationen können die restlichen Module (graue Module in Abbildung 1) ausgelesen werden, die aus Daten- und Fehlerkorrekturblöcken bestehen. [3]

4 Fehlerkorrektur

Beim Einlesen eines QR-Codes können diverse Probleme auftreten. Der Code kann beispielsweise verschmutzt sein oder die Kamera, mit der der Code gelesen wird, funktioniert nicht richtig.

Wir werden im Folgenden zwischen Ausfällen und Fehlern unterscheiden. Bei einem Ausfall kann ein Modul nicht gelesen werden, uns ist aber die Position des Moduls bekannt. Ein Ausfall würde dann auftreten, wenn die Kamera statt eines schwarzen oder weißen Moduls z.B. ein grünes Modul erkennt. Dieses Modul kann keinem Binärwert zugeordnet werden.

Bei Fehlern ist die Position des fehlerhaften Moduls nicht bekannt. Ein Fehler würde auftreten, wenn ein Modul durch Verschmutzung des QR-Codes fälschlicherweise als schwarz anstatt weiß, oder umgekehrt, eingelesen wird.

Die Anzahl der Fehler und Ausfälle, die korrigiert werden können, wird mit folgender Formel berechnet:

$$e + 2t \leq d - p \tag{1}$$

- e = Anzahl der Ausfälle
- t = Anzahl der Fehler
- d = Anzahl der Fehlerkorrekturwörter
- p = Anzahl der Fehlerkorrekturwörter für die Fehlererkennung

Betrachtet man die Formel fällt auf, dass doppelt so viele Ausfälle wie Fehler korrigiert werden können. Bei Ausfällen sind die Module, die ausgefallen sind, bekannt. Dadurch lassen sich Ausfälle leichter korrigieren. Die Anzahl der Fehlerkorrekturwörter d ist für jede Version und jedes Fehlerkorrekturlevel festgelegt (siehe Tabelle S. 35ff in [3]). In der Formel steht p für die Anzahl der Fehlerkorrekturwörter, ab der nur noch eine Fehlererkennung möglich ist. [3]

Betrachtet man als Beispiel einen QR-Code der Version 8 mit der Fehlerkorrekturstufe L: Dieser Code hat 48 Fehlerkorrekturwörter und er kann bis zu 12 Fehler korrigieren (siehe Tabelle 36 in [3]). Eingesetzt in die Formel 1 ergibt sich:

$$0 + 2 \cdot 12 = 48 - 24 \tag{2}$$

Das bedeutet solange mehr als 24 fehlerfreie Fehlerkorrekturwörter gelesen werden können, können 12 Fehler oder 24 Ausfälle korrigiert werden.

5 Reed-Solomon Codes

Die Fehlerkorrektur bei QR-Codes basiert auf Reed-Solomon Codes. Bei RS-Codes handelt es sich um eine 1960 von I. S. Reed und G. Solomon gefundene Klasse von Codes, die leicht zu dekodieren ist und trotzdem eine gute Fehlererkennung zulässt. Da RS-Codes sehr gut Burstfehler (Fehler die hintereinander liegen) erkennen , werden sie z.B. auch bei Audio-CDs eingesetzt. [1]

Da die Rechnung mit RS-Codes bei QR-Codes recht schwer nachzuvollziehen ist, soll im Folgenden die Rechnung vereinfacht dargestellt werden (siehe Script von Johannes Blömer [4]).

Im folgenden Beispiel werden wir eine Nachricht mit drei Stellen und fünf Zeichen kodieren. Das Fehlerkorrekturwort wird die Länge fünf haben. Kurz wird ein solcher RS-Code als RS(5,3,5) geschrieben. Die erste Fünf steht für

die Anzahl der Zeichen, die Drei für die Nachrichtenlänge und die letzte Fünf für die Länge des Fehlerkorrekturwortes.

5.1 Fehlerkorrekturwort bestimmen

Bei einem RS(5,3,5) Code werden fünf Zeichen benutzt. Angenommen diese fünf Zeichen sind $0, 1, 2, 3, 4$. Alle Rechnungen werden damit in einem fünfelementigen Körper durchgeführt. Die Rechnungen werden also mit Modulo-5 berechnet. Die Nachricht darf maximal drei Zeichen lang sein und soll hier 124 lauten. Um den Code zu berechnen setzt man die Stellen der Nachricht als Koeffizienten eines Polynoms ein. Dieses Erzeugerpolynom nennen wir $a(x)$. Das Erzeugerpolynom zur Nachricht 124 ist $a(x) = 1 \cdot x^2 + 2 \cdot x + 4$. Das Codeword wird gebildet, indem die Werte des Polynoms an fünf Stellen ausgerechnet werden. Diese Stellen heien Stützstellen. Die Stützstellen sind beispielsweise $0, 1, 2, 3, 4$. Die dritte Stelle ergibt sich dann aus $c_3 = a(2) = 1 \cdot 2^2 + 2 \cdot 2 + 4 (mod 5) = 2$. Rechnet man die Werte fr die restlichen Stützstellen aus, so erhält man das Codewort $c = (4, 2, 1, 4, 3)$. [4]

5.2 Korrektur bei Ausfällen

Um bei einem Ausfall die Nachricht decodieren zu können, müssen mindestens so viele Zeichen übertragen worden sein, wie die Nachricht lange ist. Angenommen das empfangene Codewort ist $c_E = (4, 2, \_, \_, 3)$. Die dritte und vierte Stelle wurden nicht übertragen. Zur Rekonstruktion der Nachricht muss eine Lagrange-Interpolation [5] durchgeführt werden.

Das Erzeugerpolynom $a(x)$ kann mit der Formel 3 bestimmt werden.

$$a(x) = \sum_{i=1}^{m} \frac{a(u_i)}{g_i(u_i)} \cdot g_i(x) \tag{3}$$

In Formel 3 wird mit m die länge der Nachricht bezeichnet und $a(u_i)$ sind die entsprechenden Stellen des empfangenen Codewortes $4, 2, 3$. Zuerst muss $g_i(x)$ bestimmt werden. Dazu werden die Stützstellen u_i in die Formel 4 eingesetzt.

$$g_i(x) = \prod_{j=1, j \neq i}^{m} (x - u_j), i = 1, \ldots, m \tag{4}$$

In diesem Beispiel erhält man:

$$g_1(x) = (x-1)(x-4) = x^2 + 4 \tag{5}$$
$$g_2(x) = x(x-4) = x^2 + 1x \tag{6}$$
$$g_3(x) = x(x-1) = x^2 + 4x \tag{7}$$

$g_1(u_1) = 4$	$g_1(u_1)^{-1} = 4$
$g_2(u_2) = 2$	$g_2(u_2)^{-1} = 3$
$g_3(u_3) = 2$	$g_3(u_3)^{-1} = 3$

Abbildung 2. Zwischenergebnis

Werden in diese Polynome die Stützstellen eingesetzt, können die Werte für $g_i(u_i)$ und danach die Inversen bestimmt werden. Anschließend berechnet man $a(u_i) \cdot g_i(u_i)^{-1}$.

$$a(u_1) g_1(u_1)^{-1} = 1 \tag{8}$$
$$a(u_2) g_2(u_2)^{-1} = 1 \tag{9}$$
$$a(u_3) g_3(u_3)^{-1} = 4 \tag{10}$$

Nun können die Werte in die Formel 3 eingesetzt werden und das Erzeugerpolynom berechnet werden.

$$a(x) = (x^2 + 4) + (x^2 + 1x) + 4 \cdot (x^2 + 4x) = x^2 + 2x + 4 \tag{11}$$

Die Nachricht kann von den Koeffizienten des Erzeugerpolynoms abgelesen werden. Die Nachricht lautet $1, 2, 4$.

5.3 Korrektur bei Fehlern

Angenommen die erhaltene Nachricht lautet $c = (4, 2, 3, 4, 3)$. An der dritten Stelle wurde statt einer Eins eine Drei übertragen. Um das Erzeugerpolynom zu bekommen, muss ein Gleichungssystem aufstellen und gelöst. Dazu benötigt man die beiden Polynome:

$f(x) = f_1 \cdot x + f_0$ **und** $g(x) = g_3 \cdot x^3 + g_2 \cdot x^2 + g_1 \cdot x + g_3$

Der Grad der beiden Funktionen hängt vom RS-Code ab. Bei unserem RS(5,3,5) hat f den Grad eins und g den Grad drei. Aus f und g erzeugen wir nun $p(x, y)$:

$$p(x, y) = f_1 \cdot xy + f_0 \cdot y + g_3 \cdot x^3 + g_2 \cdot x^2 + g_1 \cdot x + g_3$$

Die Stützstellen mit den zugehörigen Y-Werten sind in 12 dargestellt.

$$(0, 4), (1, 2), (2, 3), (3, 4), (4, 3) \tag{12}$$

Setzen wir diese in die Gleichung ein, so erhalten wir ein Gleichungssystem (13 bis 17).

$$4f_0 + g_0 = 0 \tag{13}$$
$$2f_1 + 3f_0 + g_3 + g_2 + g_1 = 0 \tag{14}$$
$$f_1 + 4f_0 + 3g_3 + 4g_2 + 2g_1 = 0 \tag{15}$$
$$2f_1 + 2g_3 + 4g_2 + 3g_1 = 0 \tag{16}$$
$$2f_1 + 4f_0 + 4g_3 + g_2 + 4g_1 = 0 \tag{17}$$

Aus Gleichung 13 folgt $g_0 = f_0$. Als nächstes setzt man g_0 in das Gleichungssystem ein und formt es mit dem Gaußalgorithmus um (18 bis 21).

$$4f_0 + 2g_3 = 0 \quad (18)$$

$$f_1 + 4f_0 + 3g_3 + 4g_2 + 2g_1 = 0 \quad (19)$$

$$3f_1 + 4f_0 + 3g_2 = 0 \quad (20)$$

$$2f_1 + f_0 = 0 \quad (21)$$

Damit wir das Gleichungssystem eindeutig lösen können, setzten wir für f_1 die erste Stützstelle $f_1 = 4$ ein. Löst man nun das Gleichungssystem, erhält man $f_1 = 4$, $f_0 = 2$, $g_3 = 1$, $g_2 = 0$, $g_1 = 0$, $g_0 = 2$ und damit $f(x) = 4x + 2$ und $g(x) = x^3 + 2$. Um auf unsere Erzeugerfunktion zu kommen, müssen wir nun eine Polynomdivision von $g(x)/f(x)$ durchführen und anschließend unser Ergebnis negieren. Die Polynomdivision ergibt $4x^2 + 3x + 1$ was negiert unser Erzeugerpolynom $x^2 + 2x + 4$ und damit unsere Nachricht $c = (1, 2, 4)$ ergibt. Gibt es bei der Polynomdivision einen Rest, so kann die Nachricht nicht korrigiert werden, da zu viele Fehler aufgetreten sind.

6 Maskierung

Nachdem nun die Daten und Fehlerkorrekturwörter erstellt wurden, müssen diese maskiert werden. Das Maskieren ist notwendig, um zu verhindern, dass die Daten zufällig Positionsmarker oder Ausrichtungsmuster ergeben. Es gibt acht verschiedene Funktionen, die jeweils eine Maske generieren. Eine dieser Funktionen ist beispielsweise $(i + j) \mod 2 = 0$. Diese Funktion liefert ein Schachbrettmuster. Bevor die Maske per XOR-Operation auf die Daten angewandt wird, muss zuerst die beste Maske ausgewählt werden. Dazu wird jede Maske bewertet. Ein Bewertungskriterium ist beispielsweise wie viele gleichfarbige Module in einer Zeile oder Spalte sind. Die Maske die am besten abschneidet wird benutzt. [3]

7 QR-Code zusammenfügen

Nachdem die Daten maskiert wurden, muss der QR-Code noch mit den Hauptpositionsmarkern und wenn nötig mit den Ausrichtungsmustern versehen werden. Wie in Abbildung 1 zu sehen ist, werden im Code auch noch Versions- und Formatinformationen hinzugefügt. Die Formatinformationen beinhalten das Maskierungsmuster. Die Versionsinformationen die Version des Codes und die Stärke der Fehlerkorrektur. Jetzt ist der QR-Code fertig. [3]

8. Zusammenfassung

Die QR-Codes werden vielseitig eingesetzt. Neben der Warenkennzeichnung werden sie auch in der Werbung benutzt. Sie wurden von der Firma Denso entwickelt, die mit ihm den Barcode als Warenkennzeichnung ablösten.

Der auch als Matrix und 2-D Code bezeichnete QR-Code kann bis zu 4296 alphanumerische Zeichen speichern. Auf der höchsten Fehlerkorrekturstufe können etwa 30 % der Daten wieder hergestellt werden.

Bei der Fehlerkorrektur wird unterschieden zwischen Ausfällen und Fehlern. Bei Ausfällen wissen wir welches Modul nicht gelesen werden kann. Bei Fehlern hingegen können wir nicht sagen wo dieser aufgetreten ist. Aus diesem Grund ist es leichter Ausfälle zu beheben.

Die Fehlerkorrektur eines QR-Codes benutzt Reed-Solomon Codes um Daten zu rekonstruieren. Diese RS-Codes werden auch bei Audio-CDs eingesetzt, da sie sehr gut Burstfehler korrigieren können. Burstfehler sind Fehler die z.B. durch Kratzer auf einer CD auftreten und eine ganze Folge von Daten zerstören.

Die Daten eines QR-Codes werden mit einer Maske versehen, damit im Code zufällig entstandene Muster das Einlesen nicht stören.

Der QR-Code hat durch seinen Aufbau und seine Fehlererkennungsmechanismen eine hohe Fehlertoleranz. Diese hohe Fehlertoleranz ist absolut berechtigt, wenn man sich überlegt, dass QR-Codes mit Smartphone-Kameras ausgelesen werden können. Diese Kameras haben größtenteils keinen Bildstabilisator und liefern bei schlechter Beleuchtung keine guten Bilder.

Ich denke die pixeligen Codes werden uns in Zukunft im Bereich des maschinellen Sehens noch öfter begegnen.

Literatur

[1] Sweeney, P., *Error control coding: from theory to practice*, Wiley, 2002.

[2] Seidemann T., *Barcodes Sweep the World*, http://www.barcoding.com/information/barcode_history.shtml, letzter Zugriff: 11.06.2012

[3] *ISO/IEC 18004 Information technology Automatic identification and data capture techniques Bar code symbology QR Code*, Organization for Standardization, 2000.

[4] Johannes Blömer, *Skript zur Vorlesung Algorithmische Codierungstheorie*, Universität Paderborn, 2007.

[5] J. Stoer, *Numerische Mathematik 1*, Springer, 2004.

