# Was ist möglich und was nicht: Theoretische Grenzen der byzantinischen Fehlertoleranz bei verteilten Anwendungen.

Mykhaylo Varshavskyy
simyvars@stud.informatik.uni-erlangen.de

**Kurzzusammenfassung**

Dieses Papier befasst sich mit Lösungen vom Einigungs-Problem im Hinblick auf die Fehlertoleranz. Insbesondere wird gezeigt, dass die Tolerierung von m fehlerhaften Knoten erfordert mindestens 3m+1 Knoten insgesamt. Eine andere Unmöglichkeitsaussage bezieht sich auf den asynchronen Fall. Es gibt kein asynchrones Protokoll, das das Vorkommen eines einzigen Fehlers tolerieren kann.

## 1 Einführung

### 1.1 Motivation

Eines der grundlegenden Probleme von Kommunikation zwischen verteilten Anwendungen ist das Erreichen einer Einigung zwischen den beteiligten Prozessen.

Ein Beispiel dafür ist das in den Datenbanksystemen auftretende “transaction commit problem”. Eine Menge von Prozessen beteiligt sich an einer Transaktion. Nun soll das Resultat der Transaktion auf die gemeinsame Datenbank angewendet werden, oder beim Auftreten von Problemen soll die Transaktion ignoriert werden. Unabhängig von der konkreten Entscheidung müssen alle Datenmanager die gleiche Entscheidung treffen.

Sollten alle Prozesse sowie das Netz zuverlässig sein, ist das Erreichen einer Entscheidung naheliegend. Aber in einem realen System können verschiedene Fehler auftreten. Deshalb soll das Einigungsprotokoll das Auftreten von möglichen Fehlern tolerieren.

### 1.2 Grundsätzliches

Formal gesehen, wird die Situation durch folgendes Modell beschrieben:

Es sind insgesamt n Prozesse, davon aber höchstens m fehlerhaft. Das System ist voll vermascht. Zu jeder Nachricht ist sowohl Absender als auch Empfänger bekannt. Die Kommunikation ist zuverlässig. Die Prozesse sollen eine Einigung zwischen 2 Werten (0 und 1) erzielen. Die Abarbeitung erfolgt entweder synchron oder asynchron.

Asynchrone Verarbeitung bedeutet, dass keine Annahmen über die relativen Geschwindigkeiten von Prozessen oder über die Verzögerungszeit beim Liefern einer Nachricht gemacht werden. Die Prozesse haben auch keinen Zugang zu Uhren, so dass die auf Auszeiten basierende Algorithmen nicht angewendet werden können. Es ist auch unmöglich den Tod eines Prozesses festzustellen, so dass es unmöglich zu sagen ist, ob ein Prozess gestorben ist oder nur sehr langsam läuft.

arbeit.fm: 27.05.02 10:54

# 2 Unlösbarkeitsaussagen

## 2.1 Unlösbarkeit bei asynchronen Nachrichten

### 2.1.1 Systemmodell

Jeder von n Prozessen besitzt einen Anfangswert (0 oder 1). Es werden 2 Operationen an das Kommunikationssystem angewendet.

send(p,m)-Übergabe der Nachricht m an das Kommunikationssystem für den Prozess p als Empfänger.

recieve(p)- Entnahme von (p,m) aus dem Nachrichtensystem oder leeres Ergebnis

### 2.1.2 Bezeichnungen

Eine Konfiguration beschreibt die Zustände aller Prozessoren und des Nachrichtensystens.

Ein Schritt überführt eine Konfiguration in die andere durch Ausführung von recieve, send oder einem lokalen Übergang.

Ein Übergang wird durch ein Ereignis e=(p,m) bestimmt. Eine Konfiguration C geht dabei in die Konfiguration e(C) über.

Ein Ablaufplan s ist eine Folge von Ereignissen, die auf C angewandt werden kann.

Lokales Ergebnis ist der Wert, wofür sich der Prozessor unwiderruflich entschieden hat.

### 2.1.3 Beweis der Unlösbarkeitsaussage.

Lemma 1.

