



GMD Research Series

GMD –
Forschungszentrum
Informationstechnik
GmbH

Gregor Gärtner

Replikationskonzept für ein verteiltes Diskurssystem im Internet

© GMD 2001

GMD – Forschungszentrum Informationstechnik GmbH
Schloß Birlinghoven
D-53754 Sankt Augustin
Germany
Telefon +49 -2241 -14 -0
Telefax +49 -2241 -14 -2618
<http://www.gmd.de>

In der Reihe GMD Research Series werden Forschungs- und Entwicklungsergebnisse aus der GMD zum wissenschaftlichen, nichtkommerziellen Gebrauch veröffentlicht. Jegliche Inhaltsänderung des Dokuments sowie die entgeltliche Weitergabe sind verboten.

The purpose of the GMD Research Series is the dissemination of research work for scientific non-commercial use. The commercial distribution of this document is prohibited, as is any modification of its content.

Anschrift des Verfassers/Address of the author:

Gregor Gärtner
Institut für Autonome intelligente Systeme
GMD – Forschungszentrum Informationstechnik GmbH
Schloß Birlinghoven
D-53754 Sankt Augustin

Die Deutsche Bibliothek – CIP-Einheitsaufnahme:

Gärtner, Gregor:

Replikationskonzept für ein verteiltes Diskurssystem im Internet /
Gregor Gärtner. GMD – Forschungszentrum Informationstechnik GmbH. -
Sankt Augustin : GMD – Forschungszentrum Informationstechnik, 2001
(GMD Research Series ; 2001, No. 13)
Zugl.: Ilmenau, Techn. Univ., Diplomarbeit, 2001
ISBN 3-88457-396-9

ISSN 1435-2699

ISBN 3-88457-396-9

Zusammenfassung

Neue Organisations- und Kooperationsformen in Politik und Wirtschaft führen zu Diskursen mit einer hohen Anzahl von räumlich und zeitlich verteilten Akteuren. In einem solchen Umfeld bieten sich verteilte Diskurssysteme an, die sachbezogene Kommunikation zwischen den Akteuren zu unterstützen. Das Internet ist auf Grund seiner weiten Verbreitung und kostengünstigen Verwendbarkeit als Kommunikationsmedium für diese Systeme prädestiniert.

Die hohe Anzahl zu bedienender Akteure eines Diskurssystems erfordert den Einsatz von Replikationstechniken, um einen Dienst mit hoher Verfügbarkeit und Performanz zu erbringen. Dabei ist auch die Gruppe mobiler Akteure zur Erschließung neuer Anwendungsfelder für Diskurssysteme zu unterstützen. Replikationskonzepte zur Realisierung dieser Anforderungen bilden den thematischen Rahmen dieser Arbeit.

In einer problemorientierten Analyse des Standes der Kunst werden auf der Basis einer klaren Anforderungsmodellierung die verwendbaren und die unzulänglichen Teile existierender Replikationskonzepte identifiziert. Es wird gezeigt, daß bei Verwendung des Internets als Kommunikationsmedium Replikationskonzepte mit schwacher Konsistenz zu bevorzugen sind. Diese werden in den untersuchten Publikationen jedoch nur in Ausschnitten eines jeweiligen Gesamtkonzepts betrachtet. Damit wird die Ausarbeitung eines Gesamtkonzepts zur vorrangigen Aufgabe und die Identifikation seiner signifikanten Aspekte samt ihren Interdependenzen rückt in das Zentrum dieser Arbeit. Das Ergebnis ist eine Darstellung des entworfenen Replikationskonzepts aus unterschiedlichen Perspektiven:

Das Systemmodell beschreibt die Systemarchitektur und liefert einen Überblick über die Ideen und Teilkonzepte, die dem Replikationskonzept zugrunde liegen. Im Konsistenzmodell werden die Konsistenzbeziehungen zwischen den verteilten Replikatabständen, ihren Veränderungen und den nebenläufig erteilten Änderungsaufträgen unter Verwendung der Prädikatenlogik 1. Stufe formal spezifiziert. In der zugehörigen Konfliktbehandlung wird der Konflikt-Begriff um die Kontextdiskrepanz erweitert. Im Abgleichmodell wird ein Verfahren entwickelt, welches mit partieller Replikation auch mobile Replikatverwalter unterstützt. Mit einem Algorithmus für

die Auswahl von Abgleichpartnern realisiert es einen zeitoptimierten Abgleichprozeß. Im Fehlermodell wird dargelegt, daß auf dem entwickelten Replikationskonzept basierende Systeme durch eine zunehmende Anzahl von Fehlern nur allmählich in ihrer Funktionalität reduziert werden, ohne abrupt auszufallen. Aspekte der Mobilität, die über die im Konsistenz- und Abgleichmodell behandelten Gesichtspunkte hinausgehen, sind Gegenstand des Mobilitätsmodells. Schließlich werden im Nebenläufigkeitsmodell die im Replikationskonzept erfüllten Anforderungen an die Nebenläufigkeit der Systemkomponenten für hohe Performanz dargestellt.

Im letzten Kapitel wird eine Realisierung des Gesamtmodells unter Verwendung der Specification and Description Language (SDL) vorgestellt.

Schlagworte: Replikation, Konsistenz, Diskurssysteme, Mobile Computing, Verteilte Systeme

Abstract

New forms of cooperation in politics and economics lead to discourses with a high number of spatially and temporally distributed actors. In such an environment, distributed discourse systems offer an excellent way to support goal-directed communication between such actors. The Internet has evolved due its wide dissemination and low-cost usability and therefore provides a natural communication medium for these systems.

The high number of actors to be served by a discourse system requires the use of replication techniques that can provide a service with high availability and performance. Supporting mobile actors will open up new areas of application for distributed discourse systems. Replication concepts suitable to our requirements form the thematic background of this work.

In a problem-oriented analysis of the state of the art, the applicable parts of existing replication concepts are identified on the basis of our requirements. It is shown that, in case of using the Internet as the communication medium, replication concepts with weak consistency are to be preferred. In the examined publications, however, only some aspects of these concepts are treated without relating them to an overall concept. Therefore the creation of an overall concept becomes the major task and the identification of its significant aspects along with their interdependencies moves into the foreground of this work. As a result, the developed replication concept is presented from different perspectives:

The system model describes the system architecture and provides an overview of the main ideas of the replication concept. In the consistency model the consistency relationships between the distributed replica bases, their modifications and concurrently issued update requests are specified formally with first-order predicate logic. In the corresponding conflict detection and resolution procedure the notion of conflict is expanded to include the notion of context discrepancy. In the reconciliation model a procedure is developed which also supports mobile replica managers with partial replication. It realizes a time-optimized reconciliation process with an algorithm for the selection of reconciliation partners. In the error model it is demonstrated that the functionality of systems based on the replication concept is reduced

incrementally by an increasing numbers of errors without breaking down abruptly. Aspects of mobility that go beyond the issues dealt with in the consistency and reconciliation model are addressed in the mobility model. Finally the concurrency requirements placed on the system components in order to achieve high performance are described in the concurrency model.

In the last chapter a realization of the overall model is introduced using the Specification and Description Language (SDL).

