Seminararbeit

Replikationsarchitekturen

Informationsverwaltung in Netzen
Sommersemester 2003

Konrad Kretschmer
kkretsch@inf.fu-berlin.de
Betreuer: Manuel Scholz

Berlin, 3. Juli 2003

Inhaltsverzeichnis

1. Einleitung....4
2. Hintergrund....5
   2.1 Optimistische Replikation....5
   2.2 Datensynchronisation....5
   2.3 Selektive Replikation....6
   2.4 FICUS....6
3. RUMOR....7
   3.1 Gossip....8
   3.2 Versionsvektoren....8
   3.3 Ableichstopographie: Adaptiver Ring....8
   3.4 Garbage Collection....9
4. ROAM....11
   4.1 Das WARD-Modell....11
   4.2 Das erweiterte WARD-Modell....13
5. Literaturverzeichnis....15

1. Einleitung

In dieser Arbeit sollen Verfahren zur Handhabung von Replikationen von Daten in P2P-Netzwerken behandelt werden. Besonders bei Teilnahme von *mobilen* Teilnehmern tritt das Problem der Ressourcenoptimierung auf, da bei ihnen nicht von einer zuverläßigen, billigen und dauerhaften Verbindung zum Netzwerk ausgegangen werden kann.

Als mobile Teilnehmer gelten solche, die mit mobilen Maschinen arbeiten (Notebook, PDA, o.Ä.) und sich dadurch auszeichnen, dass sie ihren Standort des Öfteren wechseln und nicht über eine kostengünstige Verbindung zum Netzwerk verfügen. Es müssen also Methoden verfügbar sein, die den Verkehr zwischen dem mobilen Teilnehmer und dem Netzwerk minimieren und flexibel genug sind, mit geringen Kosten auf veränderte Teilnahmebedingungen am Netzwerk zu reagieren. Dies bedeutet eine verbesserte Auswahl und eine erhöhte Auslagerung von Daten auf der Maschine selbst.

Zunächst werden in Kapitel 2 einige Grundlegende Begriffe und Mechanismen erklärt, die dann in den vorgestellten Systemen Anwendung finden werden.

In Kapitel 3 wird das System RUMOR vorgestellt, das die Replikate in einem adaptiven Ring verwaltet. In Kapitel 4 wird das auf RUMOR aufbauende System ROAM beschrieben, dass wesentliche Verbesserungen in der Leistungsfähigkeit bei hoher Teilnehmeranzahl bringt.

2. Hintergrund

In diesem Kapitel sollen einige, grundlegende Begriffe erläutert werden, die Mechanismen beschreiben, auf die in den vorgestellten Replikationssystemen zurückgegriffen wird.

2.1 Optimistische Replikation

Bei verteilten Systemen jeder Form tritt das Problem der Behandlung von unterschiedlichen Versionen von Informationen innerhalb des Netzwerkes auf. Wenn sich z.B. mehrere Teilnehmer die selbe Version einer Datei aus dem Netzwerk herunterladen, sie unabhängig voneinander bearbeiten und anschließend wieder in das Netzwerk hochladen, so entsteht ein Versionskonflikt.

In älteren, sogenannten *konservativen* Algorithmen werden Probleme dieser Art dadurch gelöst, dass entweder die Daten die einmal von einem Teilnehmer heruntergeladen wurden für weitere Teilnehmer solange gesperrt werden, bis der Teilnehmer die veränderten Daten wieder in das Netzwerk hochgeladen hat, oder sie freigibt, oder nur die erste hochgeladene Version eines Teilnehmers als gültige Aktualisierung anerkannt und in das Netzwerk aufgenommen wird und weitere, die eventuelle Versionskonflike enthalten, da sie nicht auf der aktuellen Version basieren, zurückgewiesen werden. Dadurch erspart man sich weitere Konfliktbehandlungen muss jedoch große Einbußen bezüglich der Verfügbarkeit der Daten machen.

Optimistische Replikation umgeht dieses Problem dadurch, dass sie es weitgehend ignoriert. Es wird davon ausgegangen, dass solche Konflikte relativ selten auftreten und meistens automatisiert und ohne Schwierigkeiten durch Verschmelzung der Versionen gelöst werden können (s. *3. Datensynchronisation*).

Da es bei P2P-Systemen sehr viel schwerer als bei klassischen Server-Client-Modellen ist, vorherzusagen, welche Teilnehmer Daten untereinander austauschen, sind bei ihnen optmistische Replikationsalgorithmen aufgrund der erhöhten Verfügbarkeit der Daten wesentlich lohnender. Für mobile Teilnehmer bedeutet optimistische Replikation eine Verringerung der notwendigen Verbindungszeit zum Netzwerk, da eventuell auftretende Versionskonflikte auf der mobilen Maschine selbst gelöst werden können und nicht über die Sperrung von Daten verhindert werden.