[6] DENSO WAVE INCORPORATED, *About QRcode*, http://www.qrcode.com/en/index.html, letzter Zugriff: 11.06.2012

Hochverfügbare Computersysteme durch Virtualisierung

Henning Ziegler
Proseminar: Fehlertolerante Systeme
E-Mail: henning.ziegler@uni-oldenburg.com

***Zusammenfassung*—Im Folgenden wird ein kurzer Überblick über die Bereitstellung von hochverfügbaren Systemen durch Virtualisierung gegeben. Dazu werden wichtige Begriffe wie Hochverfügbarkeit, geplante/ungeplante Downtime sowie Methoden zur Messung von hochverfügbaren Systemen definiert und erläutert. Weiterhin werden der allgemeine Aufbau und die Konzepte hinter hochverfügbaren Systemen erläutert und die Wahl des Standortes diskutiert. Hauptaugenmerk ist jedoch die Realisierung der Hauptkonzepte mit Hilfe von Virtualisierungstechniken.**

I. Motivation

Computersysteme haben einen immer größeren Einfluss auf unser Leben. Computer werden nicht nur im täglichen Leben, sondern auch in sicherheitskritischen Bereichen eingesetzt, wie z.B. in Banken zur Abwicklung der Überweisungen, in der Luftfahrt zur Flugsicherung oder zur Steuerung von Ampelanlagen oder Kraftwerken. In solchen Systemen kann ein längerer Ausfall der Computer schwerwiegende Folgen haben. In Banken kann ein Ausfall zu hohen Geldverlusten führen und in der Flugsicherung kann ein Ausfall im Ernstfall Menschenleben kosten. Aber auch der Ausfall der Serverinfrastruktur großer Unternehmen kann viel Geld kosten. Insofern ist es wichtig, diese Systeme möglichst ausfallsicher („hochverfügbar“) zu konzipieren.

II. Hochverfügbarkeit

A. Definition

Ein Computersystem gilt als verfügbar wenn es seine Aufgabe erfüllt. Beispielsweise ist ein Webserver verfügbar, wenn die Webseite, die er hosted, abrufbar ist. Das Verhältnis zwischen Ausfallzeit (Downtime) und Betriebszeit (Uptime) wird als Verfügbarkeit bezeichnet:

$$Verfügbarkeit = \frac{Uptime}{Uptime + Downtime} \qquad (1)$$

Dabei ist der Unterschied zwischen geplanter Ausfallzeit und ungeplanter Ausfallzeit wichtig. Als geplante Ausfallzeit bezeichnet man beispielsweise die Nichtverfügbarkeit während geplanter Wartungsarbeiten. Die Dauer von geplanten Ausfallzeiten kann man relativ genau abschätzen. Ungeplante Ausfallzeiten hingegen entstehen spontan durch Hard- oder Softwarefehler.

Unter Hochverfügbarkeit versteht man nun die Fähigkeit eines Systems, auch im Fehlerfall zu funktionieren. Ein hochverfügbares System sollte also eine möglichst geringe (ungeplante) Downtime haben. Je nach Grad der Verfügbarkeit teilt man solche Systeme in sogenannte Verfügbarkeitsklassen ein. [1]

Tabelle I: Einteilung nach HRG [2] [3]

AEC	Bezeichnung	Beschreibung
AEC-0	Conventional	Betrieb kann unterbrochen werden, Datenintegrität unwichtig
AEC-1	High Reliable	Betrieb kann unterbrochen werden, Datenintegrität wichtig
AEC-2	High Availability	Betrieb darf nur zu festgelegten Zeiten minimal unterbrochen werden
AEC-3	Fault Resilent	Betrieb muss innerhalb festgelegter Zeiten ununterbrochen gewährleistet sein
AEC-4	Fault Tolerant	Betrieb muss 24 Stunden 7 Tage die Woche gewährleistet sein
AEC-5	Disaster Tolerant	Verfügbarkeit muss unter allen Umständen gewährleistet sein (auch im Katastrophenfall)

Tabelle II: Einteilung nach Verfügbarkeit [2] [3]

Stufe	Bezeichnung	Verfügbarkeit	Downtime pro Jahr
2	stabil	99%	3,7 Tage
3	verfügbar	99,9%	8,8 Stunden
4	hochverfügbar	99,99%	52,2 Minuten
5	fehlerunempfindlich	99,999%	5,3 Minuten
6	fehlertolerant	99,9999%	32 Sekunden
7	fehlerresistent	99,99999%	3 Sekunden

B. Verfügbarkeitsklassen

Verfügbarkeitsklassen dienen der besseren Abgrenzung der einzelnen Systeme zueinander. Eine gängige Einteilung ist die von der Harvard Research Group (HRG) entwickelte Availability Environment Classification (AEC). Diese sechs Klassen unterteilen Computersysteme nach ihrer Gewährleistung der Verfügbarkeit: AEC-0 – keine Gewährleistung bis AEC-5 – Gewährleistung selbst im Katastrophenfall. Die Einteilung ist der Tabelle I zu entnehmen. [2]

Eine etwas feinere Einteilung von hochverfügbaren Computersystemen kann anhand der prozentualen Verfügbarkeit pro Jahr vorgenommen werden (Tabelle II).

C. Weitere Kennzahlen

Neben der allgemeinen Verfügbarkeit sind noch weitere Kennzahlen wichtig. So ist beispielsweise die Mean Time Between Failure (MTBF), also die mittlere ausfallfreie Zeit, ein wichtiger Indikator. Die MTBF sollte bei einem hochverfügbaren System möglichst groß sein. Ebenso wichtig ist die Mean Time To Repair (MTTR), also die mittlere Reperaturzeit, nach einem Systemausfall. Im Optimalfall sollte die Reparatur bzw. die Wiederherstellung der Funktion so wenig Zeit wie möglich benötigen. [2] [3]

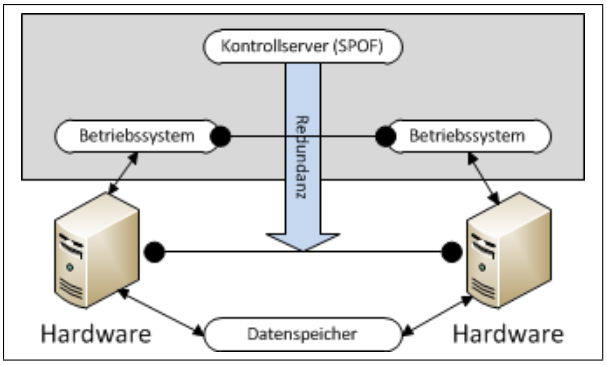

Abbildung 1: Redundantes System, in Anlehnung an [4]

III. Genereller Aufbau hochverfügbarer Systeme

Die Aufgabe bei dem Aufbau von hochverfügbaren Systemen liegt darin, das System gegen möglichst viele Fehlerklassen abzusichern. Diese Fehler gehen von Benutzerfehlern über Speicherprobleme/-verlust bis hin zu Katastrophen (Unwetter, Feuer, Sabotage, etc.).

Da eine Absicherung gegen alle Fehlerklassen in den meisten Fällen sehr kostspielig ist, muss vorher abgewägt werden, welche Fehler wirklich auftreten und wie viel Schaden sie anrichten können. Benutzerfehler richten verhältnismäßig wenig Schaden an, treten aber relativ häufig auf. Umweltkatastrophen hingegen verursachen viel Schaden, sind aber eher selten. Insofern wird es für die meisten Systeme nicht notwendig sein, sie gegen Umweltkatastrophen zu schützen.

A. Absicherung des Systems gegen Fehler

Um Benutzerfehler zu verhindern, müssen entsprechende Mechanismen in der Software implementiert werden. hardwareseitig können Benutzerfehler nicht abgefangen werden. Auf das Abfangen von Benutzerfehlern wird hier deswegen auch nicht weiter eingegangen. Gegen den Ausfall der Hardware wird massiv das Konzept der Redundanz ausgenutzt. Das bedeutet, dass die Hard- und Software mehrfach parallel vorhanden sein muss. Dadurch kann bei Ausfall eines Computers ein anderer seine Aufgabe übernehmen. Die redundanten Computer werden während des Betriebs mit denselben Daten wie das Hauptsystem versorgt. Dadurch wird ein unterbrechungsfreier Übergang zwischen zwei Computern gewährleistet. Dazu wird jedoch eine Kontrollkomponente benötigt, die die Hardware überwacht und im Fehlerfall die Umschaltung auf eine andere physische Maschine anstößt (Abbildung 1 oben).

Um die Datensicherheit zu gewährleisten, werden redundante Speichersysteme genutzt. Ein solches Speichersystem ist zum Beispiel das ”Redundant Array of Independent Disks”(RAID). Hierbei werden die Daten je nach RAID-Modus auf mehreren Platten gespiegelt oder mit Paritätsinformationen versehen. So kann der Ausfall einzelner Festplatten relativ einfach kompensiert werden. Zusätzlich sollten regelmäßig Backups der Daten angefertigt werden, um fehlerhafte Daten, die beispielsweise durch fehlerhafte Benutzereingaben verursacht wurden, wiederherstellen zu können.

Jedoch sorgt redundante Hardware nicht zwangsläufig für eine höhere Verfügbarkeit. Es gilt auch sogenannte Single Point of Failures (SPOFs) zu identifizieren und zu vermeiden. SPOFs sind Komponenten von deren Funktion das ganze System abhängt. So ist beispielsweise die Kontrollkomponente des Clusters ein solcher Single Point of Failure. Fällt die Kontrollkomponente aus, ist die Steuerung des Clusters nicht mehr gewährleistet. Also wird der Ausfall einer physischen Maschine nicht kompensiert. Ebenfalls ein Single Point of Failure ist das Dateisystem. Dies ist auf den ersten Blick nicht ersichtlich, da die Daten in der Regel wie oben beschrieben redundant gespeichert werden. Jedoch wird dafür eine Komponente (Volume Manager) benötigt, die das Speichern der Daten auf den Festplatten organisiert. Ohne diesen Volume Manager ist der Zugriff auf die Daten nicht mehr möglich. Es hängt also im Wesentlichen davon ab, wie genau man das System auf SPOFs untersucht. Je genauer desto mehr SPOFs treten unter Umständen zutage. [4]

Ein weiterer wichtiger Punkt ist, die Systemarchitektur möglichst simpel zu halten. Denn je komplexer ein System ist, desto anfälliger ist es für Fehler. Ursachen für Ausfälle sind bei solchen Systemen schwerer auszumachen, da die Architektur im Allgemeinen unübersichtlicher ist. Das kann eine höhere Downtime zur Folge haben als ein vergleichbares einfach aufgebautes System. [4]

B. Anforderungen an den Standort

Um einen hochverfügbaren Betrieb eines Computersystems zu gewährleisten, ist auch der Standort des Rechenzentrums wichtig. Dabei gilt je sicherer die klimatischen und geologischen Bedingungen, desto unwahrscheinlicher ist der Ausfall durch Umweltkatastrophen (z.B.: Erdbeben, Unwetter, etc.).

Auch der innere Aufbau ist relevant. Viele Rechenzentren besitzen mittlerweile einen doppelten Boden in dem die Kabel verlegt sind, um über Wartungsklappen zugänglich zu sein. So können sehr schnell Reparaturen durchgeführt werden. Ebenfalls wichtig ist eine redundante Stromversorgung. Im besten Fall aus verschiedenen Stromnetzen, um selbst bei einem Stromausfall in einem Netz weiterhin betriebsbereit zu bleiben. Zusätzlich ist auch ein Notstromaggregat sinnvoll, um kurzfristige Stromausfälle zu überbrücken.

Um Sabotage vorzubeugen, dürfen nur bestimmte Personen Zutritt zum Rechenzentrum bekommen. Der Zutritt muss also beispielsweise über Zugangschipkarten geregelt werden. [4]

IV. Hochverfügbarkeit mit Virtualisierung

Zusammengefasst zeichnen sich hochverfügbare Systeme durch einige wesentliche Eigenschaften aus. Sie sollten eine hohe MTBF und eine geringe MTTR haben. Das schließt eine geringe ungeplante Downtime ein. Im folgenden Abschnitt werden Techniken vorgestellt, wie diese Eigenschaften mithilfe von Virtualisierung erreicht werden können.

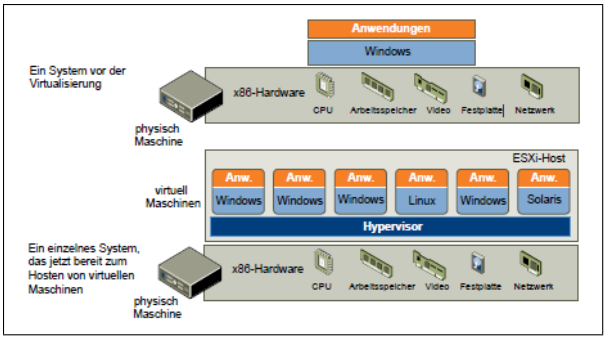

Abbildung 2: Vergleich Virtualisierung [6]

A. *Was ist Virtualisierung*

Virtualisierung ermöglicht es, vorhandene Ressourcen auf mehrere voneinander unabhängige logische Systeme zu verteilen. Diese logischen Systeme werden auch virtuelle Maschinen genannt. Jede virtuelle Maschine verhält sich wie ein komplett eigenständiger Computer und ist in der Regel von den anderen virtuellen Maschinen, die auf derselben Hardware laufen, isoliert.

Um die Computerhardware virtualisieren zu können, wird eine weitere Schicht zwischen der Hardware und den virtuellen Maschinen notwendig: Der Hypervisor. Der Hypervisor abstrahiert von der zugrundeliegenden Hardware und stellt den virtuellen Maschinen so einheitliche Umgebungen zu Verfügung. [5]

Das Hauptproblem dabei ist, dass konventionelle Betriebssysteme in der Regel exklusiven Zugriff auf die Hardware erwarten. Der Hypervisor muss also beispielsweise dafür sorgen, dass ein Betriebssystem in einer virtuelle Maschine nur den ihm zugewiesenen Speicher manipulieren kann. Dazu muss der Hypervisor privilegierte Prozessorbefehle abfangen und gegebenenfalls anpassen, um die gegenseitige Isolation der virtuellen Maschinen aufrecht zu erhalten. Heutzutage bieten die meisten Prozessorfamilien Hilfsmittel für diesen Zweck an. [5]

Abbildung 2 zeigt den Unterschied zwischen dem herkömmlichen Betrieb eines Servers und dem Betrieb mit Virtualisierung. Im herkömmlichen Betrieb läuft das Betriebssystem direkt auf der Hardware, sodass pro Hardware lediglich eine Maschine vorhanden ist. Ist auf der physischen Maschine hingegen ein Hypervisor installiert, können auf ein und derselben Hardware mehrere Betriebssysteme/Anwendungen ausgeführt werden.

