Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Institut für praktische Informatik und Medieninformatik
Fachgebiet Telematik
Prof. Dr. Dietrich Reschke

Hauptseminar Telematik
Sommersemester 2003

Konsistenzerhaltung in dezentralen verteilten Systemen

vorgelegt von:
Drießel, René
Bahnhofstraße 19b
98692 Ilmenau
Tel.: +49-(0)170-5222516
E-Mail: studium@driessel.de
Matr.-Nr.: 28694

Inhaltsverzeichnis

1 Einleitung

Dezentrale verteilte Systeme werden schon seit einiger Zeit immer wieder in der Öffentlichkeit kontrovers diskutiert. Die Gründe für diese Diskussionen sind die gesellschaftlichen Umbrüche, die durch die Anwendung dieser Systeme ausgelöst werden.

Auf der anderen Seite ist mit verteilten Systemen im Allgemeinen und dezentralen verteilten Systemen im Besonderen eine Erhöhung der Ausfallsicherheit und Reduzierung der Kosten möglich. Der Einsparungseffekt resultiert dabei hauptsächlich aus der Möglichkeit Ressourcen, wie Drucker und Dateien, gemeinsam nutzen zu können. Das Internet als größtes verteiltes System der Welt wurde speziell im Hinblick auf die Ausfallsicherheit und die gemeinsame Nutzung von Ressourcen entwickelt.

Obgleich dezentrale verteilte Systeme gegenüber zentralen verteilten Systemen Vorteile haben, ist die Entwicklung von dezentralen verteilten Systemen häufig kompliziert. Speziell die Forderung nach Konsistenzerhaltung macht die Implementierung sehr schwierig. Aus diesem Grund findet man in der Praxis mehr zentrale als dezentrale Ansätze für verteilte Anwendungen.

Als Hauptanwendungsgebiete von dezentralen verteilten Systemen haben sich der so genannte Dateitausch, Instant Messaging und das verteilte Rechnen herauskristallisiert.

- Bei Dateitauschdiensten tauschen die Anwender Dateien über das Netzwerk. Für eine höhere Verfügbarkeit werden (wie bei verteilten Dateisystemen) die einzelnen Dateien häufig auf anderen Rechnern zwischengespeichert.

- Mit Hilfe von Instant Messaging Systemen, wie das freie Jabber [2] können Nutzer in Echtzeit Kurznachrichten und Dateien austauschen.

- Die Möglichkeit, Berechnungen zu parallelisieren und verteilte durchzuführen, ist eines der vielversprechendsten Anwendungen.

Speziell im Bereich des verteilten Rechnens und des Dateitauschs sind Konsistenzerhaltungsaspekte sehr wichtig. Innerhalb dieser Arbeit soll ein Überblick über die verschiedenen Konsistenzbegriffe sowie über die bestehenden Verfahren zur Konsistenzerhaltung in dezentralen verteilten Systemen gegeben werden.

2 Topologien verteilter Systeme

Es existieren verschiedene Möglichkeiten des Aufbaus von verteilten Systemen. Diese werden unter dem Begriff der Topologie zusammengefasst. Die topologische Betrachtung kann dabei auf verschiedenen Ebenen erfolgen: physisch, logisch, verbindungsorientiert oder organisatorisch. Die einzelnen Möglichkeiten der Konsistenzerhaltung werden von der Topologie entscheidend bestimmt. Laut Minar [6] existieren 4 verschiedene topologische Grundtypen von verteilten Systemen. Wie allerdings im Folgenden zu sehen ist, kann man diese 4 Grundtypen auf 2 reduzieren.

2.1 Zentrale verteilte Systeme

Diese Art von verteilten Systemen lässt sich am einfachsten implementieren. Sie basieren auf einer typischen Client-Server-Architektur. Alle benötigten Daten und Funktionen sind in einem zentralen Server gespeichert. Die Clients empfangen die für sie relevanten Informationen von dem zentralen Server. Eines der bekanntesten zentralen Systeme ist Seti@Home [7]. Der zentrale Server funktioniert als eine Art Job Dispatcher für die einzelnen Clients.