2.2 Datensynchronisation

In jedem P2P-Netzwerk muss, im Gegensatz zu Client-Server-Architekturen, die Information über Daten, die von einem Teilnehmer verändert wurden, an alle anderen Teilnehmer weitergeleitet werden. Dies kann entweder durch *Sofortige Benachrichtigung* oder durch *Periodischen Abgleich* geschehen.

Bei Sofortiger Benachrichtigung werden alle anderen Teilnehmer so schnell es geht von der Veränderung des Datenbestandes informiert. Die Nachricht der Veränderung wird sofort an einen oder mehrere andere Teilnehmer geschickt, die sie weitergeben und so das gesamte Netzwerk immer über den aktuellen Stand der Daten informiert ist. Dieser Vorgang kann unverhältnismäßig viel Ressourcen verbrauchen, falls keine wichtigen oder relevanten Daten modifiziert wurden und ist für mobile Teilnehmer ungünstig, da sie sich öfter und eventuell zu ungünstigen Konditionen mit dem Netzwerk verbinden müssen.

Beim Periodischen Abgleich hingegen wird die Information der Datenveränderung nicht sofort, sondern zu einem späteren, günstigeren Zeitpunkt in

das Netzwerk geschickt. Bei Architekturen mit Optimistischer Replikation ist diese Variante notwendig, da man bei ihnen auch die Daten erneuern kann, wenn man nicht mit dem Netzwerk verbunden ist. Erst wenn der mobile Teilnehmer sich das nächste Mal, zu einem von ihm gewählten, günstigen Zeitpunkt, zum Beispiel, wenn er sich in relativer, geographischer Nähe zu den anderen Teilnehmern des Netzwerkes befindet, mit dem Netzwerk verbindet, wird dieses vom aktuellen Zustand der Daten informiert. Da bei Optimistischer Replikation davon ausgegangen wird, dass es zu keinem Versionskonflikt kommt, und für auftretende Versionskonflikte sowieso Lösungsalgorithmen bestehen, besteht der einzige Nachteil dieser Variante darin, dass die Informationen im Netzwerk, im Vergleich zur sofortigen Benachrichtigung, nicht so schnell auf dem aktuellen Stand sind.

2.3 Selektive Replikation

Jedes P2P-System verfügt über eine gewisse Granularität, die angibt, wie groß die kleinste, logische oder zusammenhängende Einheit sein soll. Dies können zum Beispiel ganze Volumes, Verzeichnisse oder nur einzelne Dateien sein. Für den Datenabgleich muss es aber nicht immer notwendig sein, z.B. ein ganzes Verzeichnis zu synchronisieren, wenn ein Teilnehmer nur eine einzelne Datei verändert hat. Um solche Arten von unnötigem Datentransfer zu vermeiden gibt es die *Selektive Replikation*. Bei der Selektiven Replikation kann jeder Teilnehmer ohne Einschränkungen nur die Daten aus dem Netzwerk herunterladen, die er auch wirklich benötigt, auch in feinerer als der allgemeinen Granularität. Wieder ist der Vorteil gerade für mobile Teilnehmer groß, bei denen keine kostengünstige, zuverlässige Verbindung zum Netzwerk vorausgesetzt werden kann.

2.4 FICUS

Das Dateiverwaltungssystem FICUS wurde 1990 entwickelt.[2] Es ist ein reines P2P-System, das auf optimistischer Replikation basiert. Die Grundgranularität von FICUS ist die Volume-Ebene, später wurde jedoch Selektive Replikation hinzugefügt.

FICUS arbeitet mit sofortiger Benachrichtigung, verfügt aber auch zur Sicherstellung der Konsistenz über periodisch aufgerufenen Abgleich.

FICUS ist Teil des SunOS 4.1.1 und dort ein Bestandteil des Kernels, was Nachteile bezüglich der Distribution und der Portation auf andere Betriebsysteme nach sich zieht. Als reines P2P-Netzwerk sinkt die Leistungsfähigkeit von FICUS bei größer werdender Teilnehmerzahl schnell ab.

Alle im Folgenden dargestellten Systeme basieren auf FICUS und viele der dort benutzten Algortihmen stammen oder sind inspiriert von FICUS.

3 RUMOR

Das 1998 entwickelte System RUMOR basiert auf FICUS, weist jedoch einige nützliche Änderungen und Erweiterungen auf.[3] So ist auch RUMOR ein optimistisches Replikationskonzept, das mit der Granularität eines Volumes arbeitet. Da es jedoch vor allem für mobile Teilnehmer handhabbar sein soll, wird bei RUMOR die Synchronisation ausschließlich durch Periodischen Abgleich durchgeführt; auf Sofortige Benachrichtigung wurde aufgrund der häufig auftretenden Abwesenheit von Teilnehmern vom Netzwerk verzichtet. Auch wurde RUMOR mit der Möglichkeit zur Selektiven Replikation ausgestattet um auch hier den Verkehr von unnötigen Daten zu minimieren.