B. *Aufbau eines Virtualisierungsclusters*

In einem Virtualisierungscluster wird nun noch weiter von den physischen Maschinen abstrahiert, um so größere Mengen verwalten zu können. Dazu werden mehrere VM-Hosts (Hardware mit installiertem Hypervisor) zusammengefasst und bilden einen sogenannten Ressourcen-Pool, in dem die virtuellen Maschinen verteilt werden. Um eine korrekte Verteilung zu gewährleisten, wird eine Kontrollinstanz benötigt. Die Aufgabe der Kontrollinstanz ist es die VM-Hosts möglichst gleichmäßig zu belasten um Ressourcenengpässe zu vermeiden. Meistens ist diese Kontrollinstanz ein externer Verwaltungsserver. Es gibt jedoch auch Verfahren, bei denen die einzelnen VM-Hosts die Verteilung untereinander ausmachen. [5] [6]

Die Daten der virtuellen Maschinen werden in der Regel auf externen Speichersystemen, wie NAS (Network Attached Storage) und SAN (Storage Area Network), gespeichert, damit die Daten durch einen Ausfall des VM-Hosts nicht beeinträchtigt wird.

V. Erreichung der Hochverfügbarkeit

Auf den ersten Blick hat Virtualisierung einen negativen Effekt auf die Verfügbarkeit. Da mehrere Systeme auf derselben Hardware laufen, führt deren Ausfall zum Ausfall von gleich mehreren Systemen. Wohingegen ohne Virtualisierung lediglich ein System vom Ausfall betroffen wäre. Außerdem ist gerade die Kontrollinstanz, die die Verteilung der virtuellen Maschinen organisiert, ein Single Point of Failure.

Mit einem Ressourcen-Pool hat man jedoch bereits einen großen Schritt in Richtung Hochverfügbarkeit getan. Ein richtig konfigurierter Ressourcen-Pool ist in der Lage, virtuelle Maschinen auf jedem beliebigen VM-Host zu starten und zu betreiben. Dies ist eine Grundvoraussetzung für die Hochverfügbarkeit und ein großer Vorteil gegenüber konventionellen Methoden.

A. *Verringerung der geplanten Downtime*

Mithilfe von Virtualisierung ist es möglich, die geplante Downtime zu verringern. Geplante Downtime entsteht bei Wartungsarbeiten, wenn beispielsweise Hardware ausgetauscht oder repariert werden muss. Durch die einheitliche Architektur, die die Hypervisoren bieten, kann eine virtuelle Maschine praktisch auf jeder Hardware im Ressourcen-Pool ausgeführt werden. Muss ein VM-Host ersetzt werden, kann man dies ausnutzen. Die auf ihm laufenden virtuellen Maschinen könne dabei auf andere VM-Hosts im Ressourcen-Pool verschoben werden.

Dieser Vorgang geht ohne sichtbare Unterbrechung der Funktion der verschobenen virtuellen Maschinen vonstatten. Die Verfügbarkeit bleibt also gewährleistet. Der betreffende VM-Host kann nun ohne Probleme repariert oder ersetzt werden. Es kommt in diesem Fall also zu keiner wartungbedingten Downtime. Alle größeren Virtualisierungsprodukte - wie Citrix XenServer, VMware vSphere oder Microsofts Hyper-V - unterstützen diese sogenannte Live-Migration von virtuellen Maschinen. [7]

Man benötigt in diesem Sinne also kein geeignetes Backup-System, sodass sich Wartungsarbeiten an der Hardware relativ einfach und schnell erledigen lassen. Des Weiteren wird die geplante Downtime erheblich reduziert, weshalb das System eine höhere Verfügbarkeit aufweist.

B. *Verringerung der ungeplanten Downtime*

Auch die ungeplante Downtime kann mithilfe von Virtualisierung verringert werden. Um den Ausfall der Hardware zu erkennen, werden meistens Heartbeat-Signale verwendet. Heartbeat-Signale werden in kurzen Zeitabständen gesendet und zeigen so die Funktion (wie der Herzschlag bei Lebewesen) des Senders an. Bleibt das Signal aus,

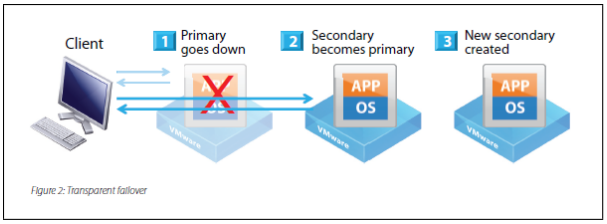

Abbildung 3: Transparenter Übergang [8]

kann man davon ausgehen, dass der entsprechende Sender ausgefallen ist.

Der Ausfall der Hardware kann im Wesentlichen durch zwei Möglichkeiten kompensiert werden.

Die erste Möglichkeit, die von den meisten Virtualisierungslösungen unterstützt wird, startet virtuelle Maschinen nach dem Ausfall eines VM-Hosts auf einem anderen VM-Host im Ressourcen-Pool. Diese Lösung ist relativ einfach zu realisieren, da die Daten der virtuellen Maschinen in einem externen Speichersystem (NAS oder SAN) abgespeichert werden. Die Daten gehen also bei einem Hardwareausfall nicht verloren und die virtuelle Maschine kann neu gestartet werden. Der Nachteil dieser Variante ist jedoch, dass der aktuelle Status der ausgefallenen VM nicht zwangsläufig übernommen werden kann. Außerdem gibt es keinen transparenten Übergang, da die neugestartete VM erst einige Zeit benötigt, um alle Aufgaben zu übernehmen.

Dieses Problem geht die zweite Lösungsmöglichkeit an. Dabei werden zwei virtuelle Maschinen gekoppelt. Eine virtuelle Maschine beinhaltet das Hauptsystem (Primär-VM). Sie übernimmt alle Aufgaben und hat vollen Zugriff auf die Datenspeicher. An die zweite virtuelle Maschine (Sekundär-VM) werden alle Eingabedaten der Primär-VM weitergeleitet. Der interne Status beider Maschinen ist also gleich. Mit dem Unterschied, dass die Sekundär-VM keine Daten der Primär-VM verändern darf. Die Sekundär-VM kann man als eine Art Schattenkopie der Primär-VM ansehen. Sinnvollerweise laufen die beiden VMs auf unterschiedlicher Hardware, da sonst ein Hardwaredefekt beide VMs ausfallen lassen könnte.

Fällt nun die Primär-VM aus, kann die Sekundär-VM unmittelbar alle Aufgaben der Primär-VM übernehmen. Da alle Eingabedaten der Primär-VM gespiegelt wurden, geschieht der Übergang für den Anwender vollkommen transparent. Die Sekundär-VM wird so zur neuen Primär-VM. Um sich weiterhin gegen einen Fehler in der neuen Primär-VM abzusichern, wird nun eine neue virtuelle Maschine gestartet und mit der Primär-VM gekoppelt. So ergibt sich wieder die oben genannte Ausgangssituation. (Abbildung 3)

Die zweite Methode ist der ersten vorzuziehen, da dadurch ein transparenter, für den Nutzer nicht bemerkbarer Übergang von der defekten zu funktionierenden VM ermöglicht wird. [8]

VI. Fazit

Alles in allem lässt sich durch Virtualisierung ein System hochverfügbar machen. Im Gegensatz zu konventionellen Hochverfügbarkeits-Clustern, ist diese Technik auch günstiger und flexibler. Während bei konventionellen Hochverfügbarkeits-Clustern häufig spezielle Hard- und Software benötigt wird, genügt hierbei jede beliebige kompatible Hardware. Dadurch ist auch die Wartung weniger kostenintensiv und einfacher durchführbar.

Zudem lassen sich Ressourcen-Pools leicht erweitern oder verkleinern und können so an veränderte Anforderungen angepasst werden. Durch diese Flexibilität kann auch der Ausfall mehrerer Systeme kompensiert werden.

Ein häufiger Trugschluss ist jedoch, dass man durch Virtualisierung direkt eine hochverfügbare Architektur besitzt. Dies ist aber nicht der Fall. Um Hochverfügbarkeit zu gewährleisten, muss meistens noch einiges an Mehraufwand betrieben werden. So müssen unter anderem SPOFs eliminiert werden, die die Verfügbarkeit des Systems negativ beeinträchtigen könnten. Auch die Standortwahl des Rechenzentrums ist entscheidend, da der Aufbau des Standortes wesentlich die Reparaturzeiten beeinflusst. Diese Punkte sind jedoch auch bei herkömmlichen hochverfügbaren Architekturen zu beachten. Allerdings kommt bei Virtualisierung noch die Tatsache hinzu, das der Ausfall einer physischen Maschine gleich den Ausfall mehrerer virtuelle Maschinen zu Folge hat. Diesen Nachteil zu beheben fordert den meisten Aufwand.

Abgesehen davon bietet Virtualisierung, durch die oben genannten Vorteile (z.B. kostengünstige Anschaffung, Flexibilität), aber in jedem Fall ein stabiles Grundgerüst für jede hochverfügbare Architektur.

Literatur

[1] Website, 2012, online verfügbar: http://www.tecchannel.de/test_technik/grundlagen/429794/grundlagen_der_hochverfuegbarkeit/; besucht am 9. Juni 2012.

[2] Website, 2012, online verfügbar: http://www.tecchannel.de/test_technik/grundlagen/430342/hochverfuegbarkeit_kennzahlen_und_metriken/; besucht am 9. Juni 2012.

[3] R. Ronneburger, Vorlesung - PDF, online verfügbar: http://bis2.informatik.uni-leipzig.de/download/2002s_s_ieb/RalfRonneburger_Hochverfuegbarkeit.pdf; besucht am 10.07.2012.

[4] K. Schmidt, *High Availability and Desaster Recovery*. Springer-Verlag, 2006.

[5] F. Thorns, Ed., *Das Virtualisierungs-Buch*. C&L Computer und Literatur Verlag, 2007.

[6] Website - PDF, 2012, online verfügbar: http://pubs.vmware.com/vsphere-50/topic/com.vmware.ICbase/PDF/vsphere-esxi-vcenter-server-50-basics-guide.pdf; besucht am 7. Juli 2012.

[7] Website, 2012, online vefügbar:http://www.vmware.com/technical-resources/virtualization-topics/high-availability/load-balancing/load-balancing.html; besucht am 11. Juni 2012.

[8] "Protecting mission-critical workloads with vmware fault tolerance," 2012, online verfügbar: http://www.vmware.com/files/pdf/resources/ft_virtualization_wp.pdf; besucht am 11. Juni 2012.

Erkennen und Beheben von Verklemmungen in verteilten Systemen

René Heck
Systemsoftware & verteilte Systeme
Universität Oldenburg
Oldenburg, Deutschland
Email: rene.heck@informatik.uni-oldenburg.de

***Zusammenfassung*—Einführung in die Problematik der Verklemmungen im Zusammenhang mit der Fehlertoleranz in verteilten Systemen. Weitergehend Erläuterung des Algorithmus von Srinivasan & Rajaram zum erkennen und lösen von Verklemmungen in verteilten Systemen als generalisiertes Modells sowie das bewerten der Darstellung der Veröffentlichung.**

***Keywords*-Fehlertoleranz; Verteilte Systeme; Wait-For Graph; Verklemmungs- erkennung; Verklemmungs- behebung**

I. Einführung

A. Fehlertoleranz

Informal wird Fehlertoleranz so verstanden, dass ein System ein vordefiniertes Verhalten aufweist, wenn Fehler auftreten. [2] Ein System ist fehlertolerant, wenn das System eine schwächere Spezifikation SPEC' $\supseteq$ SPEC noch erfüllt. Wobei die SPEC' beschreibt, welches Verhalten noch 'akzeptabel' ist, wenn das System nicht mehr in seinem definierten Zustand SPEC arbeitet. [3]
In der Fehlerdiagnose wird das System analysiert, um fest zu stellen, ob es sich in einem fehlerfreien Zustand befindet. dies kann durch direkte Tests(Ist-Soll-Vergleiche) oder indirekte Tests(Ist-Ist-Vergleiche mehrerer Ergebnisse) festgestellt werden. Wenn dies nicht der Fall ist, wird durch Fehlerlokalisierung ermittelt, in welchem Subsystem sich der Fehler befindet.
Nachdem ein Fehler diagnostiziert wurde, müssen die Subsysteme in einen fehlerfreien Zustand zu überführen werden oder das wiederherstellen eines fehlerfreien Zustandes veranlasst werden. Bei der Wiederherstellung eines Fehlerfreien Zustandes kommt es zu einer Fehlerkompensation oder Fehlerbehandlung. Bei dem Überführen in einen fehlerfreien Zustand wird das System neu konfiguriert.

B. Verteilte Systeme

Eine präzise Definition des Begriffes eines verteilten Systems ist schwierig. Nach Tanenbaum ist ein verteiltes System ein Zusammenschluss unabhängiger Computer, der sich für den Benutzer als ein einzelnes System präsentiert. [4] Weiterhin kann ein verteiltes System aber auch aus Prozessen bestehen, die keinen gemeinsamen Speicher besitzen und daher über Nachrichten miteinander kommunizieren. [6]

C. Lamport'sche Uhren

Lamport'sche Uhren sind ein Konzept um den Verlauf von Ereignissen in einer happend-befor-Relation zu ordnen. Das bedeutet, dass die kausale Ordnung der logische Zeitstempel es dem System ermöglicht, Ereignisse die von einander abhängig sind, in einer eindeutig Reihenfolge zu ordnen. [12] Diese logischen Uhren sind nötig, da es sehr schwierig ist, die Zeit in einem verteiltem System synchron zu halten. Die lokalen Uhren in einem Rechner werden in der Regel über einen Quarz gesteuert. Ja jedoch jeder Quartz eine leicht andere Schwingungsfrequenz besitzt, führt dies zu Abweichungen.

D. Verklemmungen

Definition nach Tanenbaum: Eine Menge von Prozessen befindet sich in einer Verklemmung, wenn jeder dieser Prozesse auf ein Ereignis wartet, das nur ein anderer Prozess aus dieser Menge verursachen kann. [5] Notwendige Bedingungen für Verklemmungen sind:

1) Gegenseitigerausschluss(mutual exclusion): Wird eine Ressource von einem Prozess genutzt, dann müssen alle anderen Prozesse, die diese Ressource auch verwenden wollen, warten bis die Ressource freigegeben worden ist.

2) Halten und Warten(hold-and-wait): Prozesse dürfen Ressourcen nachfordern auch wenn sie bereits Ressourcen halten und ohne diese freigeben zu müssen.

3) Keine Enteignung(no-preemption): Eine Ressource, die einem Prozess zugeteilt wurde, wird nur freigegeben, wenn der Prozess sie freiwillig zurück gibt.