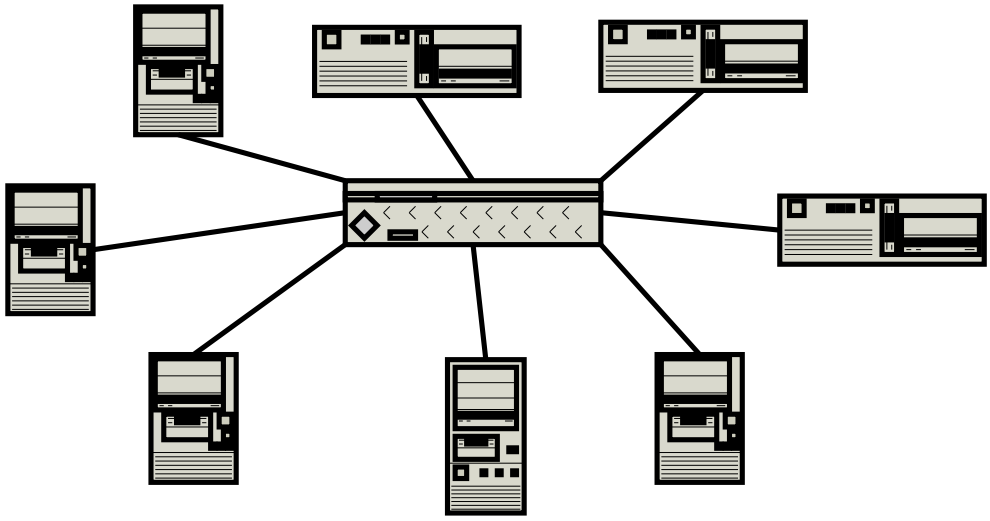

Abbildung 1: zentrales verteiltes System

2.2 Dezentrale verteilte Systeme

Ein Gegenentwurf zu dem typischen Client Server System sind die dezentralen verteilten Systeme. Bei diesem Topologieentwurf ist jeder Knoten des Netzes

prinzipiell gegenüber den anderen Knoten gleichberechtigt. Daraus folgt die Notwendigkeit, dass jeder einzelne Knoten des Netzes ein so genannter Servant (sowohl Client als auch Server) ist. Das bekannteste dezentrale verteilte System ist sicherlich das Dateitauschprotokoll Gnutella. Auch verschiedene Multiagentensysteme erheben den Anspruch dezentrale verteilte Systeme zu sein.

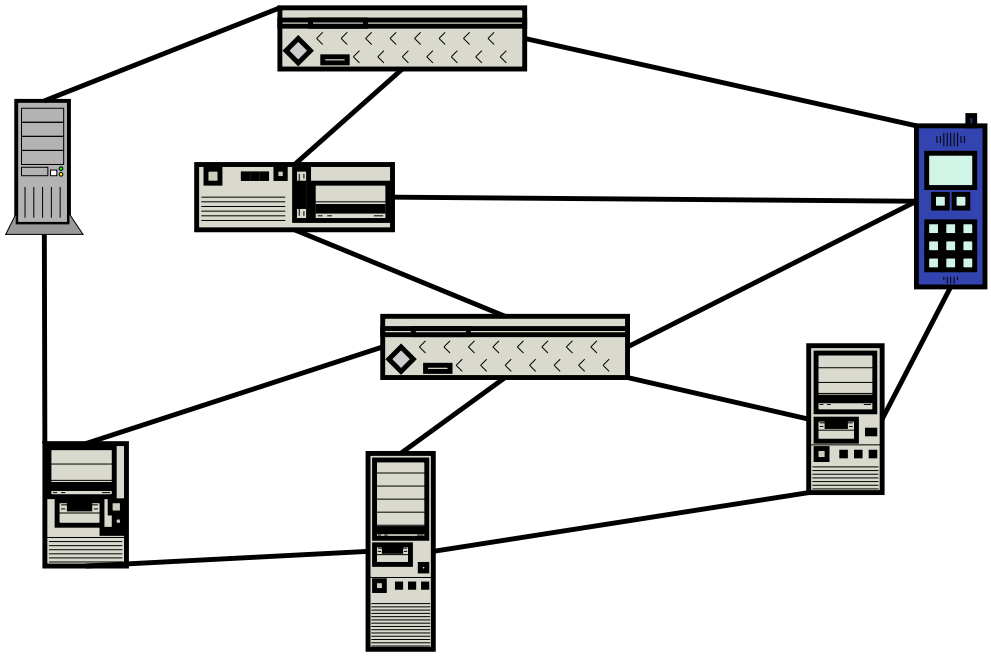

Abbildung 2: dezentrales verteiltes System

2.3 Hierarchische Systeme

Ein hierarchisches System kann man als eine Erweiterung eines zentralen verteilten Systems sehen. Die zweistufige Architektur wird durch eine n-stufige Architektur ersetzt. Die Weisungen durchlaufen nur eine längere Hierarchie. Eines der bekanntesten hierarchischen Systeme ist sicherlich das Domain Name System (DNS), das für die Namensauflösung im Internet zuständig ist. Als weitere lassen sich die verschiedenen Protokolle für Verzeichnisdienste, wie beispielsweise LDAP (Lightweight Directory Access Protocol) anführen.

2.4 Ringsysteme

Ringsysteme minimieren die Anzahl der benötigten Verbindungen zwischen den einzelnen Knoten. Gleichzeitig sind alle Knoten gleichberechtigt. Man kann Ring-

systeme als eine Vereinfachung eines dezentralen Systems sehen. Die bekannteste Implementierung ist ein Token Ring Netzwerk.