Keywords: Replication, Consistency, Discourse Systems, Mobile Computing, Distributed Systems

Danksagung

Ich möchte an dieser Stelle meinem betrieblichen Betreuer Herrn Dr. Gernot Richter danken, der mir als Ansprechpartner immer zur Verfügung stand und mich sehr intensiv betreut hat. Des weiteren möchte ich meinem Hochschullehrer Herrn Prof. Dr. Winfried E. Kühnhauser für die Ermöglichung dieser Diplomarbeit und ihre kritische Begleitung danken.

Mein besonderer Dank gilt meinen Eltern, die mich während des Studiums in jeglicher Art und Weise unterstützt haben.

Inhaltsverzeichnis

1	Einleitung	1
2	Grundlagen	7
2.1	Begriffe	7
2.2	Voraussetzungen	14
2.3	Anforderungen	16
3	Stand der Kunst in der Replikation	19
3.1	Starke Konsistenz	20
3.1.1	Read-One-Write-All	20
3.1.2	Replikationskonzepte mit Quorumalgorithmen	23
3.1.3	Zusammenfassung der Schlußfolgerungen	26
3.2	Schwache Konsistenz	27
3.2.1	Bayou	27
3.2.2	Ficus	29
3.2.3	Coda	31
3.2.4	TACT	32
3.2.5	Zusammenfassung der Schlußfolgerungen	33

4	Modell	35
4.1	System	35
4.2	Konsistenz	44
4.3	Abgleich	60
4.4	Fehler	83
4.5	Mobilität	88
4.6	Nebenläufigkeit	91
5	Realisierung	95
5.1	Darstellungsmittel	96
5.2	System	98
5.3	Replikatverwalter	100
5.4	Nebenläufigkeit und der Verteilungsprozeß	106
5.5	Prozesse zur Realisierung von Abgleichsitzungen	110
6	Schlußfolgerungen und offene Fragen	121
	Abbildungsverzeichnis	125
	Tabellenverzeichnis	126
	Literaturverzeichnis	128

Kapitel 1

Einleitung

Diskurssysteme im Internet gewinnen stetig an Bedeutung. Dies ist zum einen auf den Trend zu globalen Organisationsformen in Politik und Wirtschaft zurückzuführen, zum anderen auf die aktuelle Erkenntnis aus dem Bereich der Soziologie, daß eine qualitativ hochwertige Lösung von komplexen Problemen nicht von wenigen Experten zu erreichen ist, sondern von möglichst vielen Interessierten mit unterschiedlichem Hintergrund. Als Folge entsteht der Bedarf an einer Diskursform, in der eine hohe Anzahl von räumlich und zeitlich verteilten Akteuren eingebunden ist. Traditionelle Formen zur Durchführung von Diskursen auf der Grundlage einer „Face-to-face“ Kommunikation verlieren in diesem Umfeld an Bedeutung. Diskurssysteme bieten sich zur Organisation einer räumlich und zeitlich verteilten, sachbezogenen Kommunikation an, die häufig der konsensorientierten Lösung von Problemen dient. Das Internet ist auf Grund seiner weiten Verbreitung und kostengünstigen Verwendbarkeit als Kommunikationsmedium für diese Systeme prädestiniert.

Diese Arbeit entstand in der *GMD - Forschungszentrum Informationstechnik GmbH*, die ein verteiltes Diskurssystem, das Mediations- und Diskurssystem *Zeno*, entwickelt. Dieses unterstützt Diskurse insbesondere durch gemeinsame Arbeitsräume für eine inhaltsbezogene Ablage und Verknüpfung von Diskursbeiträgen und Dokumenten, die in einer Baumstruktur verwaltet werden. *Zeno* soll in dem europäischen Projekt *Demos* eingesetzt werden, in dessen Kontext diese Diplomarbeit steht. Ziel des Projekts ist der Entwurf eines Softwaresystems zur Unterstützung von Diskursen mit neuartigen Methoden, um eine breite, auch europaweite Beteiligung von Bürgern an politischen Diskursen und Entscheidungen zu fördern und damit eine

Verringerung des Abstands zwischen den Bürgern und den Entscheidungsträgern zu erzielen. Unter Berücksichtigung der spezifischen Anforderungen dieses Projekts ist ein Dienst für mehr als eintausend Klienten zu erbringen, der sich durch eine *hohe Performanz* und *hohe Verfügbarkeit* auszeichnet.

Die Erbringung eines Dienstes mit solchen qualitativen Eigenschaften ist eine anspruchsvolle Aufgabe. Jede einzelne Komponente eines Systems und damit jedes System selbst ist in der Verfügbarkeit und Performanz limitiert. Daher wird versucht, diese Eigenschaften durch den Einsatz zusätzlicher Komponenten im Sinne von Redundanz zu verbessern und die damit einhergehenden Nebeneffekte weitgehend einzugrenzen. Der Einsatz redundanter Komponenten zur Verbesserung der Verfügbarkeit und Performanz wird als Replikation bezeichnet. In dieser Technik werden Artefakte (Replikate) aus Datenbeständen (Replikatbeständen) auf verschiedenen Netzknoten (Replikatverwaltern), die miteinander kommunizieren können, als Kopien in Konsistenzbeziehungen gehalten. Abgleiche bringen abweichende Replikatbestände in Übereinstimmung, welche durch die Bearbeitung von verteilt anfallenden Klientenaufträgen zur Datenänderung entstanden sind.

Es existieren zwei verschiedene Arten von Konzepten zur Realisierung von Replikation: Replikationskonzepte, die *starke* Konsistenz realisieren, stellen sicher, daß aus der Sicht eines Klienten zu jedem Zeitpunkt der gleiche Replikatbestand auf allen Replikatverwaltern, denen ein Klient Aufträge erteilen kann, vorliegt. Damit ist die Replikation für Klienten transparent und alle Klienten scheinen auf einem gemeinsamen Replikatbestand zu arbeiten, in welchem keine durch die Replikation bedingten Konflikte auftreten. Diese Eigenschaften sind nur mit einem hohen Grad an Übereinstimmung der Replikatbestände zu erreichen, der zwangsläufig zu einem hohen Kommunikationsaufwand führt.

Replikationskonzepte, die *schwache Konsistenz* realisieren, lassen hingegen aus der Sicht eines Klienten abweichende Replikatbestände auf verschiedenen Replikatverwaltern zu. In regelmäßigen Abständen wird der Grad an Übereinstimmung der Replikatbestände durch Abgleiche erhöht. Dabei werden Konflikte bekannt, deren Existenz in der nebenläufigen, autonomen Bearbeitung von Klientenaufträgen auf den einzelnen Replikatverwaltern begründet ist. Diese Konflikte sind in einer Konfliktbehandlung aufzulösen. Die genannten Nachteile spielen jedoch in der Anwendungsdomäne verteilter Diskurssysteme eine untergeordnete Rolle, weil weder

vollständige Transparenz noch Konfliktfreiheit gefordert werden; es überwiegen die Vorteile: Eine autonome Bearbeitung von Klientenaufträgen durch einzelne Replikatverwalter kann bei einem geringen Kommunikationsbedarf eine sehr hohe Performanz und Verfügbarkeit erzielen. Dies ist insbesondere bei der Verwendung von Kommunikationsmedien wie dem Internet von Bedeutung, die größere unvorhersehbare Schwankungen in Bandbreite und Latenzzeit aufweisen. Ein weiteres Vorteil von Replikationskonzepten mit schwacher Konsistenz ist ihre Eignung für die Unterstützung *mobiler Nutzer* und mobiler Replikatverwalter. Neben komfortabler Offline-Arbeit werden damit auch neue Formen der Moderation erschlossen, indem sonst online ablaufende Diskurse vorübergehend offline auf einer Tagung oder Besprechung weitergeführt und moderiert werden können. Die Hardware mobiler Nutzer ist durch limitierte Ressourcen gekennzeichnet, insbesondere durch eine geringe Größe des Massenspeichers und eine über längere Zeiträume unsichere Kommunikationsverbindung. Dies motiviert die Einführung einer partiellen Replikation, die es einem mobilen Replikatverwalter zur effizienten Verwendung seiner Ressourcen erlaubt, nur Teile eines Replikatbestands zu replizieren.