4) Zyklischeswarten-Bedingung(circular-wait): Zwei oder mehr Prozesse bilden einen Zyklus derart, dass jeder Prozess auf eine Ressource des nachfolgenden Mitglieds des Zyklus wartet. [7]

Unter diesen Bedingungen können Verklemmungen dazu führen, dass ein System partiell oder sogar vollständig nicht weiter arbeiten kann.
Ein einfaches Beispiel ist, wenn ein Rechner A etwas 'kopieren' möchte und dafür benötigt er einen Drucker und

einen Scanner. Zuerst sichert sich A im Netzwerk den Scanner, um das Dokument einlesen zu können. Zusätzlich dazu benötigt der Rechner noch den Drucker, um das Dokument sofort auszudrucken. Der Scanner meldet A zurück, dass er verfügbar und nun für ihn bereit ist.
Jedoch meldet sich der Drucker negativ zurück, da dieser Rechner B zugesagt hat etwas auszudrucken. An diesem Punkt herrscht noch keine Verklemmung. Da sobald B gedruckt hat, der Drucker A zur Verfügung stehen würde. B hingegen möchte ein Fax versenden. Hierfür benötigt er neben dem Drucker für den Sendebericht noch das Modem zum versenden und den Scanner zum einlesen des Dokumentes. Inzwischen kam es dazu, dass die Anfrage von B zuerst bei dem Drucker ankam. So dass der Drucker sich für B bereit hält. Jedoch kommt die Nachricht zum gravierend erst verspätet bei dem Scanner an, was durch eine längere Signallaufzeit entstanden sein kann. So dass der Drucker sich nun schon für A bereit hält.
An dieser Stelle ist nun eine Verklemmung entstanden, da A auf den Drucker wartet, von den B reserviert hat wartet und B auf den Scanner von A wartet. Da die Rechner in diesem Beispiel alle Geräte gleichzeitig benötigen, um ihren Auftrag durch zu führen. Auf diese Art und Weise können in einem Rechner selbst durch unterschiedliche Prozesse Verklemmungen entstehen. Wie auch in dem Beispiel schon angedeutet, ist in einem verteiltem System die Kommunikation zwischen den Teilnehmern und den Ressourcen von Bedeutung sein kann und weiter zunimmt je größer das System wird. In vielen Konzepten wird der Nachrichtenverlust im Netzwerk selbst nicht als Fehlerquelle betrachtet.

E. Wait-For Graphen

Sind ein Spezialfall eines Ressourcen-Zuteilungsgraphen. In einem Wait-For Graphen werden die einzelnen Ressourcen nicht explizit aufgeführt, da jeder Ressourcentyp, wie Drucker, Scanner, usw nur eine einzelne Instanz besitzt. Bezogen auf das Beispiel bedeutet dies, dass nur ein Drucker, Scanner und Modem in unserem Netzwerk als Ressourcen zur Verfügung gestellt werden. Somit ergibt sich aus dem Beispiel der Wait-For Graph:

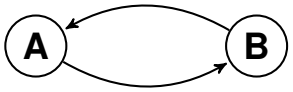

Abbildung 2. Wait-For Graph zu dem Beispiel

Rechner A wartet auf eine Ressource, die Rechner B zur Verfügung steht und Rechner B wartet auf eine Ressource die Rechner A zur Verfügung steht. Um welche Ressourcen es sich dabei explizit handelt, ist nicht weiter von Interesse, da die Ressourcen erst wieder abgegeben werden, wenn der Auftrag abgearbeitet wurde.

II. Historische Entwicklung der Algorithmen

Der Algoritmus von Selvarj Srinivasan & R. Rajaram baut auf mehreren algorithmischen Ansätzen auf.

A. Bracha, G., Toueg, S. (1987)

[8] Der Initiator verschickt in der ersten Phase Nachrichten, um sich ein Bild von dem Wait-For Graphen zu machen. In der zweiten Phase empfängt der Initiator von allen unblockierten Prozessen die Antworten. Aus den Antworten kann der Initiator erkennen, ob für ihn eine Verklemmung vorliegt. Dafür ermittelt er, ob er aus den Antworten sich selbst als unblockiert deklarieren kann. Für diesen Algorithmus werden $4k$[1] im Wait-For Graph Nachrichten benötigt und $4t^2$.

B. Brzezinski J. Helary J.M.,Raynal M., Singhal, M (1995)

[9] Der Algorithmus gruppiert die Prozesse in eine verteilte Momentaufnahme und bildet daraus einen logischen Ring. In diesem Ring lässt er einen Token kreisen. Wenn der Token das zweite mal in Folge den Prozess passiert und dieser seinen Zustand noch nicht geändert hat, wird eine Verklemmung erkannt. $\frac{1}{2}k^2$ Nachrichten werden hierfür benötigt und $4t$.

C. Kshemkalyani A.D., Singhal M., (1994)

[10] Bei diesem Algorithmus wird ebenfalls der reduzierte Wait-For Graph erstellt, während parallel bereits versucht wird, eine Verklemmung zu erkennen. Um eine Verklemmung zu entdecken, benötigtet der Algorithmus $4k - 2kn + 2l$[3][3] Nachrichten und $2t$.

III. Selvarj Srinivasan & R. Rajaram Algoritmus

A. Vorraussetzungen

Das System besteht aus n Prozessen, die eindeutig zu identifizieren sind. Die Prozesse können untereinander mittels Message passing('einfaches' Verschicken von Nachrichten) kommunizieren, ohne das sie gemeinsam Speicher(Shared Memory) nutzen. Es wird von einem verlustfreien Netzwerk ausgegangen.
Ein Prozess, der aktiv ist, hat die Möglichkeit Anfragen auf Ressourcen selbst zu stellen oder sie zu beantworten. Ebenso kann er seine Anfrage abbrechen oder die Anfrage eines anderen bestätigen. Während ein Prozess im Zustand blockiert ist, darf er nur noch Anfragen auf seine Ressourcen bestätigen oder Verwaltungsnachrichten verschicken, um eine Verklemmung zu erkennen.
Die Konsistenz der Nachrichtenreihenfolge wird durch Lamport'sche Uhren gewährleistet. [12]

[1]k ist die Anzahl der Kanten in dem WFG

[2]t ist der Durchmesser des Wait-For Graphen(Der Durchmesser D(G) eines Graphen G ist der größtmögliche Abstand zweier Knotenpunkte von G.) [13]

[3]kn die Anzahl der Knoten im WFG

[3]l ist die Nachrichtenlänge

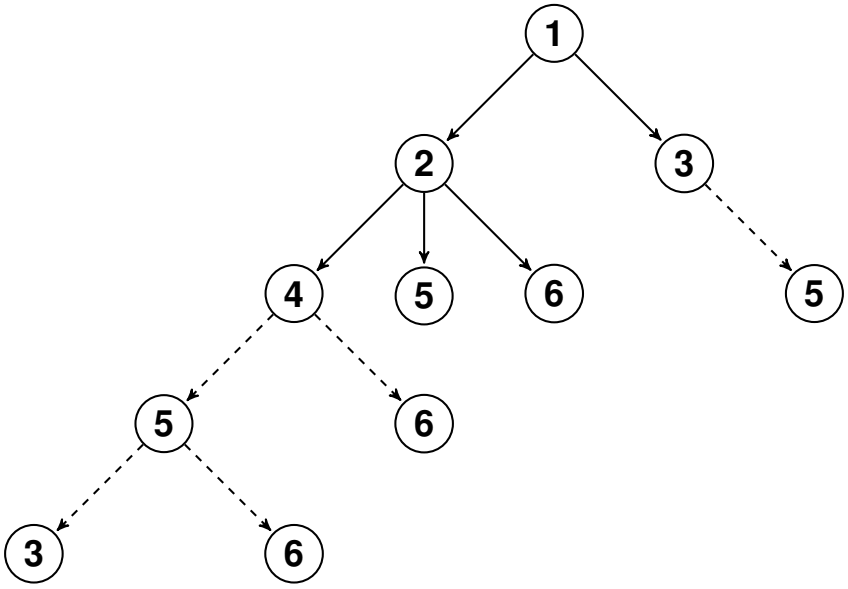

Abbildung 1. Verteilter aufspannender Baum

B. Der Algorithmus

1) Verklemmungserkennung: Der Initiator schickt zu jedem seiner Nachfolgern eine Anfrage. Sind diese ebenfalls blockiert, schicken sie eine Anfrage an ihre Nachfolger und warten eine Zeit auf die Antwort ihrer Nachfolger. Wenn von dem aktuellem Prozess ein Nachfolger der Vorgänger wäre, so spart er sich diese Nachricht und wartet auf seine anderen Nachfolger, sofern diese vorhanden sind. Über diese Methode wird der aufspannende Baum generiert.
Ein nicht blockierter Nachfolger antwortet seinem Vorgänger mit einem Report, um nicht weiter unnötige Anfragen zu generieren. Die Vorgänger warten auf den Report ihrer Nachfolger. Haben sie von allen Nachfolgern einen Report erhalten, können sie ihren Status ermitteln. Im Anschluss schicken sie ihren Report an ihre Vorgänger.

2) Verklemmungsbehebung: Es wird der Prozess von dem Initiator ausgewählt, der die meisten Vorgänger im Wait-For Graph hat die verklemmt sind. Dies kann er aus den Report's schließen, die er von seinen eigenen Nachfolgern bekommt. Diesem wird direkt ein Signal geschickt, um ihn zu beenden.

C. Beispiel

Es existieren insgesamt 6 Prozesse. Von diesen Prozessen warten alle auf eine Ressource, ausgenommen der 6. Prozess. Dieser kann seinen Auftrag abarbeiten. Folgende Prozesse wartenProzess auf einander: $P_1 \models 2 \wedge 3, P_2 \models (4 \wedge 5) \vee 6, P_3 \models 5, P_4 \models 5 \vee 6, und P_5 \models 3 \wedge 6$. Abbildung 2 zeigt nun den Wait-For Graphen aus dieser Annahme.

Hier in Abbildung 3 ist jedoch nicht deutlich zu erkennen dass P_2 auf P_4 und P_5 wartet **oder** auf P_6!
Ausgehend davon, dass der P_1 der Initiator ist, um eine

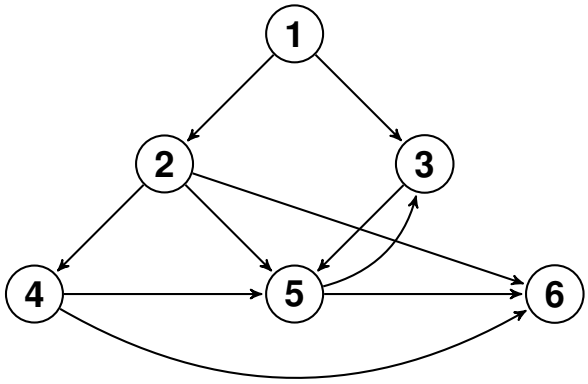

Abbildung 3. Wait-For Graph

Verklemmung zu entdecken und beheben, verschickt er Nachrichten, an seine Nachfolger auf die er wartet. Abbildung 1 zeigt den aufspannenden Baum zu dem Wait-For Graph. Nachdem alle Prozesse ihre Anfragen gestellt haben, antwortet P_6 auf die Anfragen sofort, die ihm zugesandt wurden, dass er nicht Verklemmt ist und keine Nachfolger hat.
Wenn P_5 diesen Report erhält, hat er alle benötigten Informationen um selbst einen Report an seine Vorgänger P_2,P_3 und P_4 zu schicken. Auf einen Report von P_3 muss er nicht warten, da dieser ebenfalls eine Anfrage an ihn gestellt hatte. Dieser Report enthält die Informationen, dass er blockiert ist, deklariert sich als Opfer und fügt hinzu das 2 Prozesse auf ihn warten. P_4 antwortet P_2 auf die Anfrage mit den Informationen, dass er selbst nicht blockiert ist jedoch P_5. P_2 wertet seinen Status ebenfalls aus und sendet

Abbildung 4. k ist die Anzahl der Kanten in dem WFG, P repräsentiert die Anzahl der Prozesse,kn die Anzahl der Knoten im WFG, l ist die Nachrichtenlänge und t ist der Durchmesser des Wait-For Graphen

Algorithmus	Dauer	Nachrichtenkomplexität	Nachrichtenlänge
Bracha, G., Toueg, S [8]	$4t$	$4k$	$O(l)$
Brzezinski J. [9]	P^2	P^2	$O(kn)$
Kshemkalyani [10]	$2t+2$	$2k-2kn+2l$	$O(k)$
Kshemkalyani [11]	$2t$	$2k$	$O(k)$
Selvarj Srinivasan, R. Rajaram [1]	$2t$	$2k$	$O(l)$

an P_1, dass er nicht blockiert ist jedoch P_5. P_3 ermittelt auf den Report von P_5 hin das er ebenfalls blockiert ist, wählt sich als Opfer aus mit ebenfalls P_2 Vorgängern und sendet dies an P_1, den Initiator.
Nun kann P_1 selbst seinen Status bestimmen, da es seinen Zustand nicht auf unblockiert deklarieren kann, erklärt der Algorithmus das eine Verklemmung vorliegt. Wählt P_3 aus um der Verklemmung zu beheben. Da dieser die meisten Nachfolger hat. [1]

IV. Performance Vergleich

Um die Performance zu vergleichen wird zuerst einmal die Nachrichtenkomplexität betrachtet. Dies bedeutet, das betrachtet wird wie viele Nachrichten der Algorithmus im 'worst-case' also im schlimmsten Fall benötigt um einen fehlerfreien Zustand herzustellen. Des weiteren ist von

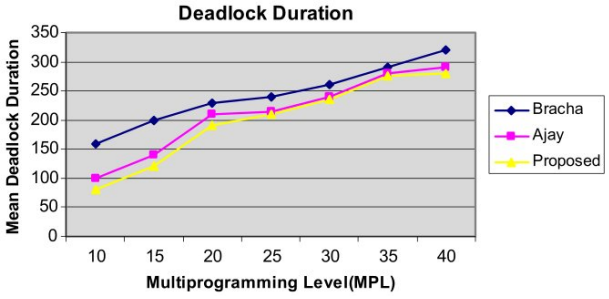

Abbildung 5. Mittlere Dauer der Verklemmung [1]

Interesse wie lange es dauert den Algorithmus auszuführen. Da dies von System zu System unterschiedlich sein kann, wurde in die Dauer in abstrakte Zeiteinheiten aufgeteilt. Eine Zeiteinheit wurde definiert als die Zeit, die das System benötigt eine Nachricht zu verschicken.
Die quantitativen Studien, die veröffentlicht wurden, basieren auf Implementierungen der Algorithmen von

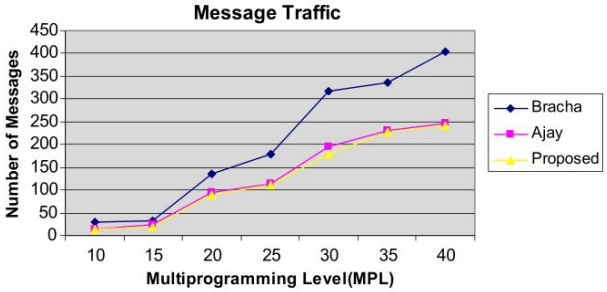

Abbildung 6. Nachrichtenkomplexität [1]

Bracha [8], Ajay [11] und dem vorgestelltem Algorithmus. Es wurden verglichen: die Nachrichtenlänge(Abbildung 7), Nachrichtenkomplexität(Abbildung 6) und die Zeit(Abbildung 5), die die Algorithmen benötigen in Relation zu dem Parallelisierungsgrad, welcher die Anzahl der Prozesse repräsentiert. Dieser variierte zwischen 10 und 40 in den Simulationen. Jede Simulation bestand aus 20 Iterationen mit den gleichen Parametern und es wurde der mittlere Wert aus diesen gebildet.