RUMOR arbeitet ausschließlich auf der Anwendungsebene. Deshalb ist das System leicht und auch ohne Administrationsrechte zu installieren. Es greift im Gegensatz zu FICUS nicht direkt auf Systemaufrufe des Kernels zu und ist damit leicht auf andere Betriebssysteme portierbar. RUMOR wurde in C++ geschrieben. Da also die für das System benötigten Informationen nicht im Dateisystem selbst gespeichert werden können, geschieht dies mit Hilfe einer Datenbank.

Der Abgleich zweier Replikationen wird in drei Phasen durchgeführt: Der Scanphase, der Kontaktierphase und der Abgleichsphase (s. *Abb.1*). In der Scanphase werden die Attribute des Replikats aus der Datenbank gelesen und mit der tatsächlichen Replikation, durch rekursives Durchlaufen des Volumes, verglichen und aktualisiert (s. *Versionsvektoren*). In der Kontaktierphase wird das in der Abgleichstopographie nächste Replikat kontaktiert (s. *Adaptiver Ring*) und dessen Attribute auf die lokale Maschine übertragen. In der Abgleichsphase wird der eigentliche Datentransfer durchgeführt. Was in der Abgleichsphase getan wird hängt vom Ergebnis des Vergleichs in der Kontaktierphase ab. Wenn die lokale Version neuer ist als die kontaktierte, dann geschieht nichts. Der Abgleich wird immer nur in einer Richtung, also von der iniziierenden Seite aus, durchgeführt. In diesem Fall wird also nicht das kontaktierte Replikat aktualisiert, da davon ausgegangen werden kann, dass diese an die aktueller Version gelangt, wenn sie einen Abgleich startet (s. *Gossip*). Wenn die kontaktierte Version neuer ist, dann wird der Datentransfer gestartet, und das kontaktierende Replikat auf den aktuellen Stand gebracht. Die Attribute werden jetzt noch nicht aktualisiert, da dies erst in der Scanphase des nächsten Abgleichs stattfindet, und bis dahin nicht relevant ist, da lokal noch Änderungen vorgenommen werden können. Wenn beide am Abgleich teilnehmende Replikate über identische Versionen verfügen, so wird nichts unternommen. Die letzte Möglichkeit ist die, dass ein Versionskonflikt auftritt. In diesem Fall muss ein automatisierter Konfliktlösungsalgorithmus gestartet werden, oder manuell von einem Benutzer eingegriffen werden.

Durch diese Dreiphasenarchitektur ist auch die Selektive Replikation problemlos möglich.

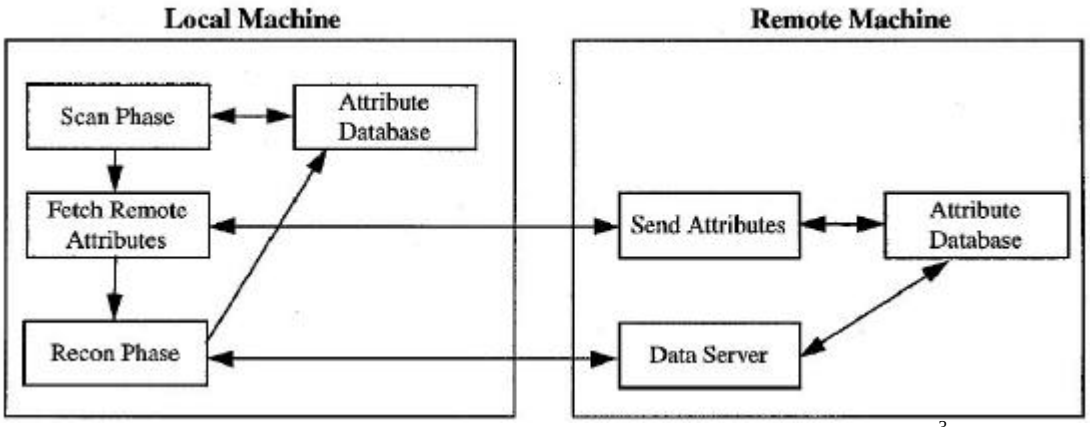

Abb. 1: Die Dreiphasenarchitektur des Periodischen Abgleichs bei RUMOR [3].

3.1 Gossip

Während des periodischen Abgleichs verbindet sich der Teilnehmer mit nur einem weiteren Teilnehmer, und überspielt alle Informationen über die er verfügt zu diesem Teilnehmer. Dies sind nicht nur Informationen über den verschickenden Teilnehmer und dessen Daten, sondern auch so genannten *Gossip*. Diese „Gerüchte“ sind Informationen über andere Teilnehmer des Netzwerks. So breiten sich die Informationen über Aktualisierungen immer weiter durch das Netzwerk aus, auch wenn der die Informationen betreffende Teilnehmer nicht mit dem Netzwerk verbunden ist.