Das Ziel dieser Arbeit liegt in der Entwicklung eines Replikationskonzepts für hohe Performanz und Verfügbarkeit bei schwacher Konsistenz für das Diskurssystem. Mobile Nutzer sind durch mobile Replikatverwalter und partielle Replikation zu unterstützen. Weil existierende Replikationskonzepte den Anforderungen nur teilweise genügen, stand die Entwicklung eines eigenen Konzepts zu Beginn dieser Arbeit im Vordergrund. Die Analyse existierender Publikationen zeigte, daß Replikationskonzepte mit schwacher Konsistenz bisher nur in Ausschnitten eines jeweiligen Gesamtkonzepts betrachtet wurden. Damit wurde die Ausarbeitung eines Gesamtkonzepts einschließlich der Identifikation seiner signifikanten Aspekte und deren Interdependenzen die vorrangige Aufgabe.

Die Arbeit beginnt mit der Einführung in die Begrifflichkeit der Replikation. Es folgt eine Betrachtung der Voraussetzungen, die für verteilte Diskurssysteme im Internet gelten, insbesondere für das Projekt Demos. Darauf aufbauend werden die Anforderungen entwickelt.

Auf der Basis dieser Anforderungen erfolgt eine problemorientierte Analyse des Standes der Kunst. Es werden Replikationskonzepte mit starker sowie schwacher

Konsistenz vorgestellt und die bessere Eignung der letzteren für unsere Anforderungen aufgezeigt. Dabei werden Unzulänglichkeiten herausgearbeitet und unpassende Speziallösungen identifiziert. Anschließend wird zusammengefaßt, welche Teilkonzepte aus der existierenden Literatur übernommen werden können und welche originär zu entwickeln sind.

Im darauf folgenden Kapitel wird das eigene Replikationskonzept modelliert und in seiner Gesamtheit betrachtet. Im Systemmodell wird die Systemarchitektur vorgestellt und ein Überblick über die grundlegenden Ideen und Teilkonzepte des Gesamtmodells gegeben.

Im Konsistenzmodell werden die Konsistenzbeziehungen in der Prädikatenlogik 1. Stufe formal spezifiziert. Danach werden Garantien für Klientensitzungen vorgestellt, deren Einhaltung die Konsistenz der Sicht von Klienten erhöht. Anschließend folgt die Beschreibung der Konfliktbehandlung, die Konflikte unter Berücksichtigung der Baumstruktur auflöst.

Im Abgleichmodell wird der Abgleichprozeß im Detail beschrieben. Es werden verschiedene verteilte Algorithmen diskutiert und ihre Schwächen im Kontext der Anforderungen des Abgleichmodells aufgezeigt. Darauf basierend wird das Teilkonzept der Abgleichsitzungen zum paarweisen Abgleich der Replikatverwalter vorgestellt und ein zeitoptimierter Algorithmus zur Auswahl der (stationären) Abgleichpartner entwickelt. Anschließend wird das Teilkonzept der Abgleichsitzungen um die partielle Replikation für mobile Replikatverwalter erweitert.

Im Fehlermodell werden konkrete Aussagen über das Systemverhalten in Fehler-situationen getroffen. Bei zunehmender Anzahl von Fehlern wird das System nur allmählich in seiner Funktionalität (hauptsächlich Performanz) reduziert, ohne abrupt auszufallen (*graceful degradation*). Nach Ausfällen von Netzknoten oder Kommunikationsverbindungen sind keine *Recovery*-Maßnahmen notwendig.

Im Mobilitätsmodell werden die zur Unterstützung mobiler Nutzer notwendigen Aspekte betrachtet, welche über die im Konsistenz- und Abgleichmodell behandelten Gesichtspunkte hinausgehen.

Das Nebenläufigkeitsmodell zeigt schließlich, welche Prozesse unabhängig voneinander ablaufen können. Die daraus resultierende hohe Performanz wird insbesondere durch eine zum Abgleichprozeß nebenläufige Bedienung von Klienten erreicht.

Das letzte Kapitel der Arbeit stellt eine Umsetzung des Modells unter Verwendung der *Specification and Description Language* (SDL) vor. Nach einer kurzen Einführung in SDL wird die Realisierung des Replikationskonzepts in einem System mit seinen einzelnen Komponenten, Prozessen und Nachrichten vorgestellt.

Kapitel 2

Grundlagen

Inhalt

2.1	Begriffe	7
2.2	Voraussetzungen	14
2.3	Anforderungen	16

Dieses Kapitel beginnt mit der Einführung der für diese Arbeit wesentlichen Begriffe. Anschließend folgt die Beschreibung der Voraussetzungen und Anforderungen, die sich im Projekt Demos auf das Diskurssystem Zeno beziehen. Diese sind nicht auf das Projekt Demos und den Einsatz von Zeno beschränkt, sondern gelten, wie das entworfene Replikationskonzept, zu großen Teilen allgemein für verteilte Diskurssysteme im Internet, auch wenn wir unsere Analysen auf das vorliegende Umfeld beschränken.

2.1 Begriffe

Im folgenden werden einige technische Begriffe eingeführt, die im Rahmen dieser Arbeit verwendet werden. Grundbegriffe der Informationstechnologie (wie beispielsweise Klient, Dienst, usw.) werden vorausgesetzt.

Angelehnt an [Sch96, HR83] wird der Begriff Operation wie folgt eingeführt:

OPERATION

Eine **Operation** spezifiziert einen Vorgang, der nach festgelegten Regeln aus einem gegebenen Datenelement ein neues Datenelement erzeugt. Dabei gelten folgende Eigenschaften:

- **Atomarität.** Eine Operation wird vollständig oder überhaupt nicht ausgeführt.
- **Beständigkeit.** Die Auswirkungen einer ausgeführten Operation sind persistent, also in einem nichtflüchtigen Speicher vorhanden.

In dieser Arbeit werden ausschließlich Operationen und keine Transaktionen betrachtet, denn für die vorliegende Anwendung kann auf schwergewichtige Transaktionen und deren Verwaltung verzichtet werden. Ferner gilt für jede (schreibende) Operation, daß deren Ausführung und damit deren Auswirkung bei Bedarf zurückgenommen werden kann.