Aus den Diagrammen kann man entnehmen, dass die Dauer und die Nachrichtenkomplexität bis eine Verklemmung gelöst wurde, sich Algorithmen von Ajay und Selvarji ich annähernd gleich verhalten. Es besteht jedoch einen Unterschied bei der Nachrichtenlänge, diese bildet zugunsten des Algorithmus von Selvarj welcher eine feste Nachrichtenlänge lediglich benötigt [1].

V. Zusammenfassung

A. Fehlertoleranz

Der hier aufgearbeitete Algorithmus befasst sich mit sowohl mit der Fehlerdiagnose als auch die Fehlerbehandlung. Da das System zulässt das Verklemmungen entstehen, kann hier von Fehlertoleranz

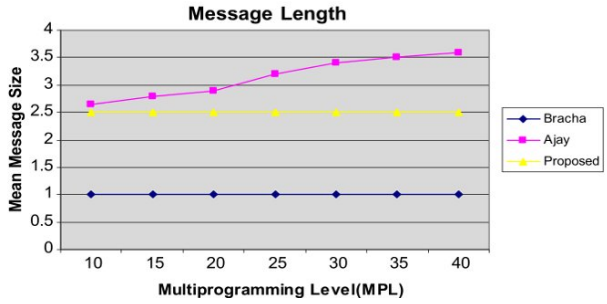

Abbildung 7. Nachrichtenlänge [1]

gesprochen werden.

1) Fehlerdiagnose: Wie bereits dargestellt, wird ein Prozess in diesem verteiltem System versuchen, eine Verklemmung festzustellen und zu lokalisieren. Es muss nicht unbedingt ein Fehler vorliegen damit der Algorithmus angestoßen wird. Wartet jedoch ein Prozess zu lange Zeit auf eine Ressource löst dieser den Algorithmus zur Erkennung und Behebung aus. Dafür wird die definierte Kommunikation zwischen den Prozessen ausgenutzt, um den aufspannenden Baum des Wait-For Graphen abzulaufen. Die einzelnen Nachfolger entscheiden für sich ob sie ein Problem generieren. Ab diesem Moment wurde ein Fehler erkannt und lokalisiert. Jedoch ob der Fehler an dieser Stelle behoben wird, entscheidet der Initiator.

2) Fehlerbehandlung: Die Fehlerbehandlung wird in diesem Algorithmus mittels der Vorwärtsfehlerbehebung bewerkstelligt. Im Gegensatz zur Rückwärtsfehlerbehebung muss hier kein vorgehender Zustand mit protokolliert werden, was weitere Informationen benötigen würde. Ein fehlerfreier Zustand wird erreicht indem ein Kriterium, dass benötigt wird, für eine Verklemmung außer Kraft gesetzt wird. In diesem Fall wird das Opfer seine bereits reservierten Ressourcen wieder abgeben. Danach kann das System weiter arbeiten.

B. Nutzen & Optimierung

Der Nutzen in dem vorgestellten Algorithmus von Selvarj Srinivasan & R. Rajaram besteht maßgeblich in der Optimierung der Nachrichtenkomplexität. Als auch in der Zeit die der Algorithmus benötigt, um eine Verklemmung zu erkennen und zu beheben. Wie man im Abschnitt 'Performance Vergleich' deutlich erkennen konnte, ist eine erhebliche Ersparnis der Nachrichtenkomplexität erreicht worden. Im Vergleich zu dem Algorithmus von Bracha [8]. Das Verhältnis zu dem Algorithmus von Kshemkalyani [11] wurde die Nachrichtenlänge reduziert, indem die Reports eine feste Größe haben.
Somit lässt sich aussagen dass, der Algorithmus gut skaliert und durch seine kurze Nachrichtenlänge auch in Netzwerken eingesetzt werden kann bei denen keine große Datentransferkapazität zur Verfügung steht.

LITERATUR

[1] Selvaraj Srinivasan and R. Rajaram, *A decentralized deadlock detection and resolution algorithm for generalized model in distributed systems*, Distrib Parallel Databases Kluwer Academic Publishers Hingham, MA, USA, 4 August 2011.

[2] Dipl.-Inform. Felix Christoph Gartner *Formale Grundlagen der Fehlertoleranz in verteilten Systemen*, Fachbereich Informatik der Technischen Universität Darmstadt, 20. März 2001.

[3] Anish Arora and Sandeep S. Kulkarni. *Component based design of multitolerant systems.* IEEE Transactions on Software Engineering, (1):63–78, January 1998.

[4] Andrew S. Tanenbaum, Maarten van Steen *Verteilte Systeme. Grundlagen und Paradigmen* Pearson Studium, 2007

[5] Andrew S. Tanenbaum *MODERN OPERATION SYSTEMS, 2. Edition*, Prentice-Hall, New Jersey, 2001

[6] Klaus-Peter Löhr *Verteilte Systeme*, Vorlesung www.inf.fu-berlin.de/lehre/WS03/VS/vorlesung/vs1.pdf Stand: 8. Juni 2012, Berlin WS 03/04

[7] Prof. Dr.-Ing. Oliver Theel *Verteilte Betriebssysteme*, Volesung 11, Oldenburg, Wintersemester 11/12

[8] Bracha, G., Toueg, S. *A distributed algorithm for generalized deadlock detection.* Distrib. Comput. 2, 127–138, 1987

[9] Brzezinski, J., Helary, J.M., Raynal, M., Singhal, M. *Deadlock models and a general algorithm for distributed deadlock detection. J. Parallel* Distrib. Comput. 31(2), 112–125, 1995

[10] Kshemkalyani, A.D., Singhal, M. *Efficient detection and resolution of generalized distributed deadlocks.* IEEE Trans. Softw. Eng. 20(1), 43–54, 1994

[11] Kshemkalyani, A.D., Singhal, M. *A one-phase algorithm to detect distributed deadlocks in replicated databases.* IEEE Trans. Knowl. Data Eng. 11(6), 880–895, 1999

[12] Lamport, L.: *Time, clocks, and the ordering of events in a distributed systems.* Commun. ACM 21, 558–565, 1978

[13] Th. Emden-Weinert: *Einfuhrung in Graphen und Algorithmen* URL: www.or.uni-bonn.de/ hougardy/paper/ga.pdf Zuletzt besucht 13.07.2012, 3. September 1996

Natürliche Fehlertoleranz in der Arterhaltung durch Evolution

Jens Krayenborg
Fakultät II
Carl von Ossietzky Universität
Oldenburg, Deutschland

***Abstract*—Das Überleben einer an seiner Umgebung angepassten Art ist stets durch Veränderungen der Umgebung beeinflusst. Einwirkungen von aussen bedrohen den Fortbestand aller Individuen einer Art. Der Weg der Natur mit dieser stetigen Bedrohung umzugehen ist die Evolution. Diese stellt ein Konzept dar, welches das Leben auf der Erde erhalten lässt. Zu diesem Zweck werden, neben der Anpassungsfähigkeit während eines Lebenzyklus eines Individuums, Zufall und Redundanz verwendet um die Überlebenswahrscheinlichkeit auch über die Generationsgrenze hinweg zu erhöhen. In diesem Konzept wird nicht mehr das Individuum betrachtet, sondern die Art als Ganzes.**

Dieses Paper wird neben der Trennung der Begriffe, auf die Techniken der Evolution eingehen und diese aus Sicht der Fehlertoleranz betrachten.

***Keywords*-component; Fehlertoleranz, Evolution, Arterhaltung, Divergenz;**

I. Einleitung

Das Leben auf diesem Planeten ist stetig Gefahren ausgesetzt. Diesen können die einzelnen Arten durch ihre individuellen Merkmale begegnen. Nicht immer ist dieser Kampf um das Überleben erfolgreich. Das betrifft einzelne Mitglieder einer Art, kann aber auch die ganze Art betreffen. Das Massensterben am Ende der Kreidezeit, als mit den Dinosauriern fast eine ganze Spezies ausstarb, ist ein Beispiel dafür.

Wenn davon ausgegangen wird, dass der gesicherte Fortbestand der gewünschte Zustand ist, dann ist der Fehlerzustand in dem Moment erreicht, in dem die Art vom Aussterben bedroht wird.

Ob und wie erfolgreich eine Art überlebt, wie gut sie also tolerant gegenüber Fehlern ist, ist stark von der Menge seiner Merkmale abhängig. Diese Merkmale definieren die Fähigkeit einer Art mit Bedrohungen umzugehen. Dieses allein reicht allerdings in der stets veränderlichen Umwelt nicht aus. Aspekte wie Klimawandel und Umweltverschmutzung, aber auch tektonische Plattenverschiebungen und damit einhergehende Dislozierung einzelner *Populationen* einer Art, können diese in ihrem Überleben bedrohen. Eine Möglichkeit zum Überleben bietet dann meisst nur noch die Anpassung der Merkmalsmenge. Diesen Vorgang bezeichnet man als Evolution. Evolution geschieht dabei nicht innerhalb einer Generation, sondern über die Generationsgrenze hinweg.

Evolution ist kein Vorgang der nur im Bedrohungsfall angestoßen wird, sondern geschieht jederzeit. Durch Bildung neuer Varianten des *Genoms* einer Art, sog. Allele, entsteht eine abgewandelte Merkmalsmenge des Nachkommens. Dies kann durch *Mutation* geschehen, aber auch durch Mischung der Gene bei der Zeugung von Nachkommen. Inwieweit es sich dadurch besser zum Überleben eignet wird erst die natürliche Auslese zeigen.

II. Begriffe

Begriffe wie *Population* und *Art* werden in der Biologie meist nicht einheitlich verwendet. Deswegen kläre ich sie im Folgenden für diesen Artikel geltend.

Eine oder mehrere Populationen bilden eine Art. Eine Art ist hierbei reproduktiv isoliert. Individuen einer Art können also keine fruchtbaren Nachkommen mit Individuen einer anderen Art zeugen. Die Möglichkeit zur Bastardbildung ist hierbei allerdings nicht generell ausgeschlossen.

Eine Population ist eine Gruppe einer Art, die sich einerseits, wenn auch nur gering, von anderen Populationen derselben Art unterscheidet, andererseits aber auch räumlich relativ eng beieinander residieren, örtlich und zeitlich. Während z.B. der *homo sapiens* eine Art bezeichnet, stellen Dänen eine andere Population dar, als Chinesen. Sie weisen geringe Unterschiede auf, sind aber zur Zeugung von zeugungsfähigen Nachkommen fähig.

III. Genetischer Code

Die Merkmale eines Lebewesens werden durch seine *Gene* bestimmt. Die *Desoxyribonucleinsäure* (DNS oder DNA) ist dabei der molekulare Baustein der Gene. Sie setzt sich zusammen aus *Pentose-Zuckermolekülen* (PZ-Molekül) und *Phosphatgruppen* (P-Gruppe) in abwechselnder Folge. Seitlich an jedem PZ-Molekül befindet sich, durch einfache *Wasserstoffpaarbindung* verbunden, eine der *Purine Adenin* (A), *Guanin* (C) oder *Pyrimidine Cytosin* (G) und *Thymin* (T) [2, S.12ff]. Zwei dieser Stränge bilden eine sog. *Doppelhelix*, bei der die Basen durch *Wasserstoffbrückenbindungen* miteinander verbunden sind.

Die jeweils gegenüberliegenden *Basen* sind dabei komplementär. Das heisst, es bindet sich immer je ein Purin mit einem Pyrimidin. Genauer bindet sich ein Adenin stets

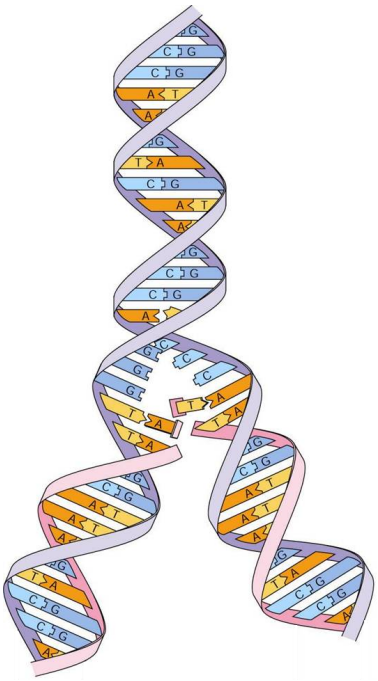

Figure 1. DNA Replikation [6]

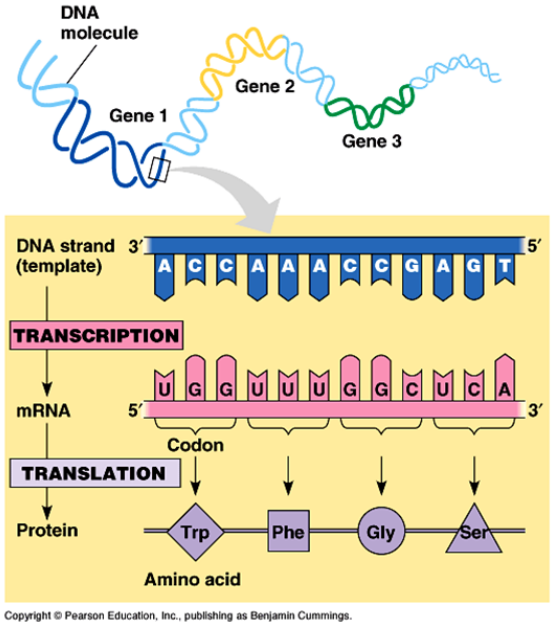

Figure 2. DNA Transkription [7]

mit einem Thymin und ein Guanin stets mit einem Cytosin. Denn Adenin und Thymin können nur zwei Wasserstoffbrückenbindungen eingehen, Guanin und Cytosin hingegen drei.

Figure 1 zeigt die *Selbstreplikation* einer DNA. Dies ist etwa zur Vorbereitung einer *Zellteilung* notwendig. Dazu lösen sich die Wasserstoffbrückenbindung. Auf diese Weise entstehen zwei getrennte komplementäre Stränge, an denen sich neue Basen binden. Nach Ausbildung des Rückgrats aus PZ-Molekülen und P-Gruppen sind dann zwei vollständige, identische DNA Doppelhelixe vorhanden [2, S.14][4, S.140].