3.2 Versionsvektoren

Um dennoch auftretende Versionskonflikte zu erkennen verwendet RUMOR Versionsvektoren. Ein Versionsvektor besteht aus einer Reihe von Zählern, einen für jeden Teilnehmer des Netzwerkes. Die Zähler geben die Anzahl der bekannten Aktualisierungen des entsprechenden Teilnehmers an. Jeder Teilnehmer hat für jede Datei über die er verfügt einen Versionsvektor. Beim Datenabgleich müssen so nur die entsprechenden Versionsvektoren miteinander verglichen werden, um festzustellen, welche Version die aktuelle ist und damit ob diese weiter propagiert werden soll. Mit Versionsvektoren kann man Versionskonflikte nicht verhindern, sie jedoch zuverläßig entdecken. Wird ein Versionskonflikt gefunden, so müssen entweder automatisierte Korrekturverfahren eingeleitet werden, oder die Benutzer müssen „per Hand“ den Konflikt bereinigen.

3.3 Abgleichstopgraphie: Adaptiver Ring

Beim Datenabgleich muss sich das abgleichende Replikat bei RUMOR mit nur einem einzigen anderen Replikat verbinden, damit der aktuelle Stand der Daten in das Netzwerk gelangt. Es muss also geklärt werden, wie dieses Replikat ermittelt wird. Hierzu ordnet RUMOR alle Teilnehmer in einem adaptiven Ring an.

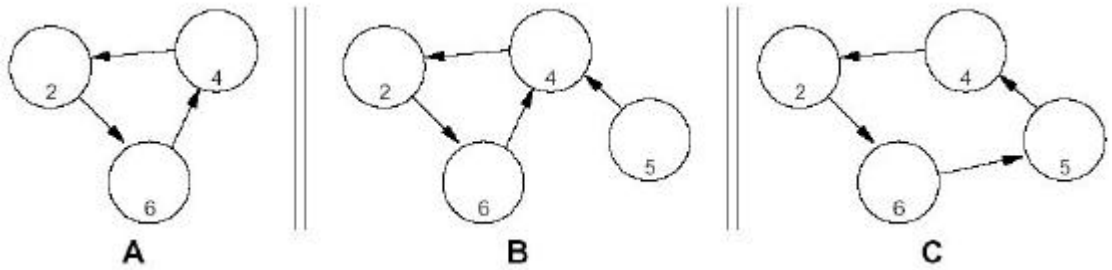

Abb. 2: Aufnahme eines neuen Teilnehmers in den adaptiven Ringe. [1]

In einem adaptiven Ring sind alle Teilnehmer fest, aber willkürlich geordnet. Jeder Teilnehmer hat einen festen Nachfolger und einen eindeutigen Vorgänger, so dass die Gesamtheit der Teilnehmer einen Ring ergibt. Die Position des einzelnen Teilnehmers innerhalb dieses Rings muss durch einen eindeutigen, metrischen Schlüssel festgelegt werden. Der Ring ist adaptiv, da seine Größe, also die Anzahl der Teilnehmer innerhalb des Rings, variabel ist (s. *Abb. 2*). Dies lässt temporäre Abwesenheit von Teilnehmern zu, was wieder einmal von besonderem Vorteil für mobile Teilnehmer ist.
Nehmen wir an, wir haben einen Ring mit drei Teilnehmern. Jedem Teilnehmer wurde eine eindeutige Zahl (Schlüssel) zugewiesen und die Information, welcher Teilnehmer zu welcher Zahl gehört in der dem System zugrunde liegenden Datenbank gespeichert, in unserem Beispiel 2, 4 und 6 (s. *Abb. 2, Bild A*). Meldet sich ein neuer Teilnehmer an, so erhält er einen Schlüssel, kontaktiert einen anderen Teilnehmer und gleicht mit diesem seine Datenbank ab (s. *Abb. 2, Bild B*). Aufgrund der Informationen über die Schlüssel kontaktiert der neue Teilnehmer denjenigen Teilnehmer im Ring, der den nächst kleineren Schlüssel hat, und gleicht mit ihm seine Daten ab. Durch den Gossip wird die Information über den neuen Teilnehmer weitergeleitet, so dass auch der Vorgänger des neuen Teilnehmers feststeht (s. *Abb. 2, Bild C*).
Sollte ein Teilnehmer nicht mit dem Netzwerk verbunden sein, so fällt dies auf, sobald sein Vorgänger sich mit ihm abgleichen will. In diesem Fall aktualisiert der Vorgänger seine Datenbank und versucht, den Teilnehmer mit dem nächst größeren Schlüssel zu kontaktieren.
Beim Datenabgleich muss sich jeder Teilnehmer also nur mit seinem Nachfolger im adaptiven Ring verbinden und diesem die Informationen über die getroffenen Aktualisierungen mitteilen. Durch den Gossip werden so nach und nach alle Teilnehmer des Netzwerkes über die Veränderung von Daten informiert. So ist zwar die Komplexität der Benachrichtigung nur linear, dafür sind bei einem Abgleich nicht alle Teilnehmer sofort informiert. Da dieser Effekt aber bei Periodischem Abgleich sowieso auftritt wird er zugunsten der Leistungsfähigkeit in Kauf genommen. Im Gegensatz zu FICUS skaliert das Programm auch bei größerer Anzahl von Teilnehmern gut. Bei mehr als ungefähr 20 Teilnehmern wird jedoch auch bei RUMOR die Leistungsfähigkeit inakzeptabel.