Angelehnt an [HHB96] sollen die folgenden Begriffe eingeführt werden:

VERFÜGBARKEIT EINES DIENSTES

Die **Verfügbarkeit eines Dienstes** bezeichnet die Wahrscheinlichkeit, daß ein potentieller Klientenauftrag zu einem Zeitpunkt t angenommen werden kann.

ZUVERLÄSSIGKEIT EINES DIENSTES

Die **Zuverlässigkeit eines Dienstes** bezeichnet die Wahrscheinlichkeit, daß ein zu einem Zeitpunkt t angenommener Klientenauftrag mit einer Bearbeitungsdauer τ erfolgreich durchgeführt und beendet werden kann.

VERLÄSSLICHKEIT EINES DIENSTES

Die **Verlässlichkeit eines Dienstes** bezeichnet die Wahrscheinlichkeit, daß zu einem Zeitpunkt t ein Klientenauftrag angenommen und mit einer Bearbeitungsdauer τ erfolgreich beendet werden kann. Die Verlässlichkeit ist das Produkt von Verfügbarkeit und Zuverlässigkeit.

ANTWORTZEIT EINES DIENSTES

Die **Antwortzeit eines Dienstes** bezeichnet die mittlere Zeitspanne, die zwischen dem Eintreffen eines Auftrags und der Beendigung seiner Ausführung liegt.

DURCHSATZ EINES DIENSTES

Der *Durchsatz eines Dienstes* bezeichnet die mittlere Anzahl von Aufträgen, die ein Dienst in einer Zeitspanne annehmen, ausführen und beenden kann.

PERFORMANZ EINES DIENSTES

Die *Performanz eines Dienstes* bezeichnet die Leistung eines Dienstes als gewichteten Wert von Antwortzeit und Durchsatz des Dienstes.

Die Begriffe im Umfeld der Replikation werden im Rahmen dieser Arbeit wie folgt eingeführt:

REPLIKATION, REPLIKAT, REPLIKATVERWALTER, REPLIKATBESTAND, KONSISTENZBEZIEHUNGEN

Artefakte, die in der Technik der *Replikation* als Kopien auf unterschiedlichen Netzknoten in *Konsistenzbeziehungen* gehalten werden, heißen *Replikate*. Die Netzknoten werden als *Replikatverwalter* bezeichnet, ihr lokaler Datenbestand als *Replikatbestand*.

Die Technik der Replikation wird in verteilten Systemen vor allem eingesetzt, um eine hohe Verlässlichkeit und hohe Performanz von Diensten zu erreichen, die Klienten erbracht werden. Die Softwarekomponenten zur Diensterbringung sind auf jedem Replikatverwalter vorhanden. Die zur Diensterbringung notwendigen Artefakte sind auf den Replikatverwaltern als Replikate verteilt, die in Konsistenzbeziehungen, modelliert in einem *Konsistenzmodell*, zueinander stehen. Konsistenz bezieht sich im Zusammenhang dieser Arbeit immer auf die Übereinstimmung von Replikaten, also Exemplaren und nicht auf das Einhalten von Strukturvorgaben, wie beispielsweise in einem Datenbankschema. Die Artefakte, deren Replikation in dieser Arbeit betrachtet wird, sind entweder einfache Datenelemente oder Bäume. Ein Datenelement kann dabei als Spezialfall eines Baumes der Höhe Null aufgefaßt werden. Als *korrespondierend* werden Replikate bezeichnet, deren Wurzeln den gleichen Kennzeichner haben. Klientenaufträge¹ ändern direkt oder indirekt die Replikatbestände der Replikatverwalter. Die Existenz unterschiedlicher Replikate ist dem Nutzer im Sinne der Transparenz möglichst zu verbergen.

¹In dieser Arbeit auch als Aufträge bezeichnet.

Folgende Begriffe sollen für das Abgleichen von Replikatbeständen und deren maximal zugelassener Abweichung eingeführt werden:

ABGLEICHPROZESS, ABGLEICHSITZUNG

Der **Abgleichprozeß** ist ein Vorgang, in welchem die Replikatverwalter regelmäßig ihre Replikatbestände durch einen Wissensaustausch abgleichen. Ein Vorgang im Abgleichprozeß, in welchem genau zwei Replikatverwalter ihre Replikatbestände durch einen Wissensaustausch abgleichen, wird **Abgleichsitzung** genannt.

Angelehnt an [CDK94, Mul89]:

EIN-EXEMPLAR-SERIALISIERBARKEIT

Zwei Operationen haben die Eigenschaft der **Ein-Exemplar-Serialisierbarkeit**, wenn ausgehend vom selben Replikatbestand die nebenläufige Ausführung einer Operation O_x auf einem Replikatverwalter R_x und einer Operation O_y auf einem Replikatverwalter $R_{y \neq x}$ nach einer Abgleichsitzung der beiden Replikatverwalter das gleiche Ergebnis liefert wie eine Ausführung von O_x und O_y in beliebiger Reihenfolge nacheinander auf einem der beiden Replikatverwalter.

STARKE KONSISTENZ, SCHWACHE KONSISTENZ

Ein Replikationskonzept realisiert **starke Konsistenz**, wenn jede ausgeführte Operation bezüglich jeder anderen ausgeführten Operation die Eigenschaft der Ein-Exemplar-Serialisierbarkeit erfüllt. Jedes Replikationskonzept, welches keine starke Konsistenz realisiert, realisiert **schwache Konsistenz**. Diese beiden Konzepte werden im folgenden als Replikationskonzepte mit starker bzw. schwacher Konsistenz bezeichnet.

Das regelmäßige Abgleichen der unterschiedlichen Replikatbestände der verschiedenen Replikatverwalter wird mit dem Begriff Abgleichprozeß bezeichnet. Ein möglichst zügiger und vollständiger Abgleich ist häufig im Interesse der Klienten eines Dienstes. Im Idealfall ist das Ergebnis eines Klientenauftrags unabhängig von den bearbeitenden Replikatverwaltern, beispielsweise weil die Replikatbestände aller Replikatverwalter gleich sind. Dies wird von Replikationskonzepten mit starker Konsistenz zu jedem Zeitpunkt gewährleistet. Replikationskonzepte mit schwacher Konsistenz erheben diesen Anspruch nicht und es können durch die Nebenläufigkeit von Aufträgen *Konflikte* auftreten, die in einer *Konfliktbehandlung* aufgelöst werden. Dafür erzielen solche Replikationskonzepte, insbesondere bei Kommunikationsmedien mit größeren unvorhersehbaren Schwankungen in Bandbreite und Latenzzeit,

eine höhere Performanz und Verfügbarkeit als Replikationskonzepte mit starker Konsistenz. Diese benötigen einen häufigeren Abgleich der Replikatbestände mit einem resultierenden hohen Kommunikationsbedarf. Dadurch verringert sich im regulären Betrieb die Performanz und im Fehlerfall die Verfügbarkeit. Replikationskonzepte mit schwacher Konsistenz werden auch als Replikationskonzepte mit optimistischer Replikation, Replikationskonzepte mit starker Konsistenz auch als Replikationskonzepte mit pessimistischer Replikation bezeichnet. Diese Begrifflichkeit ist angelehnt an die Formen der Nebenläufigkeitskontrolle von verteilten Transaktionen.