Um aus der DNA Proteine zu bilden, wird sie, wie in Figure 2 skizziert, erst in eine *Ribonucleinsäure* (RNA) , genauer in eine *messenger-RNA* (mRNA), *transkribiert*. Das geschieht auf ähnliche Weise wie eine Replikation der DNA. Allerdings wird an den Stellen, wo sich auf dem DNA-Strang ein Adenin befindet kein Thymin auf der mRNA eingefügt, sondern ein *Uracil* (U) (Figure 2). Bei der Konstruktion der mRNA wird erst die DNA-Doppelhelix aufgetrennt und in der RNA Polymerase dann die mRNA erzeugt. Aus einer DNA Doppelhelix entstehen für gewöhnlich mehrere RNA, darunter verschiedene *Transfer-RNA* (tRNA) zum Transport der freien Aminosäuren. Wo eine Transkription beginnen soll und wo sie zu enden hat, wird bei der RNA Polymerase an dem *Start-* und den drei *Stop-Codons* erkannt (Figure 3).

An der mRNA ”fahren” die *Ribosomen* entlang und bauen die *Polypeptidketten* zusammen, indem tRNA freie Aminosäuren zum Ribosomen transportieren, sich im Ribosomen an eine Stelle der mRNA koppeln, die ihrem spezifischen Code entspricht, und das Ribosomen dafür sorgt, dass in Nachbarschaft liegende Aminosäuren zur Polypeptidkette verbunden werden.

Ein tRNA-Molekül besitzt zur Bindung an die mRNA ein *Basentriplett*. Folglich wird eine Aminosäure durch ein Wort aus drei Zeichen codiert. Da das Alphabet aus vier Zeichen besteht und jedes Wort aus drei Zeichen gebildet wird, existieren 64 Verschiedene Worte in der Codesprache (Figure 3). Mit diesen 64 Worten werden 20 Aminosäuren und drei Stopp-Codons codiert. Versuche haben ergeben, dass einige Aminosäuren mit mehr Codeworten als andere codiert werden [3, S.76f]. Es existiert also eine gewisse *Redundanz* im *Genetischen Code*.

Figure 3 zeigt die 64 verschiedenen Tripletts, die die 20 verschiedenen Aminosäuren codieren. Auffallend ist, dass eine Veränderung an dritter Stelle im Triplett oft keine Veränderung der codierten Aminsäure zur Folge hat, wie am Beispiel *Alanin* (Ala) zu erkennen ist.

IV. Variation

Eine Variation wird im allgemeinen durch Veränderungen in den *DNA-Sequenzen* verursacht. Sie kann positive oder negative Folgen haben. Die Ursache dieser Veränderungen können *Mutationen* und *Genfluss*, aber auch die normale Fortpflanzung, egal ob *sexuell* oder *asexuell*, sein.

A. Mutation

Eine Mutation ist eine von aussen herbeigeführte Veränderung in der DNA-Sequenz. Sie wird immer bezüglich der DNA-Sequenz eines *Elterngens* identifiziert. Radioaktive Strahlung oder chemische Stoffe können eine solche Veränderung herbeiführen. Solche Stoffe

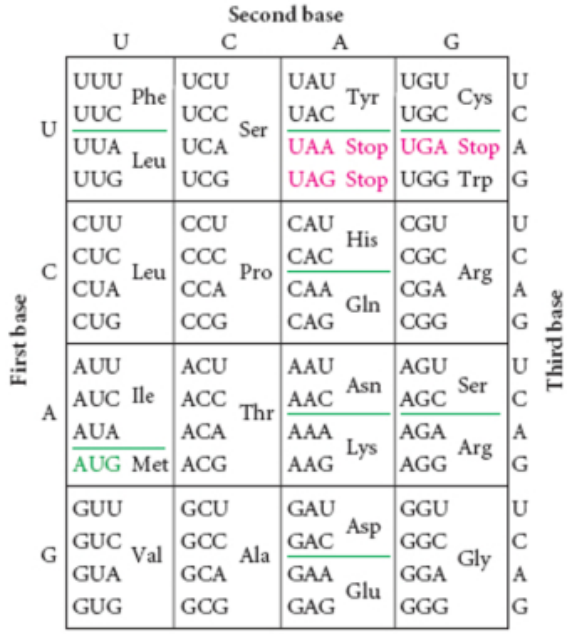

Figure 3. Genetischer Code [8]

und Strahlungen werden unter dem Begriff *Mutagen* zusammengefasst. Die Mutation verursacht dann oft eine Veränderung der Proteine, die durch die DNA codiert werden [5, S.3]. Mutationen werden, abhängig vom *Organismus* in dem sie auftreten und von der Umgebung in der dieser Organismus existiert, als *schädlich, neutral* oder *vorteilhaft* klassifiziert [5, S.2]. Zum Beispiel würde eine durch Mutation herbeigeführte *Resistenz* gegenüber eine Krankheit *X* nur in einer Umgebung als vorteilhaft bezeichnet werden, in der diese Krankheit auch vorkommt. Ansonsten würde man sie als neutral bezeichnen. Es werden vier Typen von Mutationen identifiziert: *Transition (I), Transversion (II), Deletion (III)* und *Insertion (IV)* [5, S.2ff]. Figure 4 visualisiert diese Typen.

originale Sequenz		mutierte Sequenz	Typ
CAAA<u>C</u>CTA	→	CAAA<u>T</u>CTA	(I)
CAAAC<u>C</u>TA	→	CAAAC<u>G</u>TA	(II)
CA<u>AACC</u>TA	→	CATA	(III)
CAAACCTA	→	CAAA<u>GGTTC</u>CTA	(IV)

Figure 4. Visualisierung der Mutationstypen

1) Punktmutation: Die Transition ist eine *Basensubstitution*, bei der entweder ein Purin gegen ein anderes Purin, oder ein Pyrimidin gegen ein anderes Pyrimidin ausgetauscht wird. Bei der Trasition wird ein Pyrimidin gegen ein Purin oder ein Purin gegen ein Pyrimidin substituiert.

Bei dieser Substitution kann auch eine Veränderung der durch dieses Basentriplett codierten Aminosäure resultieren. Findet eine solche Änderung nicht statt, dann wird diese Mutation als *still* oder *synonym* bezeichnet. Als Beispiel codiert das Basentriplett **TTG** die Aminosäure **Leucin**. Eine Transition nach **CTG** resultiert wieder in der Codierung von Leucin.

Führt die Substitution zu einer Veränderung der Aminosäure, so ist sie *nichtsynonym*. Je nach Resultat wird die Substitution dann als *sinnentstellend* oder *sinnverändernd* bezeichnet [5, S.3]. Bei einer sinnenstellenden Substitution entsteht ein sog. *Stopp-Codon*. Die Transversion der mittleren Base des o.g. **TTG**-Leucin nach **A** würde das Stopp-Codon **TAG** erzeugen. Eine sinnverändernde Substitution verusacht die Codierung einer anderen Aminosäure als vorher. So kann aus dem Leucin codierenden Triplett **TTG** durch Transversion der mittleren Base das Triplet **TTT** entstehen, welches **Phenylalanin** codiert.

2) Deletion und Insertion: Bei einer *Deletion* wird ein Teil der DNA-Sepuenz entfernt. Dies geschieht zum Beispiel bei einem ungleichen *crossing-over* zweier *Chromosomen*. Bei diesem Verfahren verwickeln sich zwei Chromosomen und trennen sich in jeweils zwei Teile, die dann wieder zusammengefügt werden. Dabei kann es passieren dass diese Teile vertauscht werden. Im Falle eines gleichen crossing-over kommt es zu keiner Änderung. Allerdings würde bei einem ungleichen crossing-over ein Teil der DNA bei einem Chromosom verloren gehen und beim anderen Chromosom hinzukommen (*Insertion*).

Diese beiden Mutationsarten können gravierende Auswirkungen haben. Die durch das Gen codierte Aminosäurekette kann schrumpfen oder verlängert werden (siege Figure 5), wodurch sich durch das Fehlen oder Hinzukommen einzelner Aminosäuren, auch das resultierende Protein verändert. Die Zelle verliert eventuell Fähigkeiten. Andererseits kann es auch dazu kommen, dass die Zelle Fähigkeiten hinzubekommt.

originale Sequenz:
ATG TTT TTG GCC TGT TAA TTG ATT...
Aminosäuresequenz:
Met(Start)-Phe-Leu-Ala-Cys-Stop
Sequenz nach Insertion von TA
ATG TTT TTG GCC TGT TAT AAT TGA TT...
Aminosäuresequenz:
Met(Start)-Phe-Leu-Ala-Cys-Tyr-Asn-Stop

Figure 5. Folgen in der Aminosäurekette 1

originale Sequenz:
ATG TTT TTG GCC TGT TAA TTG ATT...
Aminosäuresequenz:
Met(Start)-Phe-Leu-Ala-Cys-Stop
Sequenz nach Deletion von GCCT
ATG TTT TTG GTT AAT TGA TT...
Aminosäuresequenz:
Met(Start)-Phe-Leu-Val-Asn-Stop

Figure 6. Folgen in der Aminosäurekette 2

Wird eine *Basensequenz*, deren Anzahl der Basen ungleich einem Vielfachen von drei, hinzugefügt oder entfernt, dann kommt es zu einer *Phasenverschiebung* bei der Auswertung des genetischen Codes. Auch in diesem Fall verändert sich das resultierende Protein (siehe Figure 6). In diesem Fall jedoch betrifft die Veränderung nicht nur das durch den von der Deletion oder Insertion betroffenen Teil des DNS Strangs, sondern kann, im extremsten Fall, auf alle Gene, die nach der Veränderung folgen, Auswirkungen haben.

Inwieweit solche Mutationen Auswirkungen, egal ob positiv oder negativ, auf den gesamten Organismus haben, ist jedoch von der betroffenen Zelle abhängig.

B. Fortpflanzung

Für gewöhnlich sind die für die Fortpflanzung bedeutenden Zellen *haploid*. Das bedeutet sie besitzen nur einen Satz an Erbinformationen, die sich als DNA in den Chromosomen befinden. Paaren sich jedoch zwei Lebewesen, so kommt es zur Befruchtung. Dabei dringen die Erbinformationen des Vatertieres in die Erbinformationen enthaltende Eizelle des Muttertieres. Es entsteht eine *diploide* Zelle. Im *Zellkern* dieser Zelle, in der nun beide Chromosomensätze (diploid) vermengt sind, kommt es bei der nächsten *Meiose* zur zufälligen Aufteilung der Chromosomenmenge.

Daraus resultiert eine Neuverteilung der durch die Gene bestimmten Merkmale. Dadurch bilden die Individuen einer Art, selbst innerhalb einer Population, eine *divergente* Menge. Dies erhöht die Fähigkeit einer Art, wenn man sie als Ganzes betrachtet, auf Veränderungen der Umgebung zu reagieren.

C. Gendfluss

Wenn sich Berührungspunkte zwischen zwei oder mehr verschiedenen Populationen derselben Art ausbilden, dann kann es dazu kommen, dass sich diese *Fortpflanzungsgemeinschaften* vereinen. Dadurch vermengt sich auch der *Genpool*, was wiederum eine größere Variabilität der Merkmalsmenge für die Nachkommen bedeutet. Sind diese Berührungspunkte nur kurz vorhanden, so vermengen sich beide Populationen nicht, sondern hinterlassen gegenseitig Teile ihres jeweiligen Geenpools, was auch in einer vergrößerten Variabilität resultiert. Dieser Vorgang wird als *Genfluss* bezeichnet [1].

V. Fehlertoleranz des genetischen Codes

Figure 7 listet die Aminosäuren und ihre durchschnittlichen Wahrscheinlichkeiten in Prozent auf, bei denen es, durch Punktmutation an nur einer Stelle im Triplett, zu einem Wechsel der codierten Aminosäure kommt. Diese Werte basieren auf einer reinen Berechnung der Möglichkeiten, wie sie sich aus der Codetabelle in Figure 3 ergeben. Auch wurde von einer Gleichverteilung ausgegangen, was die Position der Mutation betrifft.

urspr. Aminosäure	I	II	III	Gesamt
Alanin	100	100	0	66
Arginin	77	100	22	66
Asparagin	100	100	66	88
Asparaginsäure	100	100	66	88
Cystein	100	100	66	88
Glutaminsäure	100	100	66	88
Glutamin	100	100	66	88
Glycin	100	100	0	66
Histidin	100	100	66	88
Isoleucin	100	100	33	77
Leucin	77	100	22	66
Lysin	100	100	66	88
Methionin	100	100	100	100
Phenylalanin	100	100	66	88
Prolin	100	100	0	66
Serin	100	100	22	74
Threonin	100	100	0	66
Tryptophan	100	100	100	100
Tyrosin	100	100	66	88
Valin	100	100	0	66
Gesamt	95	100	30	75

Figure 7. Wahrscheinlichkeit zur Änderung durch Punktmutation - Angaben in eins von Hundert

Auffällig ist, dass eine Mutation der zweiten Position im Triplett grundsätzlich einen Aminosäurenwechsel herbeiführt, wärend eine Mutation an dritter Stelle nur in ungefähr einem Drittel aller Fälle eine Folge hätte. Insgesamt führen drei von vier Punktmutationen zu einer Veränderung.

VI. Natürliche Selektion

Aufgrund der Variabilität der Merkmalsmenge der Individuen einer Art untereinander, sind diese unterschiedlich gut für das Überleben in ihrer Umgebung und in ihrer Zeit geeignet. Durch Variation kann ein Individuum Resistenzen aufbauen oder attraktiver für einen potenziellen Fortpflanzungspartner werden. Betrachtet man als Beispiel einen Fasan, dessen Federkleid durch genetische Veränderung länger und imposanter geworden ist, als es bei seinen Artgenossen der Fall ist, so würde dieser Fasan wahrscheinlich einen größeren Erfolg bei der Partnersuche haben. Das führt unweigerlich zu einer größeren Verteilung seiner Gene (einschließlich imposantem Federkleid) auf die nachkommende Generation. Dadurch verringert sich auch der Anteil der Nachkommen mit weniger imposantem Federkleid innerhalb der Population. Auf lange Sicht könnte das dazu führen, dass es irgendwann keine Fasane mit weniger imposantem Federkleid mehr gibt. Diese Fasane wurden durch die natürliche Selektion ausgesiebt.

Die natürliche Selektion ist ein wichtiger Faktor, der die Evolution einer Art überhaupt erst ermöglicht. Interessanterweise, ist die natürliche Selektion aber auch eine

Folge der Wechselwirkungen zwischen Individuen, Mutationen der Gene, Rekombinationen bei der Fortpflanzung und Veränderung der Umwelt.

VII. Zusammenfassung - Fehlertoleranz in der Evolution

Mutationen und Rekombinationen verändern die Individuen einer Art. Dadurch gewinnen oder verlieren sie Fähigkeiten, die notwendig sind um auf Veränderungen der Umgebung zu reagieren. Versucht man diesen Komplex aus Art, Überlebensfähigkeit, Evolution und Umgebung systematisch zu beschreiben, so könnte Figure 8 dienen.