3.4 Garbage Collection

Im Gegensatz zu zentralisierten System tritt bei verteilten Systemen bei der Freigabe von nicht mehr benötigten Systemressourcen (in erster Linie Speicherplatz) ein Problem auf: Da mehrere Teilnehmer über die Informationen oder die Daten selbst verfügen und diese auch dynamisch umbenennen können, ist es schwieriger

festzustellen, wann eine Datei nicht mehr existiert. Wenn alle Namen und Verweise eines Objekts gelöscht wurden, so können normalerweise auch die Systemressourcen dieses Objekts freigegeben werden, in einem verteilten System, bei dem die Objekte umbenannt werden können, ist dies nicht der Fall.

Um dieses Problem, die so genannte *Garbage Collection*, zu lösen, verwendet RUMOR einen auf dem Gossip basierenden Algortihmus, der die Information mit der Ressourcenfreigabeanforderung an alle anderen Teilnehmer weiterleitet. Diese leiten dann die Information weiter, ob sie noch Kopien oder Verweise auf das endgültig zu löschende Objekt haben. Erst wenn diese Information alle Teilnehmer erreicht hat, und bestätigt wurde, werden die Ressourcen des Objekts freigegeben.

4 ROAM

Die Replikationsarchitektur ROAM[1] wurde 1998 veröffentlicht und basiert auf dem zuvor dargestellten RUMOR und damit auch auf FICUS. Bei ROAM wird eine große Zahl an Ideen von RUMOR übernommen, die Änderungen die durchgeführt wurden zielen aber hauptsächlich auf die gesteigerte Leistungsfähigkeit bei höherer Teilnehmerzahl und der Eignung für mobile Teilnehmer ab.

4.1 Das WARD-Modell

Im Gegensatz zu RUMOR ist ROAM kein reines P2P-System. Es stellt eine hybride Form zwischen P2P-Systemen und klassischen Client-Server-Architekturen dar. Es wurde das so genannte *WARD-Modell* entwickelt, um dies zu realisieren.

Im WARD-Modell werden alle Replikate eines Volumes in WARDs gruppiert (WARD steht für Wide-Area-Replication-Domains). Die Gruppierung folgt nach mehreren Kriterien, wie z.B. der geographischen Nähe der Teilnehmer (es wird generell angenommen, dass zwei Maschinen, die geographisch nah beieinander liegen geringere Verbindungskosten und eine höhere Verbindungsqualität haben als zwei Maschinen, die geographisch weit voneinander entfernt sind), der Geschwindigkeit und Qualität der Verbindung zwischen zwei Teilnehmern und den Kosten der Verbindung zwischen zwei Teilnehmern (s. *Abb.3*). Das System zielt darauf ab, dass Verbindungen innerhalb eines WARDS wesentlich häufiger auftreten als Verbindungen von zwei Teilnehmern in unterschiedlichen WARDs.

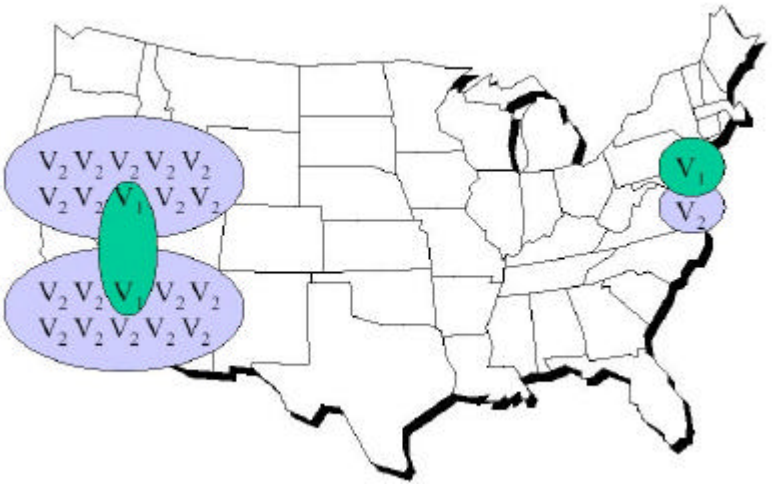

Abb. 3: Aufteilung der Replikate in WARDs aufgrund geographischer Distanz.[1]