2.5 Vor- und Nachteile der einzelnen Topologien

Bei einem zentralen verteilten System wird die Art und Weise der Verteilung zentral gesteuert. Bei einem dezentralen System ist diese Steuerung verteilt. Aus diesem Grund ist die Implementierung eines zentralen verteilten Systems einfacher als die eines dezentralen Systems. Die schwierigere Implementierung liegt unter anderem in den folgenden Punkten begründet:

- Dynamik der Teilnehmer - Bei vielen Systemen sind nicht alle Teilnehmer immer verfügbar. Dies muss berücksichtigt werden.
- Heterogenität der Informationsquellen - Durch das Fehlen eines zentralen Servers muss der einzelne Knoten die Informationen aus einer Vielzahl von Quellen beziehen.
- Fehlen einer globalen Sicht - Es ist für einen einzelnen Knoten schwierig, Informationen über das gesamte Netz zu erhalten. Speziell bei Suchabfragen können dadurch Skalierungsprobleme auftreten.
- Hohe Datenübertragungskosten - Der Kommunikationsaufwand ist innerhalb eines dezentralen Systems höher als in einem zentralen System.
- Konsistenzprobleme bei der verteilten Datenhaltung und verteilten Berechnung.

Die größten Nachteile zentraler Systeme sind unter anderem:

- Schlechte Verfügbarkeit - Bei einem zentralen Server ist die Ausfallwahrscheinlichkeit höher als bei mehreren redundanten Systemen.
- Mangelnde Fehlertoleranz - Man kann nicht überprüfen, ob die Daten, welche der Server liefert, korrekt sind oder nicht. Bei einem dezentralen System ist im Rahmen eines Wahlverfahrens die Bestimmung sicherer.
- Keine Skalierungsmöglichkeit - Der zentrale Server stellt einen Flaschenhals innerhalb des Netzwerkes dar. Er limitiert die Leistungsfähigkeit des Netzes.

Zusammenfassend kann man sagen, dass zentrale verteilte Systeme leichter zu implementieren sind. Andererseits stellt ein zentraler Server einen kritischen

Flaschenhals innerhalb eines Systems dar. Dieses Problem wird bei einem dezentralen System umgangen, indem alle Daten und Berechnungen verteilt werden können.

Der gravierendste Nachteil eines dezentralen Systems ist allerdings die Konsistenzproblematik und die damit verbundenen hohen Datenübertragungskosten, um diese sicherzustellen.

Aus diesem Grund werden in der Praxis häufig Mischformen eingesetzt. Ein Beispiel ist das Instant Messaging System Jabber [2], bei dem die einzelnen Server dezentral organisiert sind. Die Verbindungen zu den einzelnen Clients entsprechen jedoch eher dem zentralen Ansatz.

2.6 Namens- und Lokalisierungsmechanismen

Für die einzelnen Konsistenzalgorithmen ist es notwendig, dass sämtliche Hosts und Datenatome über global eindeutige Namen addressierbar sind. Dies kann zum Beispiel über eine URL (Uniform Resource Locator) oder eine GUID (Global Unique Identifier) erfolgen. Einen Überblick über Namens- und Lokalisierungsmechanismen für dezentrale verteilte Systeme bietet [9].

3 Konsistenzbegriff

Die Abgrenzung des Begriffes Konsistenz kann in einem ersten Ansatz auf zwei verschiedene Arten erfolgen. Bei der Betrachtung von verteilten Berechnungen bezieht sich der Konsistenzbegriff auf die globale Gültigkeit der Einzellösungen zu jedem Zeitpunkt.

Hierfür müssen die Einzellösungen anhand von globalen Gültigkeitsbedingungen (Constraints) überprüfbar sein. Desweiteren muss sichergestellt werden, dass kein Deadlock bei der verteilten Berechnung auftritt. Einen guten Überblick über diese Problematik liefert [8] und [1].

Im Weiteren wird die Korrektheit von verteilten Datenbeständen (Replikaten) zu einem beliebigen Zeitpunkt betrachtet. Dieses Problem lässt sich darauf reduzieren, dass geklärt werden muss, welche Daten bei einer Leseoperation geliefert werden sollen.

3.1 Strikte Konsistenz

Die schärfste und gleichzeitig einsichtigste Definition von Konsistenz, ist die folgende:

> Eine Leseoperation gibt den Wert der letzten Schreiboperation zurück.

```
a = 1; a = 2; print a;
```

Auf einer sequentiell arbeitenden Maschine wäre jeder Programmierer überrascht, wenn die dritte Zeile 1 oder einen anderen Wert als 2 ausgeben würde. Innerhalb einer verteilten Umgebung ergeben sich allerdings zwei Probleme:

- Weder Schreib- noch Leseoperationen treten zu genau einem Zeitpunkt auf. Der Absetzungszeitpunkt lässt sich zwar genau bestimmen, jedoch haben unterschiedliche Operationen unterschiedliche Laufzeiten.
- Innerhalb eines verteilten Systems lässt sich keine absolute globale Zeit feststellen. Die Synchronisation von Uhren ist nur bis zu einer begrenzten Genauigkeit möglich.