Ausgehend von der Konfiguration C führe s1 bzw. s2 zu C1 bzw. C2. Wenn die Prozesse, die in s1 einen Schritt ausfüren, verschieden sind von denen, die in s2 einen Schritt ausführen, dann ist das Ergebnis von der Anwendung s2 auf C1 und s1 auf C2 gleich.

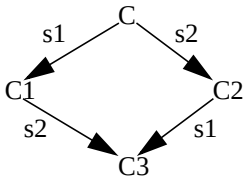


**Abb. 2.1.3.1 Verdeutlichung vom Lemma1.**

Eine Konfiguration C heisst bivalent, wenn die Menge lokaler Ergebnisse von C erreichbarer Konfigurationen enthält sowohl 0 als auch 1. Ansonsten heisst die Konfiguration 1- bzw.0-valent.

Lemma 2.

Es gibt eine bivalente Anfangskonfiguration.

Lemma 3.

Sei C eine bivalente Konfiguration und e=(p,m) ein in dieser Konfiguration anwendbares Ereignis. Sei Q die Menge von Konfigurationen, die von C aus ohne e erreicht werden können, und sei R=e(Q). Dann enthält R eine bivalente Konfiguration.

Satz

Es existiert kein Protokoll, das unter Verwendung asynchroner Nachrichten das Einigungsproblem lösen kann, wenn nur ein Prozess fehlerhaft ist.

Annahme: Es existiert ein Protokoll, das auch bei einem fehlerhaften Prozess das Einigungsproblem löst.

Wenn eine Konfiguration in einem Schritt in eine andere überführt werden kann, dann heissen die Konfigurationen Nachbarn.

Seien C0 und C1 Nachbarn, so dass Di=e(Ci) i-valent ist.

Sei C1=e'(c0) mit e'=(p',m').

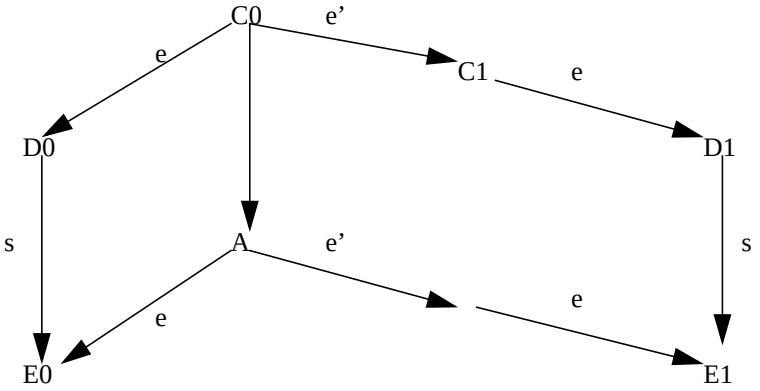


**Abb.2.1.3.2 Zum Beweis des Satzes**

Wenn p'und p verschieden sind, dann gilt nach Lemma1 D1=e'(D0), aber das ist unmöglich, denn ein Nachfolger einer 0-valententen Konfiguration kann nicht 1-valent sein.

Wenn p=p', dann sei s ein Ablaufplan, wo sich der Prozess P nicht beteiligt und A=s(C0). s kann auch auf Di angewendet werden: Ei=s(Di) - die entsprechende i-valente Konfiguration.

Wegen e(A)=E0 und e(e'(A))=E1 muss A bivalent sein, aber es widerspricht der Annahme, dass A ein bis zur Entscheidung geführter Lauf ist.

Somit ist die Annahme falsch.

## 2.2 Unlösbarkeit bei vielen fehlerhaften Prozessen

Unter einem System wird ein Knotengraph verstanden, bei dem jedem Knoten eine Einrichtung, die sein Verhalten erzeugt, und eine Eingabe zugeordnet wird. Ein Systemverhalten wird durch das Verhalten von Knoten und Verbindungen beschrieben. Die Restriktion von E auf Knoten und Verbindungen eines Untergraphen Gu bildet das Szenario Eu von Gu.