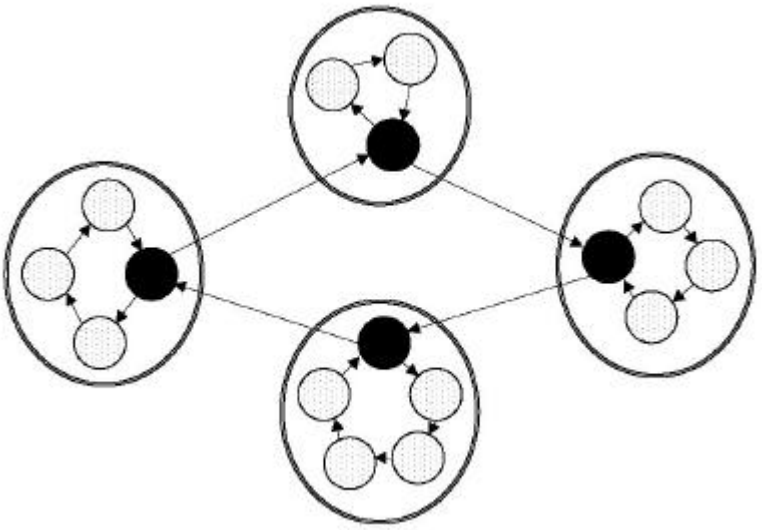

Abb. 4: Adaptive Ringtopologie innerhalb und zwischen den WARDs. [1]

In jedem WARD wird ein Teilnehmer zufällig aber fest als der so genannte *WARD-Meister* deklariert. Dem WARD-Meister fällt die Aufgabe zu, die Kommunikation zu anderen WARDs durchzuführen; alle anderen Replikate des WARDs kommunizieren nur untereinander. Auf dieses Art und Weise wird die Häufigkeit von ungünstigen Verbindungen minimiert. Es entstehen also zwei Schichten im WARD-Modell. Die untere Schicht bildet die Kommunikation innerhalb eines WARDS, die obere Schicht die Kommunikation zwischen zwei WARDs. Sowohl innerhalb eines WARDs als auch die WARD-Meister sind jeweils in einem adaptiven Ring angeordnet, also in sich ein P2P-System (s. *Abb.4*).

Sollte der WARD-Meister nicht mit dem Netzwerk verbunden sein, so kann sehr einfach ein anderer Teilnehmer (z.B. der Nachfolger des vorherigen WARD-Meisters) zum WARD-Meister werden, da der WARD-Meister innerhalb seines WARDs keine besonderen Privilegien hatte.

Für mobile Teilnehmer gibt es innerhalb des WARD-Modells zwei Formen der Bewegung. Die erste Form ist die Bewegung innerhalb des selben WARDs. Sie findet statt, wenn der Benutzer sich nur auf geringer Distanz mit seiner Maschine bewegt, wie etwa innerhalb eines Gebäudes oder einer Stadt. In diesem Fall hat die Bewegung keinerlei Auswirkungen auf den WARD, in dem sich der Teilnehmer befindet.

Bei Bewegungen eines Teilnehmers von größerer Distanz kann es dazu kommen, dass die Teilnahme an einem anderen WARD aufgrund der durch die Bewegung veränderten Verbindungskonditionen, günstiger wird als die Teilnahme an dem WARD, in dem sich der Teilnehmer zur Zeit befindet. Wenn dies der Fall ist, dann wird einerseits der mobile Teilnehmer auf normale Art und Weise zu dem neuen WARD hinzugefügt. Für die Abmeldung vom vorherigen WARD muss nur ein einziges Mitglied des alten WARDs kontaktiert, informiert und eventuell aktualisiert werden. Die anderen WARD-Teilnehmer werden nach und nach über den Gossip über das Ausscheiden des mobilen Teilnehmers informiert. Sollte kein Teilnehmer des alten WARDs verfügbar sein, so kann die Abmeldung auch zu einem späteren Zeitpunkt durchgeführt werden. Dies hat lediglich den Nachteil, das die Informationen die in den alten WARD geleitet werden müssen, bis zur vollzogenen Abmeldung auf der Maschine des mobilen Teilnehmers verbleiben müssen.

4.1 Das erweiterte WARD-Modell

Das *erweiterte* WARD-Modell erweitert das einfache WARD-Modell durch das hinzufügen von Mechanismen für Selektive Replikation, sowie einer weiteren Form der Bewegung zwischen WARDs, die es erlaubt, die Teilnahme an zwei WARDS nur für begrenzte Zeit zu besitzen, ohne sich vollständig vom vorherigen WARD abzumelden.