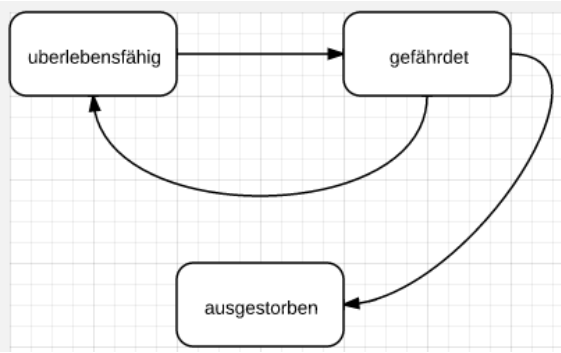

Figure 8. Artenzustände

Eine Art ist dann überlebensfähig, wenn ihre Individuen in ausreichender Menge vorhanden sind und sich Fortpflanzungspartner frequentiert genug treffen um genügend Nachkommen zu erzeugen, so dass die Menge an Individuen dieser Art nicht unter eine bestimmte Grenze fällt. Wo diese Grenze anzusiedeln ist, ist von der Art und der Umgebung abhängig. Würde es bei Einzellern schon reichen wenn lediglich ein Individuum vorhanden ist, so sind bei mehrzelligen Organismen meist mehr Individuen nötig.

Die Überlebensfähigkeit kann auf der einen Seite von Veränderungen der Umgebung verschlechtert werden, auf der anderen Seite aber auch durch Variation der Art ansich. So kann ein Umweltereignis die Population einer Art drastisch herabsetzen, so dass sich die Mitglieder nicht mehr oft genug begegnen um sich in ausreichender Menge fortzupflanzen. Die Art würde dann in den Zustand gefährdet übergehen. Eine Mutation kann dies ebenso verursachen. Eine durch Mutation auftretende Anfälligkeit gegen einen in der Umwelt schon immer befindlichen Krankheitserreger könnte dazu führen, dass die Größe der Population soweit schrumpft, dass diese schlussendlich ausstirbt.

So gesehen würden Mutationen und Umgebungsveränderungen den Mangel darstellen, der dazu führen würde, dass Teile des Systems (Individuen einer Art) nicht mehr spezifikationsgemäß (sie pflanzen sich in ausreichender Menge fort) funktionieren (Fehler). Der Ausfall der Art würde dann erreicht sein, wenn diese ausstirbt.

Mutationen stellen aber nicht unbedingt einen Mangel dar. Auf der anderen Seite kann es durch sie auch dazu kommen dass diese Art weiter expandiert und in ihrer Zahl ansteigt. Auch kann es vorkommen, dass sie auf zukünftig eintretenden Mängel besser reagieren können. So gesehen also toleranter gegenüber möglichen Fehlern werden. Auch ist eine Veränderung der Umgebung nicht unbedingt als Mangel zu sehen, so würde das Schrumpfen der Größe einer Population Y, die sich von der Art X ernährt, mit Sicherheit positive Auswirkungen auf die Art X haben. Der Umgekehrte Fall aber würde eventuell die Art X dann gefährden wenn die Art Y wächst.

In jedem Fall überleben nur die Individuen in ausreichender Zahl, wenn sie schon mit den Fähigkeiten geboren wurden, mit denen sie auf bestimmte Veränderungen reagieren, um lange genug zu überleben, um sich zu replizieren. Da die Zukunft nicht vorhersagbar ist, benötigt eine Population eine möglichst breite Vielfalt an Merkmalskombinationen um diese Überlebensfähigkeit zu erreichen. Dies schafft die Natur durch Rekombination der Gene bei der Zeugung der Nachkommen. Mutationen und Genfluss sorgen dafür, dass auch vollkommen neue Merkmale in den Genpool einer Art Einzug erhalten.

Auf diesem Wege erreicht die Natur durch Evolution eine gewisse Fehlertoleranz gegenüber negative Veränderungen. Diese Toleranz wird durch Diversität erreicht, bei der die Unterschiede zwischen den einzelnen Replikaten durch die zufällige Rekombination der Gene nach der Fortpflanzung entstehen.

References

[1] *Gene Flow* url: http://evolution.berkeley.edu/evosite/evo101/IIIC4Geneflow.shtml zuletzt besucht am: 11.07.2012

[2] F. H. C. Crick, *Die Struktur der Erbsubstanz DNA*, Spektrum der Wissenschaft, 1988.
ISBN: 3-922-508-30-8

[3] F. H. C. Crick, *Der genetische Code*, Spektrum der Wissenschaft, 1988.
ISBN: 3-922-508-30-8

[4] C. Grobstein, *Die Debatte um DNA-Rekombinationstechniken*, Spektrum der Wissenschaft, 1988.
ISBN: 3-922-508-30-8

[5] R. Devoret, *Mutation, eLS*, John Wiley & Sons Ltd, 2001.
url: http://dx.doi.org/10.1038/npg.els.0001882
doi: 10.1038/npg.els.0001882

[6] *DNA Replikation* url: http://www.pc.maricopa.edu/Biology/rcotter/Unit%202%20lessonbuilder/GrowthGenetics/index.html zuletzt besucht am: 09.07.2012

[7] *DNA Transkription* url: http://www.pc.maricopa.edu/Biology/rcotter/BIO%20205/LessonBuilders/Chapter%209%20LB/Ch9b5.html zuletzt besucht am: 09.07.2012

[8] *Genetischer Code* url: http://www.nature.com/scitable/topicpage/nucleic-acids-to-amino-acids-dna-specifies-935 zuletzt besucht am: 09.07.2012

Fehlertolerante Systeme in der autonomen Fahrzeugführung

Henning Elbers
Department für Informatik, Universität Oldenburg

***Zusammenfassung*—Die folgende Arbeit beschäftigt sich mit Ansätzen zur Fehlertoleranz am Beispiel autonomer Fahrzeugführung. Zu nennen sind hier Redundanz, Bussysteme, wie der CAN-Bus und es wird auch beleuchtet wie autonom fahrende Automobile aus Fehlern anderer Fahrzeuge via drahtloser Kommunikation lernen können. Auch die autonome Fahrzeugführung an sich und ihre Entwicklung, sowie aktuelle Forschung und praktische Anwendung werden im zweiten Teil vorgestellt. Die Arbeit versteht sich als kurzer Überblick über den Bereich der autonomen Fahrzeugführung und die damit verknüpften Möglichkeiten der Fehlertoleranz. Das Hauptaugenmerk liegt hier auf den drei Bereichen Redundanz, Fehlerdiagnose und Fehlerbehandlung, welche Ansätze für spätere praktische Anwendung geben sollen.**

***Keywords*-Fehlertoleranz; autonomes Fahren; sicherheitskritisch; Sensorik; DARPA Urban Challenge; HAVEit; Stadtpilot; kognitives Automobil; Redundanz; CAN-Bus; Fehlerdiagnose; Fehlerbehandlung**

I. Einleitung

Die Entwicklung des Automobils nahm seinen Anfang 1885/1886 durch seine Erfindung von Benz und Daimler. Die Innovationen im Automobilbereich sind Ende des 20. Jahrhunderts noch einmal gestiegen. Insbesondere ist die Sicherheit im Fahrzeug durch diverse Assistenzsysteme, wie dem Antiblockiersystem (ABS) oder dem elektronischen Stabilitätsprogramm (ESP) deutlich erhöht worden. Auch untrainierten Fahrern wird dadurch die Beherrschbarkeit des Autos erleichtert.

Die Steigerung der Sicherheit ist auf eine sich ständig weiterentwickelnde Fehlerbehandlungsstrategie zurück zu führen. Zu nennen sind in diesem Zusammenhang, z.B. im Bereich der Flugzeugindustrie oder der Raumfahrt, monolithische Entwurfsparadigmen[1] und die gleichzeitige redundante Auslegung sicherheitskritischer Systeme, wie reaktiver hard real time systems. Reaktiv, da bei Terminierung des Systems oder Programms die zeitkritische Funktionsfähigkeit nicht mehr sicher gestellt werden kann. Ein eingebettetes System oder ein Rechner mit Sicherheitscharakter muss fortlaufende Zustandswechsel durchführen und darf unter keinen Umständen einen Endzustand erreichen. Die Geburtsstunde von fehlertoleranten Architekturen und Rechensystemen wird in den 1960er Jahren gesehen, bei der Auflage des amerikanischen Raumfahrtprogramms [7].

Die Vision geht in Richtung hochautomatisiertem autonomen sicheren Fahren, bei dem der Fahrer vollständig entlastet wird und nur noch einen Zielort eingeben muss. So absurd es auch klingen mag, ein vergleichbares System waren im 19. Jahrhundert die Pferdekutschen, da das Pferd, auch ohne Kutscher, selbständig Hindernissen ausweichen und den Weg nach Hause finden konnte.

[1]Die Allokierung einer sicherheitskritischen Funktion auf genau einer ECU (electronic control unit)

II. Autonome Fahrzeugführung

Kognitive Automobile, also Automobile welche unabhängig kritische Entscheidungen treffen können (ohne menschlichen Einfluß), sind keineswegs mehr eine reine Zukunftsvision. Der Bereich der autonomen Fahrzeugführung ist schon seit Anfang des 21. Jahrhunderts ein etablierter Forschungszweig.

Unter anderem gibt es von der Deutschen Forschungsgemeinschaft (DFG) geförderte Projekte, wie der Entwurf einer visuellen Sensorplattform, die es durch verschiedene hochauflösende Videosensoren mit unterschiedlicher Brennweite ermöglichen soll, Objekte in großer, aber auch plötzlich auftretend in kurzer Entfernung zuverlässig wahrnehmen zu können. Dieses Projekt wird seit 2003 unterstützt und von Professor Dr.-Ing. Rüdiger Dillmann vom Karlsruher Institut für Technologie (KIT) geleitet [4].

Ein weiterer aktueller Forschungsbereich ist das Stadtpilot-Projekt der TU Braunschweig in Kooperation mit dem Deutschen Zentrum für Luft- und Raumfahrt. Hier hat ein Testfahrzeug namens „Leonie“ schon einige erfolgreiche Testfahrten auf dem Braunschweiger Stadtring hinter sich gebracht und hat dabei Manöver, wie Spur halten, Geschwindigkeit an den Verkehrsfluss anpassen, oder Interpretation von Ampelsignalen durchgeführt [12].

Aber nicht nur die Forschung an diesem Bereich ist allgegenwärtig. Es gibt auch viele praktische Anwendungen und Versuche zur autonomen Fahrzeugführung. Zwei Beispiele seien im Folgenden genannt:

1) DARPA Grand Challenge / Urban Challenge
2) HAVEit Final Event

Die DARPA Grand Challenge fand das erste Mal 2004 statt und umfasste eine 213 Kilometer lange Strecke durch die Mojave Wüste. Damals kam keines der teilnehmenden autonom fahrenden Autos an der Ziellinie an, welche durch alleinige Angabe der GPS-Koordinaten gefunden werden musste. 2007 wurde aus der Grand Challenge die Urban Challenge, die, wie der Name schon vermuten lässt, durch einen simulierten Stadtkurs führte. Aufgabe war es hier für die weltweit 53 teilnehmenden Fahrzeuge den 100 Kilometer

Kurs zu meistern, welcher komplexe Manöver wie Parken, Kreuzungen, insbesondere four-way-stop intersections und Einfädeln in den fließenden Verkehr umfasste. Dabei mussten die kognitiven Automobile auch mit von Menschen gesteuerten Fahrzeugen interagieren. Das Preisgeld für den ersten Platz betrug 2 Millionen Dollar [3, 6].

HAVEit[2] steht für den Zusammenschluß 17 europäischer Partner der Automobilindustrie, die das Fahren sicherer, umweltfreundlicher und komfortabler gestalten wollen. Diese Vision soll durch das autonom agierende Fahrzeug realisiert werden. Anders als bei der Urban Challenge liegt hier der Schwerpunkt mehr auf einer Unterstützung des Fahrers, als auf völliger Autonomität. Allerdings ist auch ein eigenständig handelnder Autopilot eingeplant. Die Entwicklungen und Ergebnisse wurden dann am 21. Juni 2011 beim Final Event in Borås (Schweden) auf dem Volvo Testgelände präsentiert. Es nahmen 7 Autos teil [9].

Der Anteil an Autonomität bei kognitiven Automobilen ist also variabel. Er reicht von unterstützenden Systemen, wie dem teilweisen Autopilot, bis hin zu vollständiger Autonomie, also Fahren ohne menschlichen Fahrzeugführer. Allen Fahrzeugen gemein ist aber, dass sie eine ausgeprägte Sensorik besitzen und eine Art künstliche Intelligenz, im Sinne von eigenständigem Treffen von Entscheidungen auf Basis gesammelter Sensordaten und anschließendem Handeln. Da in beiden Fällen durch Fehlfunktionen Menschenleben gefährdet werden, muss es wirkungsvolle, zuverlässige Fehlertoleranzkonzepte geben, auf die im nächsten Abschnitt eingegangen werden soll. Dabei wird zur Orientierung der „Überblick Fehlertoleranz“ verwendet [13].

III. Fehlertoleranzkonzepte

A. Motivation

Anhand von Tabelle I wird klar, welchen wichtigen Stellenwert eine Verbesserung der Sicherheit im Straßenverkehr hat. Die Anzahl der Toten und auch der Verletzten kann durch ein gut funktionierendes autonomes System der Fahrzeugführung gesenkt und auch menschliche Fehler können vermieden werden.