VOLLSTÄNDIGE REPLIKATION, PARTIELLE REPLIKATION

*Bei **vollständiger Replikation** existieren auf jedem Replikatverwalter nach einem vollständigen systemweiten Abgleich die gleichen Replikatbestände. Dies gilt nicht für **partielle Replikation**, weil hierbei Replikate im System existieren dürfen, welche nicht in dem Replikatbestand eines jeden Replikatverwalters vorhanden sind.*

Replikatverwalter, die im fehlerfreien Fall eine ständige Verbindung zum Kommunikationsmedium haben, welches die Replikatverwalter und Klienten verbindet, werden *stationäre* Replikatverwalter genannt. Im Gegensatz dazu sind *mobile* Replikatverwalter solche, die vorübergehend keine Verbindung zum Kommunikationsmedium haben können und oftmals mit geringen Ressourcen (Speicherplatz des Massenspeichers, eine über längere Zeiträume unsichere Kommunikationsverbindung mit geringer Bandbreite) ausgestattet sind. Diese Eigenschaften motivieren den Einsatz partieller Replikation, damit nur solche Replikate repliziert werden können, welche für die zu erfüllende Aufgabe eines mobilen Replikatverwalters benötigt werden. Dadurch wird der Bedarf an Massenspeicher und Kommunikation reduziert.

Weil ein Replikationskonzept auch zur Erhöhung der Verfügbarkeit dient, sind Aussagen über tolerierbare Fehler von Bedeutung, die in einem Fehlermodell getroffen werden. Das Auftreten von Fehlern erschwert die Konsistenzgewährleistung der Replikate im allgemeinen erheblich. Es existieren zwei Fehlerklassen in verteilten Systemen: *Fail-Stop*-Fehler (auch *Fail-Silent*-Fehler genannt) und Byzantinische Fehler, die jeweils bei Netzknoten und Kommunikationsverbindungen auftreten können. Diese werden angelehnt an [Lyn96] wie folgt eingeführt:

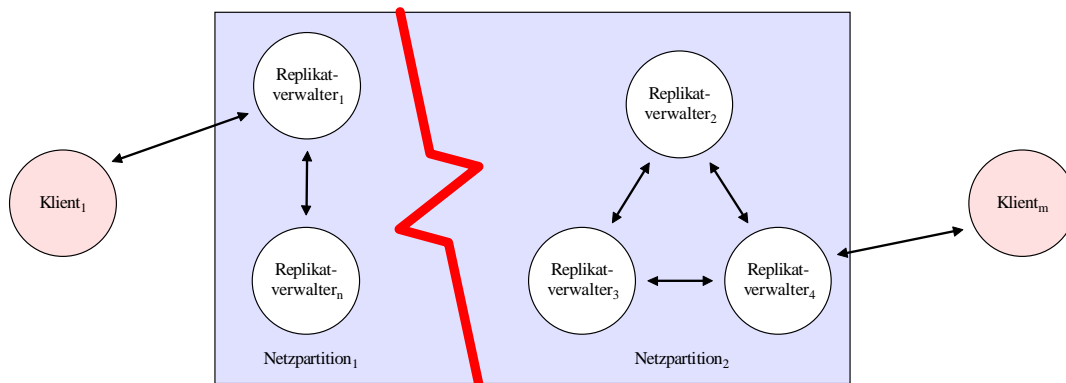


Abbildung 2.1: Beispiel einer Netzpartitionierung

FAIL-STOP-FEHLER EINER KOMMUNIKATIONSVERBINDUNG

Eine Kommunikationsverbindung arbeitet gemäß ihrer Spezifikation oder fällt vollständig aus. Ein solcher Fehler wird als **Fail-Stop-Fehler einer Kommunikationsverbindung** bezeichnet.

BYZANTINISCHER FEHLER EINER KOMMUNIKATIONSVERBINDUNG

Eine Kommunikationsverbindung kann abweichend von der Spezifikation willkürlich erzeugte Daten übertragen, zu übertragende Daten ändern und zu übertragende Daten löschen. Ein solcher Fehler wird als **Byzantinischer Fehler einer Kommunikationsverbindung** bezeichnet.

FAIL-STOP-FEHLER EINES REPLIKATVERWALTERS

Ein Replikatverwalter arbeitet gemäß seiner Spezifikation oder fällt vollständig aus. Ein solcher Fehler wird als **Fail-Stop-Fehler eines Replikatverwalters** bezeichnet.

BYZANTINISCHER FEHLER EINES REPLIKATVERWALTERS

Ein Replikatverwalter kann abweichend von der Spezifikation in einen willkürlichen Zustand wechseln und willkürliche Operationen ausführen. Ein solcher Fehler wird als **Byzantinischer Fehler eines Replikatverwalters** bezeichnet.

NETZPARTITION, NETZPARTITIONIERUNG

Eine Fehlersituation, in welcher Fail-Stop-Fehler von Kommunikationsverbindungen dazu führen, daß Replikatverwalter in verschiedene Gruppen, sogenannte **Netzpartitionen**, unterteilt werden, in welchen die Replikatverwalter nur innerhalb der Gruppe kommunizieren können, wird als **Netzpartitionierung** bezeichnet.

Im Rahmen dieser Arbeit werden ausschließlich partiell synchrone Systeme im Sinne von [Lyn96] betrachtet, so daß Fail-Stop-Fehler von Replikatverwaltern und Kommunikationsverbindungen detektierbar² sind. Es erfolgt keine Berücksichtigung von Byzantinischen Fehlern, die auf Grund der hohen Schwierigkeit und der hohen Anzahl offener Fragen dieses Themenbereichs in verteilten Systemen außerhalb des Rahmens dieser Arbeit liegen. Eine besondere Bedeutung kommt bei Fail-Stop-Fehlern von Kommunikationsverbindungen der Netzpartitionierung (Abbildung 2.1) zu, weil diese von einer Vielzahl von Replikationskonzepten nicht toleriert wird.

Für eine Beschreibung von gerichteten Bäumen, kurz Bäume genannt, die im Diskurssystem als Datenstruktur verwendet werden, sollen die folgenden Begriffe eingeführt werden [AU96, Hei77, Jun94]:

GERICHTETER BAUM, KNOTEN, WURZEL

*Ein gerichteter Baum ist ein kreisfreier, schwach zusammenhängender Digraph aus einer Menge von **Knoten** und gerichteten Kanten, für den gilt:*

1. *Ein Knoten ist hervorgehoben und heißt **Wurzel**.*
2. *Die Wurzel ist der einzige Knoten, von welchem aus jeder andere Knoten durch genau einen Weg erreichbar ist.*

VATER, SOHN

*Sind zwei Knoten m_1, m_2 mit einer Kante verbunden, so wird der Anfangsknoten m_1 als **Vater** von m_2 und der Endknoten m_2 als **Sohn** von m_1 bezeichnet.*

PFAD, LÄNGE

*Sei m_1, m_2, \dots, m_k eine Folge von Knoten in einem Baum. Dabei sei m_1 der Vater von m_2 , welcher wiederum der Vater von m_3 ist, und so weiter bis m_{k-1} , also dem Vater von m_k . m_1, m_2, \dots, m_k wird als **Pfad** von m_1 zu m_k im Baum bezeichnet. Die **Länge des Pfades** ist $k - 1$.*

²Eine ausführliche Betrachtung der Erkennung von Fail-Stop-Fehlern erfolgt in Abschnitt 4.4.