Ein Graph S überlagert einen Graphen G, wenn eine Abbildung von Knoten von S auf G existiert, so dass wenn Knoten u aus S mit k Knoten v1,...,vk verbunden ist und auf w in G abgebildet wird, dann ist Knoten w auch mit k Knoten x1,..,xk verbunden. Knoten vi auf Knoten xi dabei abgebildet wird für 1<=i<=k.

Satz

In einem System mit n Prozessen, wovon maximal m fehlerhaft sind, existieren keine Protokolle zum Lösen des Byzantinischen Einigungsproblems, wenn n<=3m ist.

Beweis:

Sei n=3 und m=1

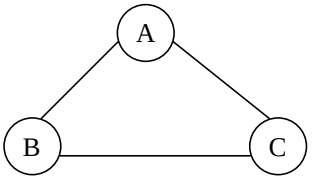


**Abb.2.2.1 Graph G.**

Annahme: Die Einrichtungen A,B und C können sich im gezeichneten Graphen immer einigen.

In der Überlagerung S wird den Knoten a und a' die Einrichtung A, den Knoten b und b' die Einrichtung B, den Knoten c und c' die Einrichtung C zugeordnet.

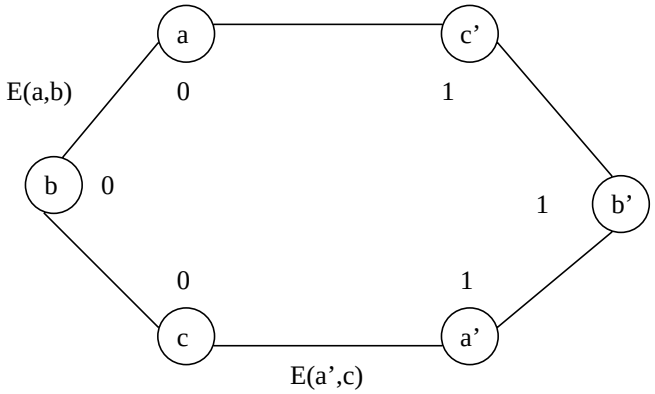


**Abb.2.2.2 Überlagerung S**

Fehleraxiom:

Sei A eine Einrichtung und E1,..,Ek das Verbindungsverhalten, so dass Ei das Verbindungsverhalten des i-ten Ausgangs eines Knoten mit zugeordneter Einrichtung A ist. Sei u ein Knoten mit k Ausgängen, dann existiert eine Einrichtung F, so das in jedem System, in dem u die Einrichtung F zugeordnet ist, das Verhalten der Ausgangskanten (u,vi) gleich Ei ist. Dies wird im folgenden mit FA(E1,...,Ek) beschrieben.

Nach Fehleraxiom existiert eine Einrichtung FA(E(a,b),E(a',c)), die für E(a,b) das gleiche Eingangsverhalten erzeugt wie das System S.

Lokalitätsaxiom:

Seien S und S' Systeme mit Verhalten E bzw. E' und isomorphen Untersystemen U bzw. U'.Wenn das Verhalten korrespondierenen Eingänge von U und U' in den Szenarien Eu und E'u gleich ist, dann sind auch die Szenarien Eu und E'u gleich.

Wird die Einrichtung A durch FA(E(a,b),E(a’,c)) ersetzt, dann ist nach Lokalitätaxiom E(b,c)=E(B,C).,

Dann entscheiden sich B und C sowie b und c für den Wert 0. Analoge Betrachtung für E(c,a’) führt dazu, dass auch a’ sich für 0 entscheiden muss. Wiederholung der Betrachtung für a’ und b’ führt zum Schluss, dass auch b’ zum Ergebnis 0 kommen muss. Anderseits analog zum anfänglichen Betrachtung, müssen sich Knoten a’ und b’ auf den Wert 1 einigen. Wiederspruch zeigt, dass die Annahme falsch ist.

Wenn n<=3m, dann wird die Knotenmenge in den Teilmengen a,b und c partitioniert. Jede Teilmenge enthält höchstens m Elemente. Seien A,B und C die entsprechenden Einrichtungen von Knoten in a,b und c, die das Problem lösen.