B. Fehlertoleranz durch Redundanz

Da ein autonom agierendes Fahrzeug somit gerade den Sicherheitsaspekt im Fahrzeug steigern soll, darf durch Versagen von einzelnen Subkomponenten nicht das System als Ganzes die Funktionsfähigkeit verlieren. Um dieses Szenario zu vermeiden ist Fehlertoleranz durch Redundanz eine Möglichkeit. Zwei Ansätze sind maskierende Hardware (strukturelle Redundanz) und diversitäre Software (funktionelle Redundanz) [5]. Beide Redundanzen sind sowohl in der Raumfahrt, als auch in der Flugzeugindustrie angewandte und bewährte Praxis [10]. Da autonom fahrende Fahrzeuge, wie vorher fest gestellt, auf Sensoren angewiesen sind und

[2]Highly automated vehicles for intelligent transport

Tabelle I
Verkehrstote in Deutschland März 2011 - März 2012 nach [1]

Monat	**Verkehrstote**
März 2011	305
April 2011	341
Mai 2011	374
Juni 2011	349
Juli 2011	348
August 2011	380
September 2011	395
Oktober 2011	380
November 2011	319
Dezember 2011	364
Januar 2012	266
Februar 2012	251
März 2012	252
Gesamt	4324

deren Daten zuverlässig auswerten müssen, bietet sich als eine Fehlerbehandlungsstrategie die Triple Module Redundancy (TMR) an (siehe Abbildung 1). Hierbei handelt es sich um eine redundant ausgelegte Hardware. Die Datenverarbeitungsanlagen (Modules), welche aus den Sensordaten die notwendigen Anweisungen an die Aktorik (beim Auto z.B. Bremsen oder Motor) weiterleiten, werden dreifach ausgelegt, dabei wird automatisch die implementierte Software ebenfalls verdreifacht, um Ausfällen in beiden Bereichen gleichermaßen vorzubeugen. Zusätzlich werden die zwischengeschalteten Voter, welche die Daten der Modules auf Unstimmigkeiten überprüfen ebenfalls redundant eingesetzt. Dadurch wird eine Fehlfunktion eines Modules, wie auch die eines Voters kompensiert. Das resultierende TMR System arbeitet dabei nach dem 2-aus-3-Prinzip (statisch strukturelle Redundanz [5, 13]), d.h. bei unterschiedlichen Ergebnissen wird durch Mehrheitsentscheidung ein Ergebnis weiter propagiert. Die drei Outputs werden dann zu einem Signal vereint, welches an die Aktorik weiter gegeben wird. Diese einzelnen TMR Bausteine können auch selbst redundant ausgelegt werden, dabei überprüft ein weiterer Voter die Voter der verschalteten Bausteine (Pipeline Prinzip) [2].

Bei erhöhten Sicherheitsanforderungen kann auch ein 3-aus-5-System verwendet werden, was nach demselben Prinzip arbeitet nur eine erhöhte Ausfallsicherheit bietet [13]. Sinnvoll wäre es auch die Sensoren mehrfach zu verbauen, wobei allerdings der Kostenfaktor eine entscheidende Rolle spielt. Das ist auch der Grund, wieso in heutigen Fahrzeugen noch keine Fehlertoleranzsysteme in einem solchen Umfang wie in Flugzeugen eingesetzt werden. Das Kosten-Nutzen-Verhältnis ist zu gering, da die meisten Unfälle ursächlich auf menschliches Versagen zurück zu führen sind. Dieser Umstand würde bei kognitiven Automobilen natürlich nicht mehr greifen.

Diversitäre Software, also Programme, die für dieselbe Aufgabe vorgesehen sind, aber unterschiedlich implementiert wurden, wäre in der autonomen Fahrzeugführung eine weitere Lösung aus dem Bereich der Redundanz. Der typi-

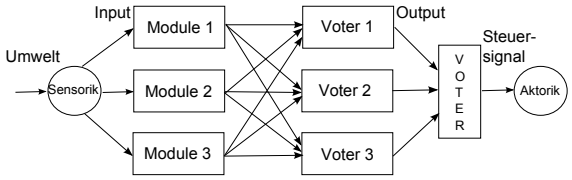

Abbildung 1. TMR System am Beispiel autonomer Fahrzeugführung

sche Ablauf für Software in sicherheitsrelevanten eingebetteten Systemen in kognitiven Automobilen wäre vereinfacht dargestellt, wie auch in Abbildung 1 zu erkennen:

$$Sensordaten \Rightarrow Auswertung \Rightarrow Steuersignale$$

Dieser Ablauf impliziert eindeutig ein Daten intensives (data-intensive) und kein zustandsorientiertes (control dominated) Entwurfsparadigma [2]. Diversitäre Software kann gerade bei Daten intensiven Programmen wirklich optimal seine fehlerkompensierende Wirkung ausspielen, da die Datenauswertung unterschiedlich vorgenommen werden kann. Ein weiterer wichtiger Vorteil bei dieser Art von funktioneller Redundanz ist aber auch die Vermeidung von Entwurfsfehlern, die durch die Programmierer entstehen könnten (unabhängiger Entwurf, gegensätzlicher Entwurf) [5].

C. Fehlerdiagnose durch Kommunikation

Im Folgenden soll auf die Fehlerdiagnose eingegangen werden, die, wie Redundanz und Fehlerbehandlung auch, ein weiterer Bereich der Fehlertoleranz ist [13].

In Automobilen werden heute schon Fahrzeugdiagnosesysteme eingebaut, die bei einem Fehler nach folgendem Prinzip vorgehen:

$$Erkennung \Rightarrow Diagnose \Rightarrow Speicherung$$

Der Vorteil liegt hier in der Speicherung der erfassten Daten und der nachträglichen Auswertung mit dem Ziel diesen Fehler in Zukunft zu vermeiden. Der Speicher kann z.B. in einer KFZ-Werkstatt ausgelesen werden. Der Nachteil dieses Systems ist, dass die Information vorerst nur den speziellen Fahrzeugführer und seine nahe Umgebung erreicht.

Bei der autonomen Fahrzeugführung kann durch Ausstattung der kognitiven Automobile mit drahtlosen Übertragungsmedien dieser Verbreitungsprozess effektiver und schneller gestaltet werden. Der Mensch kann von dieser Information vollständig abstrahieren, da das Fahrzeug im Optimalfall vor Auftreten des Fehlers den Grund für diesen schon kennt und somit schon vor der eigenen $Fehlererkennung$ die $Fehlerkorrektur$ eingeleitet hat. Dieses Ausstattungsmerkmal könnte natürlich schon heute auf die Autos übertragen werden, allerdings kann das Auto hier nicht selbstständig handeln, sondern müsste die Information dem Menschen zur Verfügung stellen und der wäre für die $Fehlerkorrektur$ zuständig. Der einschränkende Faktor ist hier der Mensch, der unzuverlässiger und langsamer arbeitet als ein autonomes System und keinen sofortigen parallelen Zugriff auf alle Subsysteme hätte.

Bei autonomen Systemen würde der Fehler im Optimalfall also einmalig auftreten und dann von anderen Systemen durch die Informationsübertragung kompensiert werden, sodass folgendes Prinzip der Fehlerdiagnose schon vor Auftreten des Fehlers selbst greift:

$$Information \Rightarrow Kompensierung \Rightarrow Korrektur$$

Dieses Prinzip lässt sich natürlich nicht nur auf Systemfehler anwenden, sondern kann auch an gefährlichen Orten mit undurchsichtiger Verkehrssituation Anwendung finden. Wenn das Auto einen Unfall hatte und das Diagnosesystem noch funktionsfähig ist kann es Autos in naher Umgebung die Koordinaten mitteilen und somit können diese dann an dieser speziellen Stelle z.B. langsamer fahren.

D. Fehlerbehandlung durch den CAN-Bus

In heutigen Autos sind schon weit über 100 Mikroprozessoren oder auch eingebettete Systeme verbaut, die wichtige Fahrzeugfunktionen regeln [14]. Diese einzelnen Subkomponenten müssen oft auch untereinander kommunizieren und das möglichst verlust- und fehlerfrei. Dafür wird heutzutage schon eine Technik verwendet, die auch in der autonomen Fahrzeugführung ein stabiles Fehlerbehandlungskonzept repräsentieren kann - das Bussystem. Ein im Fahrzeug oft verwendeter Bus ist der sogenannte CAN-Bus, der das erste Mal 1986 von Bosch auf dem SAE Kongress in Detroit unter dem Namen „Automotive Serial Controller Area Network“ vorgestellt wurde und bereits 2 Jahre danach, 1988, als Serienchip zur Verfügung stand [8]. Er ist der Fehlerkompensierung zuzuordnen und zwar insbesondere dem Bereich der Fehlerkorrektur [13].

Jede über den CAN-Bus zu übertragende Nachricht hat ein bestimmtes fest definiertes Aussehen. Die Bitfolge der Nachricht ist in genau 8 Teile unterteilt: Start of Frame, Arbitration Field, Control Field, Data Field, CRC Field, ACK Field, End of Frame und den Interframe Space (IFS) (siehe Abbildung 2). Die für die Fehlerkorrektur entscheidenden Felder werden im Folgenden erläutert: Durch das CRC Field wird per Prüfsumme der Inhalt des Data Fields auf Fehler untersucht. Beim Finden einer Unstimmigkeit im Bitmuster belegt der Empfänger der korrupten Nachricht das ACK Field mit einer von R|D[3] abweichenden Bitkombination um dem Sender einen Übertragungsfehler mitzuteilen. Der Sender wird daraufhin die Nachricht bei nächster Arbitrierung erneut senden, woraufhin der Empfänger diese wieder überprüft [2, 8].

Durch diese Art der Fehlerkorrektur können bis auf

$$Fehlerrate \cdot 4,7 \cdot 10^{-11} \quad (1)$$

[3]R: Recessive Bit (z.B.: 0), D: Dominant Bit (z.B.: 1)

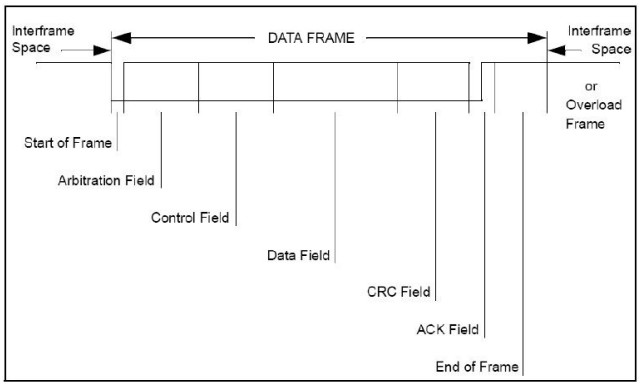

Abbildung 2. CAN Bus Data Frame nach [2]

korrupte Nachrichten erkannt und behoben werden [2]. Die $Fehlerrate$ ist hier der Durchschnitt an Fehlern bei der betrachteten Nachrichtenart. Somit können Fehler in Übertragungen fast vollständig verhindert werden, was insbesondere für kognitive Automobile aufgrund der hohen Anzahl an Sensoren und der daraus resultierenden Sensordaten einen wichtigen Stellenwert besitzt.

IV. Ausblick

Um einen erhöhten Sicherheitsstandard zu gewährleisten wäre es zweckdienlich mehrere Fehlertoleranzkonzepte zu kombinieren. Beispielsweise kann ein kognitives Automobil mit TMR-System mit Hilfe des Pipeline Prinzips mehrere Bausteine hintereinander schalten, oder zusätzlich die Kommunikation über ein Bussystem führen, wie den CAN-Bus oder über FlexRAY[4]. Für weitere, neuartige Fehlertoleranzkonzepte nicht nur im Automobilsektor kann auch der Mensch als Vorbild herangezogen werden. Es gibt schon Ansätze wie man aus menschlichem Verhalten konstruktive Anregungen für die Realisierung eines technischen autonomen Systems ableiten kann. Dabei beruft man sich hauptsächlich auf drei Ebenen des menschlichen Verhaltens: Die fertigkeitsbasierte Ebene, die regelbasierte Ebene und die wissensbasierte Ebene [11].

Die Zukunftsaussichten von autonomen Systemen im Straßenverkehr hängen aber auch stark von kulturellen und juristischen Gesichtspunkten ab, denn es muss eine klare gesetzliche Regelung geben, wer im Falle eines Unfalls haftbar gemacht wird, wenn das autonom geführte Auto für den Unfall verantwortlich ist. Nichtsdestotrotz werden sich autonome Systeme in Zukunft durchzusetzen wissen, da renommierte international erfolgreiche Automobilhersteller (VW) schon heutzutage viel Geld und Testfahrzeuge für die Forschung und den praktischen Einsatz bereit stellen [12].

[4]FlexRAY ist ein Bussystem, welches sowohl Ereignis gesteuert, als auch Zeit gesteuert arbeitet [2].

Literatur

[1] Statistisches Bundesamt. *Anzahl der bei Straßenverkehrsunfällen Getöteten in Deutschland von März 2011 bis März 2012*. Website. 2012 (letzter Zugriff: 05.06.2012). URL: http://de.statista.com/statistik/daten/studie/161724/umfrage/verkehrstote-in-deutschland-monatszahlen/.

[2] Prof. Dr. Werner Damm. *Architectural Principles 1 und 2*. Foliensatz Eingebettete Systeme II. Universität Oldenburg: Institut für Informatik. SS/2012.

[3] Darpa. *Urban Challenge*. Website. 2012 (letzter Zugriff: 05.06.2012). URL: http://archive.darpa.mil/grandchallenge/index.asp.

[4] DFG. *Visuelle Sensorplattform für kognitive Automobile*. Website. 2003 (letzter Zugriff: 11.06.2012). URL: http://gepris.dfg.de/gepris/OCTOPUS/.

[5] Prof. Dr. Klaus Echtle. *Redundanz*. 2012. URL: http://dc.informatik.uni-essen.de/Echtle/all/buch_ftv/F05_Kapitel03.pdf.

[6] Autonomes Fahren. *DARPA Grand Challenge & Urban Challenge Test für Autonome Fahrzeuge*. Website. 2012 (letzter Zugriff: 05.06.2012). URL: http://www.autonomes-fahren.de/darpa-grand-challenge-urban-challenge-test-fur-autonome-fahrzeuge/.

[7] Felix C. Gärtner. *Formale Grundlagen der Fehlertoleranz in verteilten Systemen*. Website. 2012 (letzter Zugriff: 02.06.2012). URL: http://subs.emis.de/LNI/Dissertation/Dissertation2/GI-Dissertations.02-4.pdf.

[8] Prof. Dr. Ing. Andreas Grzemba. *MOST: Das Multimedia-Bussystem für den Einsatz im Automobil*. Seite 339-343. Franzis Verlag, 2007.

[9] HAVEit. Website. 2011 (letzter Zugriff: 05.06.2012). URL: http://www.haveit-eu.org/displayITM1.asp?ITMID=6&LANG=EN.

[10] Hubert Kirrmann und Karl-Erwin Großpietsch. *Fehlertolerante Systeme in der Automatisierungstechnik*. Techn. Ber. Systemic Modeling Laboratory (LAMS), Aug. 2002.

[11] Markus Maurer. "Flexible Automatisierung von Straßenfahrzeugen mit Rechnersehen". Diss. Universität der Bundeswehr München, Juli 2000.

[12] Tobias Nothdurft u. a. *Stadtpilot: First Fully Autonomous Test Drives in Urban Traffic*. Techn. Ber. Braunschweig: Inst. of Flight Guidance, Inst. of Control Engineering. - Hildesheim: Inst. of Physics und Technology, Feb. 2012.

[13] Prof. Dr.-Ing. Oliver Theel. *Fehlertolerante Systeme*. Foliensatz Seminar Fehlertolerante Systeme. Universität Oldenburg: Institut für Informatik: Systemsoftware und verteilte Systeme. SS/2012.

[14] Prof. Dr.-Ing. Werner Zimmermann. *Embedded Systems - Was der Elchtest mit Informationstechnik zu tun hat*. Techn. Ber. Hochschule Esslingen.