NACHFOLGER, VORGÄNGER

Wenn m_1, m_2, \dots, m_k einen Pfad in einem Baum bilden, dann wird m_1 als **Vorgänger** von m_k bezeichnet, und der Knoten m_k als **Nachfolger** von m_1 . Besitzt der Pfad eine Länge größer oder gleich 1, so wird m_1 als **echter** Vorgänger von m_k und m_k als **echter** Nachfolger von m_1 bezeichnet.

UNTERBAUM

In einem Baum B wird ein Knoten m mit allen seinen echten Nachfolgern als **Unterbaum** von B bezeichnet.

BLATT

Ein Knoten, welcher keine echten Nachfolger hat, wird als **Blatt** bezeichnet.

HÖHE EINES BAUMES

In einem Baum entspricht die Höhe eines Knotens n der Länge des längsten Pfades von n zu einem Blatt. Die **Höhe des Baumes** ist die Höhe der Wurzel.

2.2 Voraussetzungen

Die Anwender sind im Projekt Demos im europäischen Raum verteilt und an das Internet angeschlossen, welches wir in Anlehnung an [Car99] als Kommunikationsmedium ansehen, welches größere nicht vorhersehbare Schwankungen in Bandbreite und Latenzzeit aufweist. Diese Eigenschaften sind unter anderem bedingt durch die Heterogenität des Netzes, die unterschiedlichen Administrationsdomänen und Charakteristika der Subnetze, welche durch einfache *ISDN*-Leitungen bis hin zu breitbandigen *ATM*-Netzen verbunden sind, sowie die zeitlich stark fluktuierende Anzahl von Nutzern. Als Faustregel wird angenommen, daß bei zunehmender geographischer Entfernung zweier Kommunikationspartner sowohl die Bandbreite geringer wird als auch die Latenzzeit steigt, weil eine höhere Anzahl von Netzknoten in unterschiedlichen Subnetzen für eine Übertragung notwendig ist. Im allgemeinen wird daher eine hohe Performanz zu erreichen sein, wenn die Nutzer einen geographisch möglichst nahen Replikerverwalter zur Verfügung haben. Im Projekt Demos sollen ungefähr zehn stationäre Replikerverwalter europaweit verteilt eingesetzt werden. Es wird angenommen, daß die Anzahl mobiler Replikerverwalter weit über der Anzahl der stationären Replikerverwalter liegt.

Für den Entwurf des Replikationskonzepts ist das Verhältnis von schreibenden zu lesenden Klientenaufträgen von Nutzern bedeutsam. Weil im derzeitigen Projektstand des Diskurssystems Zeno noch keine Statistiken über Klientenaufträge vorliegen, wird das Datenmaterial eines *Bulletin Boards* [Git00] ausgewertet, welches eine ähnliche Funktionalität wie Zeno und eine ähnliche inhaltliche Ausrichtung der Diskussionen wie im Projekt Demos hat. Das Bulletin Board besaß im Meßzeitraum ungefähr 1000 aktive Nutzer. In Demos wird diese Anzahl für die Startphase angesetzt und soll sich im Laufe des Projekts auf bis zu einige tausend Nutzer erhöhen. In dem Diagramm 2.2 wurde die relative Häufigkeit von Lese- zu Schreibaufträgen einer Woche aufgetragen, wobei Schreibaufträge in allen Diagrammen in roter Farbe und Leseaufträge in blauer Farbe dargestellt sind. Es ist zu erkennen, daß die Anzahl der Schreibaufträge konstant unterhalb von zehn Prozent der Leseaufträge liegt. Dies gilt auch für das Verhältnis von Lese- zu Schreibaufträgen über einen längeren Zeitraum, in diesem Fall neun Monate, in welchem fast 1.5 Millionen Aufträge erteilt wurden, so daß eine ausreichende statistische Sicherheit für die Aussage zugrunde liegt. Der prozentuale Anteil der Schreibaufträge liegt in diesem Zeitraum bei sechs Prozent.

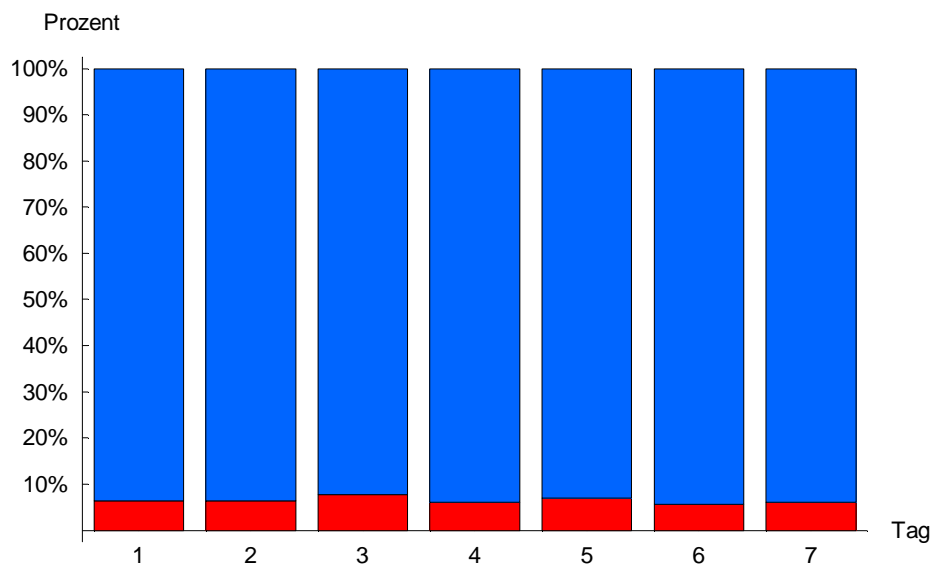


Abbildung 2.2: Relative Häufigkeit von Lese- zu Schreibaufträgen während einer Woche

Für den Entwurf eines Replikationskonzepts ist nicht nur die relative, sondern auch die absolute Anzahl von Schreibaufträgen von Bedeutung. Bei ungefähr tausend

eingetragenen Nutzern liegt die absolute Häufigkeit der Schreibaufträge bei einigen hundert bis fast eintausend pro Tag, wie es in dem Diagramm 2.3 dargestellt ist.