Es ist eigentlich nicht notwendig, dass Teilnehmer eines WARDs über alle Dateien des Volumes verfügen, auch wenn sie es nicht unbedingt benötigen. Es müssen lediglich die Metadaten des gesamten Volumes auf der Maschine jedes Teilnehmers vorhanden sein, um den Abgleich zu sichern. Auf diese Weise könnte man erhebliche Mengen an Ressourcen der einzelnen Teilnehmer einsparen, was besonders für mobile Teilnehmer von Gunsten ist, da mobile Maschinen meist über weniger Ressourcen verfügen. Hierzu muss das WARD-Modell jedoch über die Möglichkeit zur Selektiven Replikation verfügen.

Um das System mit Unterstützung für Selektiven Replikation auszurüsten, wird eine dritte Schicht in das WARD-Modell eingefügt. Die dritte Schicht befindet sich unterhalb der Kommunikationsschicht innerhalb eines WARDs. Da nicht alle Teilnehmer eines WARDs über alle Daten des im WARD verwalteten Volumes verfügen, ist der Abgleich zwischen manchen Teilnehmern bezüglich dieser disjunkten Daten unnötig. Deshalb werden alle Objekte von feinerer Granularität (z.B. auf Dateiebene) in einem eigenen adaptiven Ring organisiert. Auf diese Weise kann ein selektiver Abgleich einer einzelnen Datei durchgeführt werden, auch wenn der Nachfolger im adaptiven Ring nicht über diese Datei verfügt; es wird einfach ein Abgleich mit dem nächsten Teilnehmer vollzogen, der über sie verfügt (s. *Abb.5*).

In dem dargestellten Beispiel sind vier Teilnehmer (1, 2, 3 und 4) in einem adaptiven Ring angeordnet. Teilnehmer 1 und 3 verfügen über die Datei X, Teilnehmer 2 und 4 über die Datei Y (s. *Abb. 5, Bild A*). Ein Datenabgleich z.B. zwischen den Teilnehmern 1 und 2 wäre unsinnig, da sie über keine gemeinsamen Daten verfügen. Im erweiterten WARD-Modell wären in diesem Fall zwei adaptive Ringe vorhanden. Damit fällt zwar ein größerer Verwaltungsaufwand an, es werden aber keine unnötigen Abgleiche und Datentransfers vorgenommen und insgesamt weniger Ressourcen verbraucht.

Sollte einige Zeit später z.B. Teilnehmer 3 doch über Datei Y verfügen, so wird er problemlos in den adaptiven Ring der Datei Y aufgenommen und gleicht seine Version der Datei Y mit den Teilnehmern 2 und 4, seine Version der Datei X nur mit Teilnehmer 1 ab (s. *Abb. 5, Bild B*).

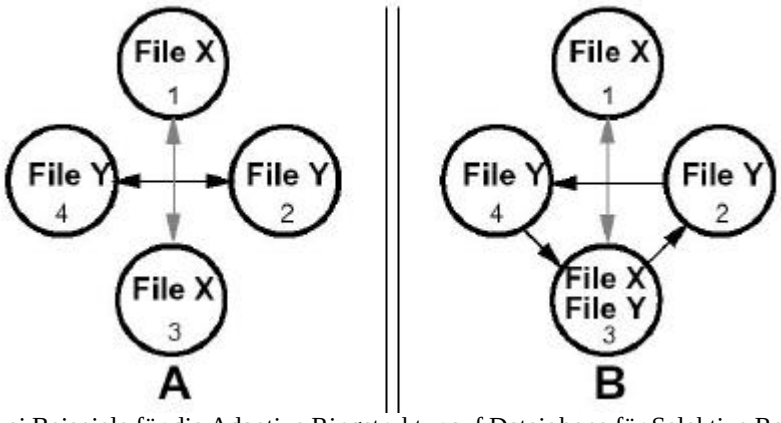

Abb. 5: Zwei Beispiele für die Adaptive Ringstruktur auf Dateiebene für Selektive Replikation.[1]

Die zweite wichtige Erweiterung des erweiterten WARD-Modells ist eine weitere Form der Mobilität. Wenn ein Teilnehmer nur für kurze Zeit in einen anderen WARD übertreten möchte, dann ist es nicht immer sinnvoll, sich komplett von seinem alten WARD abzumelden (wenn z.B. eine mehrtägige Geschäftsreise ansteht und der Teilnehmer nach relativ kurzer Zeit zu seinem ursprünglichen „Heimat-WARD“ zurückkehren wird). Die Alternative dazu ist die so genannte *WARD-Überlappung*. Bei der WARD-Überlappung wird der alte WARD des mobilen Teilnehmers nicht über ein Ausscheiden informiert. Er arbeitet ohne Einschränkung der Funktionalität immernoch mit dem mobilen Teilnehmer weiter. Doch dieser wird gleichzeitig zum neuen WARD auf herkömmliche Weise hinzugefügt und ist für die Zeit der Überlappung Mitglied in zwei WARDs gleichzeitig. Wenn der Teilnehmer wieder ausschließlich Mitglied des alten WARDs werden möchte, so wird er lediglich aus dem neuen WARD abgemeldet. Um zu unterscheiden, ob es zu einer WARD-Überlappung oder einem permanenten WARD-Wechsel kommt, werden Mechanismen angewendet, die z.B. die Dauer der Überlappung messen und dann nachträglich einen WARD-Wechsel durchführen. Der einzige Nachteil für den mobilen Teilnehmer ist die Tatsache, dass er eventuell für die Dauer der WARD-Überlappung mehr Daten als notwendig auf seiner Maschine speichern muss, um weiterhin Mitglied in beiden WARDs zu sein.