Diese Einschränkungen haben in der Literatur zu einer Vielzahl von verschiedenen Konsistenzmodellen geführt (vgl. [8] [1]). Diese Modelle unterscheiden sich alle dadurch, wie und wann die Synchronisation der einzelnen Replikate erfolgt.

In einem ersten Ansatz lassen sich die einzelnen Konsistenzmodelle in konservative und in optimistische Konsistenzmodelle unterteilen.

3.2 Konservative Konsistenzmodelle

Konservative Konsistenzmodelle zeichnen sich dadurch aus, dass sie die Konsistenz von vornherein sicherstellen. Gleichzeitig ist diese Art der Konsistenzsicherung verhältnismässig teuer.

3.2.1 Linearisierbarkeit und Sequentielle Konsistenz

Das stärkste Konsistenzmodell, welches praktisch einsetzbar ist, ist das der sequentiellen Konsistenz. Es wurde zuerst von Lamport [5] für gemeinsamen Speicher innerhalb eines Mehrprozessorsystems entwickelt. Hierbei wird die Forderung nach einer exakten Zeitbestimmung aufgegeben. Die einzelnen Operationen (Lese- und Schreiboperationen) müssen nur in der richtigen Reihenfolge (Sequenz) erfolgen. Diese Form von Konsistenzerhaltung kann unter anderem durch die Verwendung eines zentralen Servers erreicht werden, der die Linearisierung übernimmt.

Zu diesem Verfahren ergeben sich allerdings verschiedene Probleme:

- Das Verfahren ist sehr aufwendig. Innerhalb größerer Netze kann es aus Performancegründen nicht eingesetzt werden. Das Netz wird mit steigender Anzahl der Teilnehmer immer stärker belastet.
- Die Verwendung eines zentralen Servers erzeugt eine Schwachstelle innerhalb des gesamten Systems (Ausfallgefahr, Überlastungsgefahr).

Unter bestimmten Umständen lässt sich das Modell der sequentiellen Konsistenz auch in einem dezentralen verteilten System einsetzten. Dabei erfolgt die Benachrichtigung aller Replikate durch sortierte Multicastnachrichten [8], die erst dann zurückkehren, nachdem die Aktualisierungsnachricht lokal ausgeliefert wurde. Danach können sich die einzelnen Prozesse über die Reihenfolge einigen.

3.2.2 Schwache Konsistenz

Die Kosten für die Linearisierung der einzelnen Operationen sind sehr hoch. In einem schwachen Konsistenzmodell wird deswegen versucht, exakt diese Kosten zu minimieren.

Es werden sogenannte Synchronisationsvariablen eingeführt, welche einen kritischen Bereich innerhalb der Datenbasis markieren. Nach Verlassen des kritischen Bereichs werden mit Hilfe der zugehörigen Synchronisationsvariable alle Replikate synchronisiert. Die aufwendige Linearisierung muss nur noch für die Synchronisationsvariablen durchgeführt werden. Die Konsistenz der einzelnen Replikate ist allerdings nur nach einer Synchronisation sichergestellt.

Schwache Konsistenzmodelle sind durch die folgenden drei Eigenschaften definiert [8]:

- Der Zugriff auf die Synchronisationsvariablen unterliegt sequentiellen Konsistenzforderungen.
- Es darf keine Änderung an einer Synchronisationsvariable vorgenommen werden, bis alle Replikate aktualisiert wurden.
- Kein anderer Prozess darf Operationen auf dem kritischen Bereich durchführen bis die Synchronisationsvariable wieder freigegeben wird.

3.3 Optimistisches Konsistenzmodell

Im Unterschied zu den konservativen Konsistenzmodellen wird bei den optimistischen Modellen nicht versucht, die Konsistenz von vornherein sicherzustellen. Die Konsistenz wird vielmehr erst im Nachhinein über einen Merging-Algorithmus wiederhergestellt.

Diese Vorgehensweise behebt einige Probleme der konservativen Konsistenzmodelle:

- Es ist nicht mehr nötig, dass alle Repliken-Manager ständig miteinander verbunden sind. Speziell im mobilen Bereich sind konservative Konsistenzmodelle selten implementierbar.
- Konservative Konsistenzmodelle haben Skalierungsprobleme bei größeren Netzwerken.

Der optimistische Ansatz hat allerdings den entscheidenden Nachteil, dass Konflikte bei der Synchronisation auftreten können. Wenn zwei Datenatome in beiden Replikaten verändert wurden, dann kann man nicht herausfinden, welche Änderung die richtige war. Bei der Konfliktlösung muss eine externe Instanz die Entscheidung treffen, welches der zwei Atome das richtige ist.