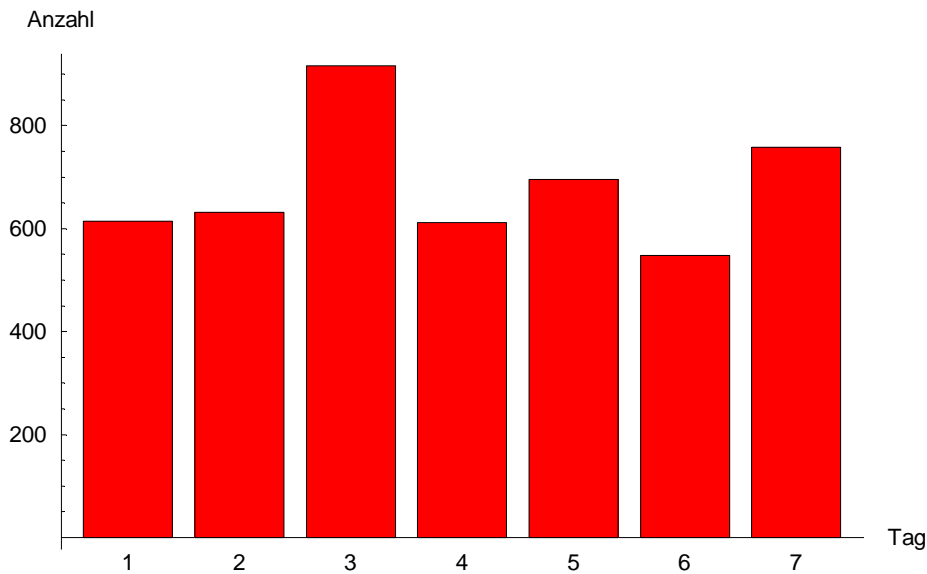


Abbildung 2.3: Absolute Häufigkeit von Schreibaufträgen während einer Woche

2.3 Anforderungen

Die Motivation für den Entwurf eines Replikationskonzepts im Projekt Demos liegt in der Notwendigkeit, einen Dienst für gegebenenfalls einige tausend Klienten zu erbringen, der sich durch eine hohe Performanz, charakterisiert durch geringe Antwortzeiten und hohen Durchsatz, sowie durch eine hohe Verfügbarkeit auszeichnet. Eine Verbesserung der Verfügbarkeit führt auch zu einer Erhöhung der Verlässlichkeit eines Dienstes, wobei wir in unserem Anwendungsumfeld keinen Bedarf sehen, durch zusätzliche Redundanz und den damit verbundenen Ressourcenbedarf die Zuverlässigkeit des Dienstes zu erhöhen, um eine weitere Steigerung der Verlässlichkeit zu erzielen.

Folgende Anforderungen werden unter Berücksichtigung der Voraussetzungen aus Abschnitt 2.2 formuliert:

- Auf Grund der größeren unvorhersehbaren Schwankungen von Bandbreite und Latenzzeit des Internets ist eine geringe Anzahl von auszutauschenden Nachrichten und deren Bündelung von Bedeutung. Die Verfügbarkeit des Dienstes

soll unabhängig von der aktuellen Kommunikationssituation sein, so daß Klientenaufträge ohne einen sofortigen Nachrichtenaustausch der Replikatverwalter zu bearbeiten sind. Damit ist eine *autonome* Annahme und Bearbeitung von Lese- und Schreibaufträgen durch jeden Replikatverwalter erforderlich, was als *Read-Any/Write-Any*-Replikationskonzept bezeichnet wird. Dieses garantiert neben einer sehr hohen Verfügbarkeit auch eine hohe Performanz und Skalierbarkeit. In einem solchen Replikationskonzept kann lediglich schwache Konsistenz realisiert werden, die jedoch für unser Anwendungsumfeld ausreichend ist. Entstehende Konflikte sind unter Einbeziehung des Anwendungskontexts in einer Konfliktbehandlung aufzulösen, wobei die Struktur von Bäumen angemessen zu berücksichtigen ist.

- Wegen der vorausgesetzten Kommunikationscharakteristika entsteht für Klienten ein Performanzvorteil, wenn ihre Aufträge von einem geographisch nahen Replikatverwalter bearbeitet werden.
- Transparenz bei Fail-Stop-Fehlern von Netzknoten und Kommunikationsverbindungen, inklusive einer Netzpartitionierung.
- Eignung für das erwartete Auftragsverhältnis. Es werden deutlich mehr Lese- als Schreibaufträge erwartet, wobei die absolute Häufigkeit von Schreibaufträgen bei mindestens einigen hundert pro Tag liegt.
- Realisierung eines Abgleichprozesses mit einem schnellen (geringe Anzahl der Schritte der Abgleichalgorithmen) und kostengünstigen (geringe Anzahl und Größe der auszutauschenden Nachrichten) Abgleich der Replikatbestände für ungefähr zehn stationäre Replikatverwalter.
- Unterstützung von Bäumen und Wäldern als Datenstruktur, die im Diskurssystem vorgesehen sind.
- Unterstützung von mobilen Nutzern. Damit werden neue Anwendungsfelder wie die Sitzungsmoderation erschlossen, in der ein Moderator in einer Sitzung offline die Gruppenarbeit dokumentiert und diese später online verbreitet. Es wird angenommen, daß sich mobile Nutzer und deren Hardware durch

limitierte Ressourcen (Größe des Massenspeichers, unsichere Kommunikation mit schmaler Bandbreite), Online- und Offline-Phasen vom Internet sowie durch Änderungen der geographischen Position auszeichnen. Diese Eigenschaften motivieren ebenfalls die Entscheidung, ein Read-Any/Write-Any-Replikationskonzept mit schwacher Konsistenz zu verwenden, weil nur diese Replikationskonzepte die Möglichkeit für eine autonome, nebenläufige Arbeit von mobilen Nutzern bieten. Eine Unterstützung dieser Nutzergruppe erfordert des weiteren:

- Eine Unterstützung von *mobilen Replikatorverwaltern*. Ein Nutzer soll den Dienst des Systems auch ohne ständige Verbindung zum Kommunikationsmedium verwenden können. Dies ist nur mit einem mobilen Replikatorverwalter möglich, weil ein Klient keine Funktionalität zur Erbringung des erforderlichen Dienstes hat und keine stationären Replikatorverwalter in einer Offline-Phase verwendet werden können.
 - Eine *partielle Replikation* für eine effiziente Nutzung der Ressourcen (Menge zu übertragender Daten für Abgleichsitzungen, Massenspeicherbedarf des mobilen Replikatorverwalters).
 - Eine leichtgewichtige Erzeugung von mobilen Replikatorverwaltern für eine hohe Flexibilität von mobilen Nutzern, so daß dieser Vorgang ohne administrative Unterstützung durchgeführt werden kann.
 - Die Einführung von Garantien für Klientensitzungen, die einen möglichst transparenten Wechsel der Replikatorverwalter für eine Auftragsannahme und -bearbeitung erlauben.
- Bearbeitung von Klientenaufträgen bei nebenläufiger Durchführung von Abgleichsitzungen, um ohne eine Verringerung der Verfügbarkeit durch einen häufigen Abgleich möglichst übereinstimmende Replikatorbestände der Replikatorverwalter zu erzielen.

Kapitel 3

Stand der Kunst in der Replikation

Inhalt

3.1	Starke Konsistenz	20
3.1.1	Read-One-Write-All	20
3.1.2	Replikationskonzepte mit Quorumalgorithmen	23
3.1.3	Zusammenfassung der Schlußfolgerungen	26
3.2	Schwache Konsistenz	27
3.2.1	Bayou	27
3.2.2	Ficus	29
3.2.3	Coda	31
3.2.4	TACT	32
3.2.5	Zusammenfassung der Schlußfolgerungen	33

In diesem Kapitel wird ein Überblick über den Stand der Forschung in der Replikation gegeben und es werden wesentliche Replikationskonzepte im Kontext unserer Anforderungen diskutiert. Dabei erfolgt eine Klassifikation in Replikationskonzepte, die starke oder schwache Konsistenz realisieren.