5. Literaturverzeichnis

Primärliteratur

[1] RATNER, DAVID H..
Roam: A Scalable Replication System for Mobile and Distributed Computing, 1998.
http://citeseer.nj.nec.com/ratner98roam.html.

[2] GUY, RICHARD G., HEIDEMANN, JOHN S., MAK, WAI, PAGE, THOMAS W., POPEK, GERALD J., ROTHMEIER, DIETER.
Implementation of the FICUS Replicated File System, 1990.
USENIX Conference Proceedings, pp. 63-71.

[3] GUY, RICHARD G., REIHER, PETER, RATNER, DAVID, GUNTER, MICHAIL , MA, WILKIE, POPEK, GERALD.
Rumor: Mobile Data Access Through Optimistic Peer-to-Peer Replication, 1998.
http://fmg-www.cs.ucla.edu/rumor98/replication.html.

Weiterführende Literatur

DROWNING, ALAN R.
Conflict resolution in symmetric replication, 1995.
Proceedings of the European Oracle User Group Conference.

HOWARD, JOHN H.
Using reconciliation to share files between occasionally connected computers, 1993.
Proceedings of the Fourth Workshop on Workstation Operating Systems, pp. 56-60.

KUENNING, GEOFFREY H., BAGRODIA, RAJIVE, GUY, RICHARD G., POPEK, GERALD J., REIHER, PETER, WANG, AN-I.
Measuring the quality of service of optimistic replication, 1998.
Proceedings of the ECOOP Workshop on Mobility and Replication.

NOVELL, INC.
Novell Replication Services white paper
http://www.novell.com/whitepapers/nrs.

PAGE, THOMAS W.,GUY, RICHARD G., HEIDEMANN, JOHN S., RATNER, DAVID H., REIHER, PETER L., GOEL, ASHVIN, KUENNING, GEOFFREY H., POPEK, GERALD J.
Perspectives on optimistically replicated, peer-to-peer filing, 1997
Software – Practice and Experience.

PARKER, DR. STOTT, POPEK, GERALD, RUDISIN, GERARD, STOUGHTON, ALLEN, WALKER, BRUCE J., WALTEN, EVELYN, CHOW, JOHANNA M., EDWARDS, DAVID, KISER, STEPHEN, KLINE, CHARLES.
Detection of mutual inconsistency in distributed systems, 1983.
IEEE Transactions on Software Engineering

POPEK, GERALD J., WALKER, BRUCE J.
The Locus Distributed System Architecture, 1985.
The MIT Press

REIHER, P., PAGE, T., CROCKER, S., COOK, J., POPEK, G.
Truffles – a secure service for widespread file sharing, 1993.
Proceedings of The Privacy and Security Research Group Workshop on Network and Distributed System Security

REIHER, PETER, HEIDEMANN, JOHN S., RATNER, DAVID, SKINNER, GREGORY, POPEK, GERALD J.
Resolving file conflicts in the FICUS file system., 1994.
USENIX Conference Proceedings, pp. 183-195

SATYANARAYANAN, MAHADEV, KISTLER, JAMES J., KUMAR, PUNEET, OKASAKI, MARIA E., SIEGEL, ELLEN H., STEERE, DAVID C..
Coda: A highly available file system for a distributed workstation environment, 1990.
IEEE Transactions on Computers

SYBASE, INC.
Sybase SQL Anywhere and Replication Servers, 1998.
http://www.sybase.com:80/products/system11/repserv.html

TERRY, DOUGLAS B., THEIMER, MARVIN M., PETERSEN, KARIN, DEMERS, ALAN J., SPREITZER, MIKE J., HAUSER, CARL H.
Managing update conflicts in Bayou a weakly connected replicated storage system., 1995.
Proceedings of the 15th Symposium on Operating Systems Principles, pp. 172-183.

WANG, AN-I A., REIHER, PETER L., BAGRODIA, RAJIVE.
A simulation framework for evaluating replicated filing environments, 1997.
Technical Report CSD-970018.

WANG, AN-I A., REIHER, PETER L., BAGRODIA, RAJIVE.
A simulation framework for evaluating replicated filing environments, 1997.
Technical Report CSD-970018.