Die Häufigkeit, mit der solche Konflikte auftreten, hängt entscheidend von der Granularität der Datenbestände, der Häufigkeit der Synchronisationen sowie der Menge an Schreibzugriffen auf die einzelnen Datenbestände ab. Je kleiner die einzelnen Datenatome, umso unwahrscheinlicher ist das Auftreten eines Konfliktes.

4 Synchronisation

Im Rahmen der datenorientierten Konsistenz bezieht sich der Konsistenzbegriff immer auf Synchronisationsaspekte. Bei aktiver Synchronisation benachrichtigt der Repliken-Manager, welcher einen Schreibzugriff auf einem Datenatom vorgenommen hat, alle anderen Replikate. Bei passiver Synchronisation fragt ein Repliken-Manager die anderen Manager, ob sie Änderungen an den Datenbeständen vorgenommen haben.

4.1 Aktive Synchronisation

Durch aktive Synchronisation ist die Erfüllung von starken Konsistenzforderungen möglich. Es ergeben sich hierbei zwei verschiedene Aktualisierungsoptionen.

4.1.1 Schreiben-Aktualisieren

Die Änderungen eines Repliken-Managers erfolgen lokal und werden allen anderen Repliken-Managern über Multicast mitgeteilt. Bei der Verwendung von vollständig sortierten Multicasts ist es möglich, auch bei mehreren Schreibern sequentielle Konsistenz aufrechtzuerhalten [8]. Die Bedingungen für einen vollständig sortierten Multicast sollen deswegen an dieser Stelle kurz erläutert werden.

> Wenn ein korrekter Prozess die Nachricht m ausliefert, bevor er m' ausliefert, dann liefert jeder andere korrekte Prozess, der m' ausliefert, m vor m' aus.

Durch die vollständige Sortierung der Nachrichten, lässt sich eine global eindeutige Reihenfolge bestimmen. Diese Reihenfolge ist nicht zwingend auch eine kausale Reihenfolge. Wie in Abbildung 3 zu sehen ist, spielt die tatsächliche zeitliche Reihenfolge der Ereignisse keine Rolle. Es existieren prinzipiell zwei Möglichkeiten die globale Ordnung zu erreichen [1].

- Durch die Verwendung eines Sequenzers, welcher global eindeutig sortierte IDs liefert.
- Durch Aushandeln der jeweiligen IDs unter allen teilnehmenden Prozessen. Hierbei schlägt jeder Prozess eine ID vor. Die größte ID wird dann verwendet.

Es ist ersichtlich, dass diese Art der Konsistenzsicherung in jedem Fall sehr teuer ist. Bei einem auf einem Sequenzer basierenden Schema stellt dieser einen kritischen Ausfallpunkt dar. Auf der anderen Seite ist die Latenzzeit bei einem Sequenzer wesentlich geringer, als wenn alle beteiligten Prozesse die global gültigen IDs erst aushandeln müssen.