Es wird eine Überlagerung durch Verdoppelung des Graphen konstruiert. Mit der gleichen Argumentation wie bei n=3 bei der Ersätzung der Knoten durch die Knotenmengen, wird A wieder durch eine Einrichtung F mit Fehlverhalten ersätzt. Die gleiche Argumentation führt wieder zum Widerspruch.

## 2.3 Weitere Unmöglichkeitsaussage.

Die Tolerierung von m möglichen Fehlern erfordert die Verbundenheit von mindestens 2m+1. Unter der Verbundenheit wird die zu entfernende Anzahl von Knoten verstanden, damit ein Graph in zwei Teile zerfällt. Zeigen wir dieses Resultat für den Fall m=1.

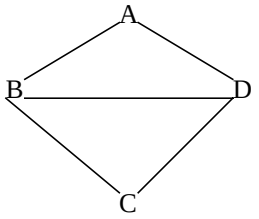


**Bild 2.3.1 Graph G**

Die Verbundenheit von G ist 2, denn B und D trennen G in zwei Teile: A und C. Es wird das folgende System betrachtet: Graph S mit 8 Knoten

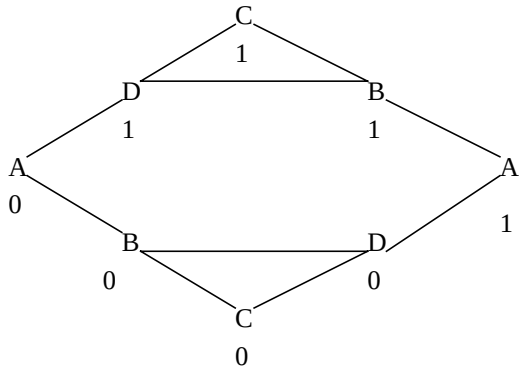


**Bild 2.3.2 Graph S**

Das resultierende Verhalten des Systems sei V. Das Szenario V1 ist auf dem folgenden Bild ersichtlich.

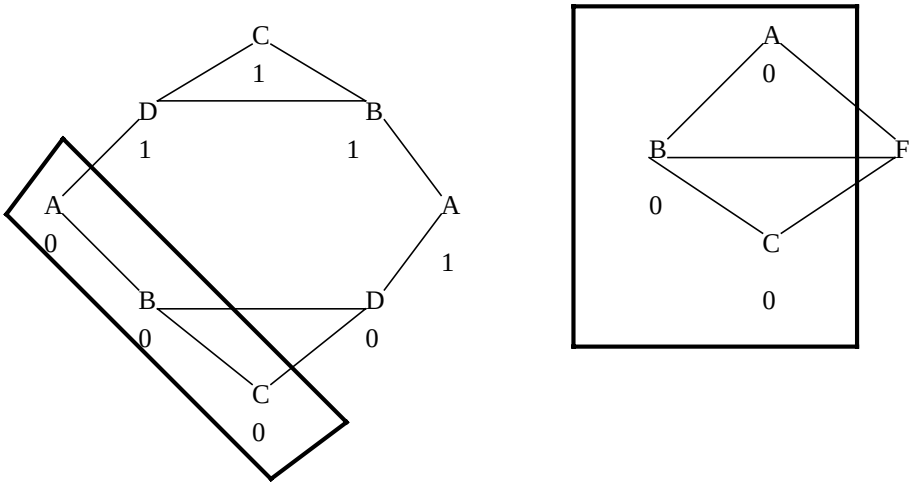


**Bild 2.3.3 Szenario V1**

A,B und C sind in dem Szenario V1 fehlerfrei. D ist fehlerhaft. D zeigt das eine Verhalten für A und das andere Verhalten für B und C. Dann müssen A, B und C zum Ergebnis 0 kommen.