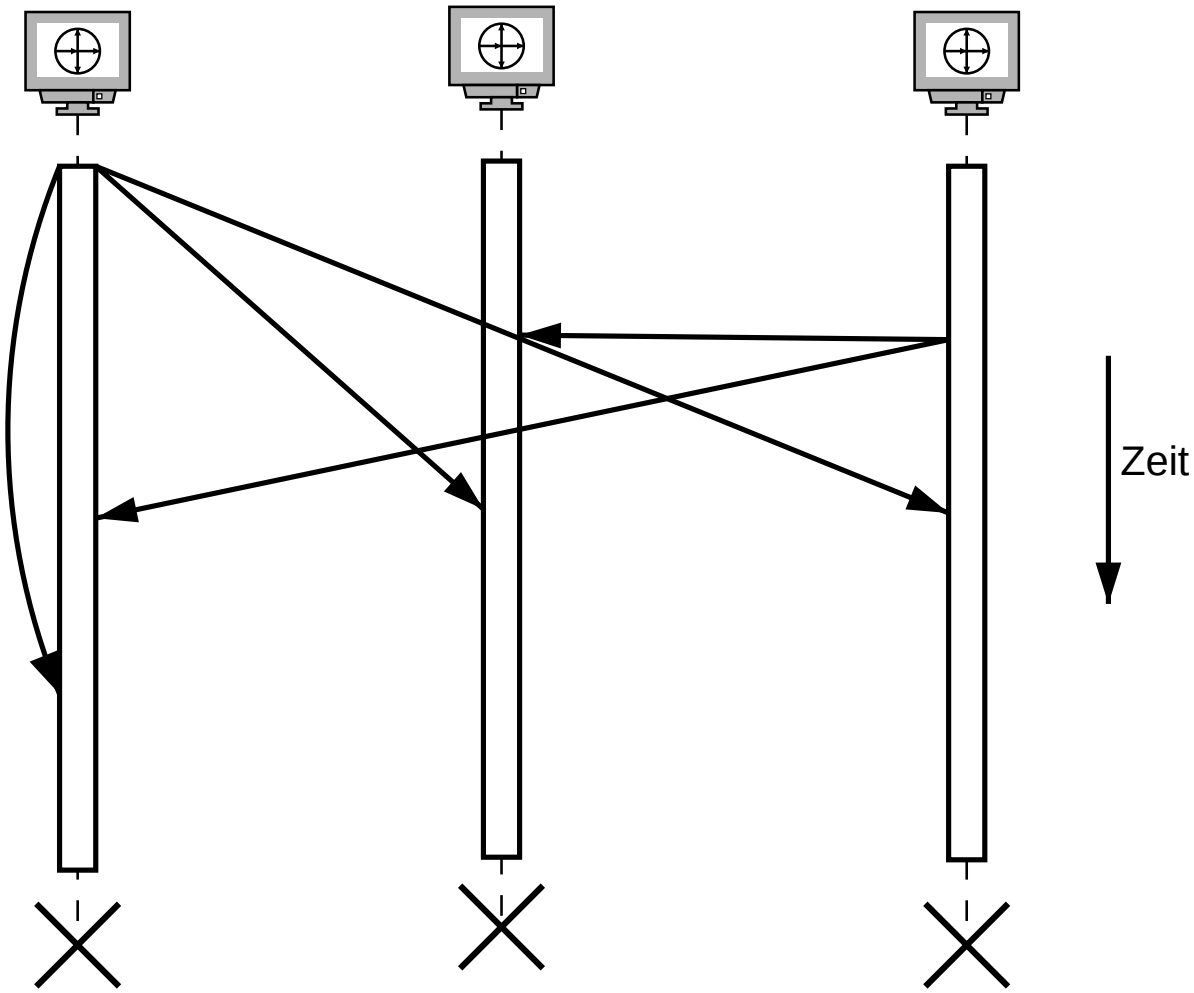

Abbildung 3: vollständig sortierter Multicast

4.1.2 Schreiben-Entwerten

Um die Kosten für die vollständige Sortierung zu reduzieren wird, bei dem Verfahren des Schreiben-Entwertens nur ein Schreibzugriff pro Zeiteinheit zugelassen. Hierbei kann ein Datenatom zu jedem Zeitpunkt entweder von einem bzw. mehreren Prozessen gelesen oder von einem Prozess gelesen und geschrieben werden. Ein Element, das gerade gelesen wird, kann beliebig oft schreibgeschützt kopiert werden. Will nun ein Prozess dieses Element schreiben, so werden alle anderen Leser über Multicast benachrichtigt. Der Prozess beginnt dabei erst dann zu schreiben, wenn er von allen anderen Prozessen eine Rückmeldung erhalten hat.

Lamport [5] hat bewiesen, dass dieses Schema sequentielle Konsistenz reali-

siert. Dieser Ansatz ist bei vielen Lesern und wenigen Schreibern sehr effizient. Wenn das Verhältnis von Lesern und Schreibern allerdings kippt, dann ist der Schreiben-Aktualisieren Ansatz vorzuziehen.

4.1.3 Token Ring

Mittels eines Token Ring Ansatzes kann man die Kosten für die vollständige Sortierung noch weiter senken [8]. Die einzelnen Repliken-Manager werden innerhalb eines logischen Rings angeordnet. Jeder einzelne Repliken-Manager kennt seinen Nachfolger. Der Repliken-Manager, der den Token hat, kann Aktualisierungen an dem Datenbestand vornehmen. Alle anderen Repliken-Manager müssen über die Datenbestandsänderung benachrichtigt werden.

Die Kostenreduktion wird allerdings mit einigen zusätzlichen Problemen erkauft. Wenn der Token verloren geht, muss er wieder erzeugt werden. Es ist allerdings sehr schwierig zu erkennen, wann der Token verloren gegangen ist, da nicht festgelegt ist, wie lange der Token bei einem Repliken-Manager verweilen kann.

4.1.4 Zusammenfassung

Alle Algorithmen haben den Vorteil, dass die Konsistenz von vornherein sichergestellt wird. Es kann nicht wie bei einem optimistischen Verfahren zu Konflikten kommen.

Für die aktive Synchronisation müssen allerdings einige Voraussetzungen erfüllt sein, die die praktische Verwendbarkeit einschränken.

- Die Verfahren sind im Bezug auf das Datenvolumen und die Latenzzeit sehr teuer. Aus diesem Grund können starke Konsistenzanforderungen nur in einem kleinen Netz umgesetzt werden.
- Alle beiden Verfahren funktionieren nur in einem sicheren Netz. Es wird immer davon ausgegangen, dass die einzelnen Prozesse wahrheitsgemäße Werte liefern.
- Die Verfahren sind zur Zeit nicht fehlertolerant. Man nimmt an, dass alle Teilprozesse korrekt funktionieren und richtige Werte liefern.

All diese Algorithmen lassen sich selbstverständlich auch mit einem schwachen Konsistenzmodell umsetzen. Ein schwächeres Konsistenzmodell skaliert besser als ein starkes Konsistenzmodell. Allerdings ist die Konsistenz dann nur unter bestimmten Bedingungen sichergestellt. Das Konsensproblem, das in den letzten beiden Punkten angesprochen wurde, kann durch Versenden signierter Nachrichten und verschiedenen Wahlalgorithmen gelöst werden.

4.2 Passive Synchronisation

Im Unterschied zur aktiven Synchronisation, erfolgt bei der passiven Synchronisation keine Benachrichtigung, wenn ein Repliken-Manager (Server) Änderungen an seinen Datenbeständen vornimmt. Vielmehr wird die Synchronisation von den Clients initiiert. Einerseits ist mittels dieser Algorithmen nur ein schwaches Konsistenzmodell umsetzbar. Auf der anderen Seite sind diese Algorithmen robuster gegenüber zeitweisen Netzwerkausfällen.

4.2.1 Caching-Algorithmen

Bei Caching Algorithmen existiert ein Server und mehrere Clients, die die Datenbestände des Servers für beschleunigte Lesezugriffe zwischenspeichern. Sie ähneln damit dem Schreiben-Entwerten Ansatz im vorigen Kapitel. Es kann hier allerdings nur ein zentraler Server schreiben. Die Aktualisierung der Caches kann wiederum auf zwei verschiedene Arten erfolgen:

- Die Clients fordern von Zeit zu Zeit neue Datenbestände vom Server ab (Poll Prinzip). Dies kann zum einen nach einer bestimmten Zeit (Timeout) erfolgen. Ein Beispiel hierfür ist das Domain Name System. Zum anderen ist es möglich, bei jedem Lesezugriff zu überprüfen, ob der Datenbestand auf dem Server aktualisiert wurde. Ist dies der Fall, dann wird der lokale Cache aktualisiert. Viele Proxy-Systeme implementieren dieses System.

- In einem anderen Ansatz werden, wie beim Schreiben-Entwerten Ansatz, alle Clients von der Datenänderung benachrichtigt (Push Prinzip). Diese Benachrichtigung kann synchron oder asynchron erfolgen. Aufgrund der Tatsache, dass nicht verfügbare Clients sehr schlecht behandelt werden können, existieren für dezentrale verteilte Systeme wenige Implementierungen. Das Usenet implementiert diesen Ansatz. Die einzelnen Newsserver schieben sich neue Nachrichten gegenseitig zu.

Reine Caching-Algorithmen werden primär in zentralen verteilten Systemen oder hierarchischen Systemen eingesetzt. Es gibt jedoch zwei Möglichkeiten um diese Algorithmen innerhalb eines dezentralen Systems einzusetzen:

- Durch den Einsatz von Wahlalgorithmen kann der zentrale Server gewählt werden. Ein Beispiel hierfür ist die Verwaltung von sogenannten Browselisten innerhalb eines SMB Netzwerkes.

- Bei dem Vorhandensein von vielen kleinen lokalen Datenbasen, wie sie in einem Multiagentensystem [4] auftreten, können Caching-Algorithmen verwendet werden.

Caching-Algorithmen sind aus naheliegenden Gründen schneller als der Schreiben-Entwerten Ansatz. Andererseits kann nur ein einzelner Server schreiben, welcher von vornherein festgelegt werden muß.

4.2.2 Merging-Algorithmen

Einen komplett anderen Ansatz verfolgen die Merging-Algorithmen. Während alle anderen Algorithmen ein konservatives Verfahren implementieren, verwenden die Merging-Algorithmen ein optimistisches Verfahren.

Alle Repliken-Manager können Änderungen an ihren lokalen Datenbeständen vornehmen. Bei Bedarf werden alle Datenbestände miteinander synchronisiert. Diese Synchronisation erfolgt durch sogenanntes Mergen (Vermischen). Hierfür schreiben alle Replikenmanager die Änderungen, die an den Datenbeständen vorgenommen wurden, mit.

Bei dem Vermischen von Datenbanken können allerdings im Unterschied zu den vorigen Verfahren Konflikte auftreten. Wenn ein Datenatom auf beiden Seiten verändert wurde, dann kann man nicht ohne weiteres feststellen, welches der richtige Wert ist. Dies muss anderweitig entschieden werden. Die Wahrscheinlichkeit für einen Konflikt hängt direkt von der Granularität der Datenbanken ab. Je kleiner die einzelnen Datenatome sind, desto seltener treten Konflikte auf.