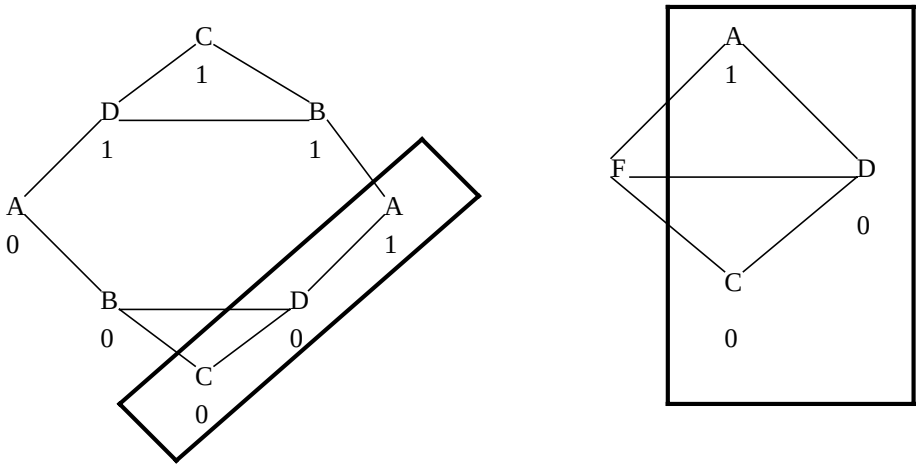


**Bild 2.3.4 Szenario V2**

Im zweiten Szenario ist B fehlerhaft.Die Einrichtung B zeigt ein Verhalten zu C und D und ein anderes zu A. Da C sich für 0 entschieden hat, müssen sich A,C und D auf 0 einigen.

Im dritten Szenario sind A, B und C fehlerfrei, aber sie haben 1 als Eingabe. D ist hier fehlerhaft.Dann erfolgt Einigung auf dem Wert 1. Dies widerspricht der Tatsache, daß sich A für 0 entscheiden muß.

Für den allgemeinen Fall können die gleichen Bilder benutzt werden. Die B und D müssen dann je aus m Knoten bestehen, so dass die Entfernung von B und D zertrennt G in zwei nichtleere Mengen A und C.

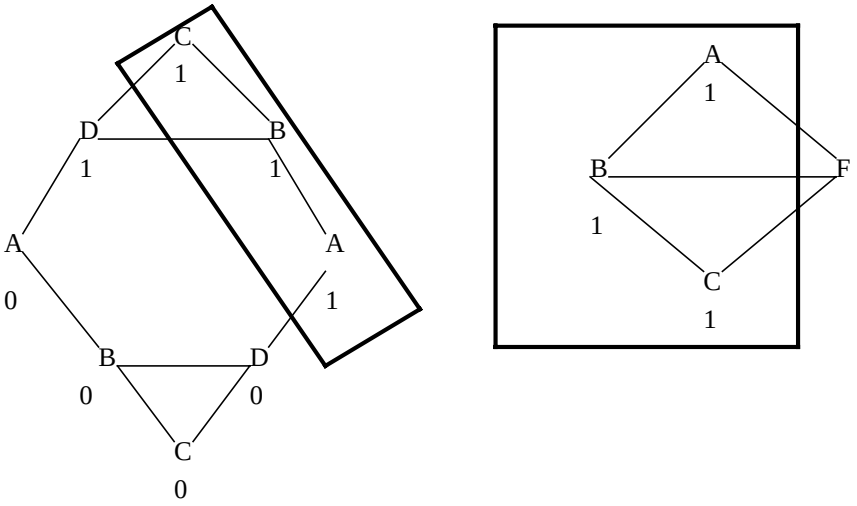


**Bild 2.3.4 Szenario V3**

**Literaturverzeichnis**


FIS85. M.I.Fisher, N.A.Lynch, M.S Robertson, “Impossibility of Distributed Consensus with One Faulty Process”, Journal of the Association for Computing Machinery, vol.32, no.2, pp.374-382, April 1985.

FIS86. M.I.Fisher, N.A.Lynch, M. Merritt, “Easy Impossibility Proofs for Distributed Consensus Problem”, Distributed Computing, vol.1, 1986.

LYN89. N.A.Lynch, “A Hundret Impossibility Proofs for Distributed Computing”, ACM 0-89791-326-4/89/008/001, 1989.