4.3 Konsens- und Wahlalgorithmen

Wie bereits weiter oben angedeutet, haben alle bisher vorgestellten Algorithmen das Problem, dass sie bei fehlerhaften oder böswilligen Repliken-Managern auch fehlerhaft arbeiten. Diese fehlerhaften Repliken-Managern sind in den eingangs erwähnten Dateitausch Netzen eher die Regel als die Ausnahme. Die einzelnen Hosts sind nicht ständig verfügbar. Desweiteren sind Leistung, Speichermenge und Bandbreite relativ unbekannt. Außerdem ist die Datensicherheit und die Vertrauenswürdigkeit der einzelnen Hosts nicht gegeben.

Um das Problem der Vertrauenswürdigkeit zu lösen, muss bestimmt werden, ob eine Änderung, die ein einzelner Repliken-Manager vorgenommen hat, von den anderen Repliken-Managern übernommen wird. Ebenso muss bei Ausfall eines Repliken-Managers, der mit einer speziellen Aufgabe betraut wurde, ein Ersatz gewählt werden.

Bei einem Konsens- und Wahlalgorithmus (Quorum Consensus [3]) gibt jeder Repliken-Manager seine Stimme zu einem bestimmten Problem ab. Um den Sicherheitsaspekt zu erhöhen, muss jeder einzelne Repliken-Manager seine Wahl signieren.

Es existieren verschieden Spielarten des Quorum Consensus. Variabel sind

hierbei die benötigte Mehrheit, sowie die Gewichtung der Stimmen. Einen Überblick über das Thema bieten [3, 1, 8].

5 Fazit

Alle Verfahren und Topologien haben ihre Vor- und Nachteile. Welches Verfahren letztendlich eingesetzt werden sollte, hängt vom Anwendungsgebiet ab.

Im mobilen Bereich, wenn einzelne Knoten von anderen Netzen getrennt werden, kommt man um die Verwendung eines optimistischen Verfahrens nicht herum. In vielen Anwendungsfällen treten die Konflikte aber auch verhältnismäßig selten auf. So verwendet zum Beispiel das bekannteste Versionsverwaltungsprogramm CVS (Concurrent Versions System) ein optimistisches Verfahren, um die Konsistenz der Dateien zu sichern. Das Auftreten von Konflikten wird durch organisatorische Maßnahmen (Absprachen zwischen den Entwicklern) minimiert.

Im Folgenden sind noch einmal alle Algorithmen und ihre Einsatzgebiete aufgeführt:

- Schreiben-Aktualisieren - vorteilhaft bei vielen Schreiber mit gleichzeitig starken Konsistenzanforderungen (z.B. verteilter Arbeitsspeicher)
- Schreiben-Entwerten - wenige Schreiber, viele Leser mit starken Konsistenzanforderungen (z.B. Multiagentensystem)
- Token Ring - Verwendung in stabilen Netzen günstig
- Caching-Algorithmen - vorteilhaft bei einer geringen Aktualisierungsrate
- Merging-Algorithmen - wenn Aktualisierungen an unterschiedlichen Stellen vorgenommen werden und die Repliken-Manager nicht immer miteinander kommunizieren können (z.B. Versionsverwaltungen)

Abbildungsverzeichnis

Literatur

[1] COULOURIS, G. ; DOLLIMORE, J. ; KINDBERG, T. : *Verteilte Systeme - Konzepte und Design*. Dritte Auflage. Addison Wesley : Precentice Hall, 2002

[2] FOUNDATION, J. S. *Jabber Webseite*. http://www.jabber.org

[3] HELAL, A. A. ; HEDDAYA, A. A. ; BHAGAVA, B. B.: *Replication Techniques in Distributed Systems*. Erste Auflage. Kluwer Academic Publishers, 1996

[4] KIRN, P. : *Multiagentensysteme*. Erste Auflage. Addison-Wesley, München, 2001

[5] LAMPORT, L. : How to Make a Multiprocessor that Correctly Excutes Multiprocessor Programs. In: *IEEE Trans. Comp.* C-29 (1979), Sept., Nr. 9, S. 690–691

[6] MINAR, N. . *Distributed Systems Topologies: Part 1*. `http://www.openp2p.com/pub/a/p2p/2001/12/14/topologies_one.html`

[7] PROJECT, S. . *Seti@Home Webseite*. http://setiathome.ssl.berkeley.edu/

[8] TANENBAUM, A. S. ; VAN STEEN, M. : *Distributed Systems - Principles and Paradigms*. Erste Auflage. New Jersey : Precentice Hall, 2002

[9] UNBEHAUN, R. . *Namens- und Lokalisierungsmechanismen in dezentralen verteilten Systemen*. Hauptseminar, Technische Universität Ilmenau. 2003