3.1 Starke Konsistenz

Replikationskonzepte mit starker Konsistenz sind bereits seit Ende der 70er Jahre aus dem Bereich der verteilten Datenbanken bekannt. Diesen Replikationskonzepten liegen einfache Prinzipien zugrunde und sie sind deshalb kostengünstig zu realisieren. Ihre intellektuelle Beherrschbarkeit ermöglicht es, die Korrektheit der zugrunde liegenden Algorithmen formal nachzuweisen. Sie haben die Eigenschaft der Ein-Exemplar-Serialisierbarkeit und sind damit besonders geeignet, um in der Anwendungsdomäne von Banken und Versicherungen eingesetzt zu werden, in der Konflikte nicht zugelassen sind.

3.1.1 Read-One-Write-All

Es gibt eine Vielzahl von Replikationskonzepten, die in die Klasse der *Read-One-Write-All*-Replikationskonzepte (ROWA) fallen. Ihnen ist gemeinsam, daß sie bei Leseaufträgen nur einen Replikatverwalter kontaktieren, während bei den Schreibaufträgen alle oder nur die verfügbaren Replikatverwalter kontaktiert werden. Damit ist eine gute Performanz in solchen Systemen gegeben, deren Nutzer kaum Schreibaufträge erteilen, wie es beispielsweise bei Verzeichnisdiensten der Fall ist. In unserem Anwendungsumfeld liegt das Verhältnis von Schreib- zu Leseaufträgen jedoch deutlich über dem Auftragsverhältnis der Verzeichnisdienste. Unterstützt durch die hohe erwartete Anzahl von Schreibaufträgen pro Tag wird die Schlußfolgerung gezogen, daß der Kommunikationsbedarf der Replikationskonzepte der ROWA-Klasse nicht unseren Anforderungen entspricht.

Einfaches-ROWA In diesem einfachen Replikationskonzept können Leseaufträge von jedem einzelnen Replikatverwalter autonom bearbeitet werden, während Schreibaufträge von allen Replikatverwaltern zu bearbeiten sind. Die gute Performanz bei fast ausschließlich vorhandenen Leseaufträgen sowie die einfache Realisierung sind die Vorteile dieses Replikationskonzepts. Der Hauptnachteil liegt in einer fehlenden Toleranz von Fail-Stop-Fehlern der Netzknoten und Kommunikationsverbindungen, weil bereits die fehlende Verfügbarkeit eines einzelnen Replikatverwalters dazu führt, daß keine Schreibaufträge angenommen und bearbeitet werden können.

Read-One-Write-All-Available (ROWA-A) Replikationskonzepte dieser Klasse sind die Weiterentwicklung des Replikationskonzepts Einfaches-ROWA. Dessen Hauptnachteil, daß bereits die fehlende Verfügbarkeit eines Replikativverwalters die Bearbeitung von Schreibaufträgen systemweit unterbindet, wird durch die Einschränkung aufgehoben, daß Schreibaufträge nur von allen verfügbaren Replikativverwaltern zu bearbeiten sind. Ist ein Replikativverwalter durch einen Fail-Stop-Fehler ausgefallen, so muß er als Recovery-Maßnahme einen Abgleich mit einem anderen Replikativverwalter ausführen, um seinen Replikativbestand auf den aktuellen Stand zu bringen.

Das bekannteste Replikationskonzept der ROWA-A-Klasse ist das *Available-Copies*-Replikationskonzept [BG84], dessen Hauptnachteil kurz skizziert werden soll. Durch Fail-Stop-Fehler von Kommunikationsverbindungen kann eine Netzpartitionierung entstehen, wie sie beispielsweise in Abbildung 3.1 dargestellt ist. Die konsistente Bearbeitung von Schreibaufträgen ist nur innerhalb jeder Netzpartition möglich, weil die zur Gewährleistung der systemweiten Konsistenz notwendige Kommunikation nicht stattfinden kann. Das in der Abbildung dargestellte Datenelement d_1 hat in der einen Netzpartition den Wert 5, während es in der anderen den Wert 7 hat. Dies ist ein Widerspruch zu den Konsistenzbeziehungen, denn das Datenelement müßte auf allen verfügbaren Replikativverwaltern den gleichen Wert haben. Um Transparenz bei Netzpartitionierung bieten zu können, sind viele Varianten des Available-Copies-Replikationskonzepts entstanden, die diesen Nachteil mit einem erheblich höheren Kommunikationsaufwand beseitigen.

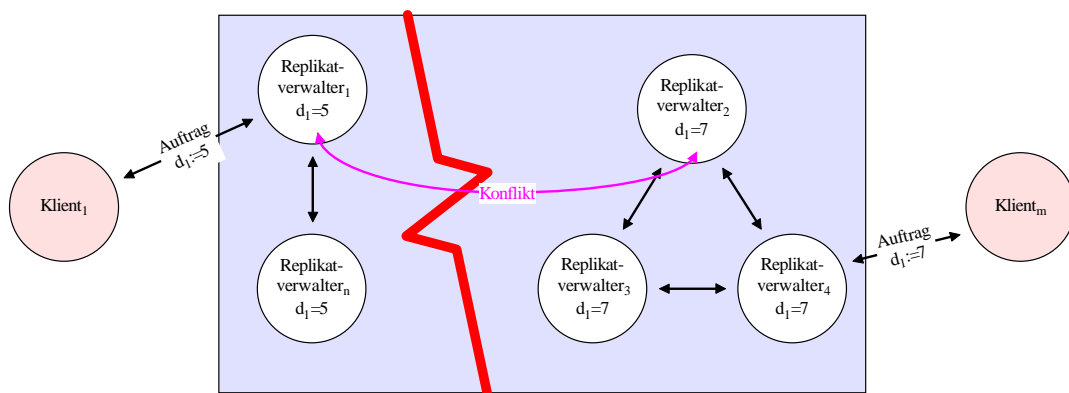


Abbildung 3.1: Available-Copies-Replikationskonzept bei Netzpartitionierung

Primary-Copy-ROWA Dieses Replikationskonzept aus dem Jahr 1976 wird selbst heute noch häufig in kommerziellen Produkten eingesetzt. Das zugrunde liegende Prinzip ist die Aufteilung der Replikatabestände, in welcher ein Replikatabestand als Original, die anderen als Kopien fungieren [AD76]. Ein besonders ausgezeichneter Replikatverwalter, der primäre Replikatverwalter, ist für die Verwaltung des Originals zuständig, während die sekundären Replikatverwalter die Kopien verwalten. Während ein Leseauftrag auf jeder Kopie ausgeführt werden darf, müssen Schreibaufträge von dem *Front-End-Verwalter* eines Klienten immer an den primären Replikatverwalter erteilt werden (Abbildung 3.2). Dieser propagiert die Schreibaufträge an die sekundären Replikatverwalter und bestätigt die Ausführung dem Klienten, wenn alle sekundären Replikatverwalter die erfolgreiche Ausführung gemeldet haben. Fällt der primäre Replikatverwalter aus, wählen die sekundären Replikatverwalter aus ihren Reihen einen neuen primären.

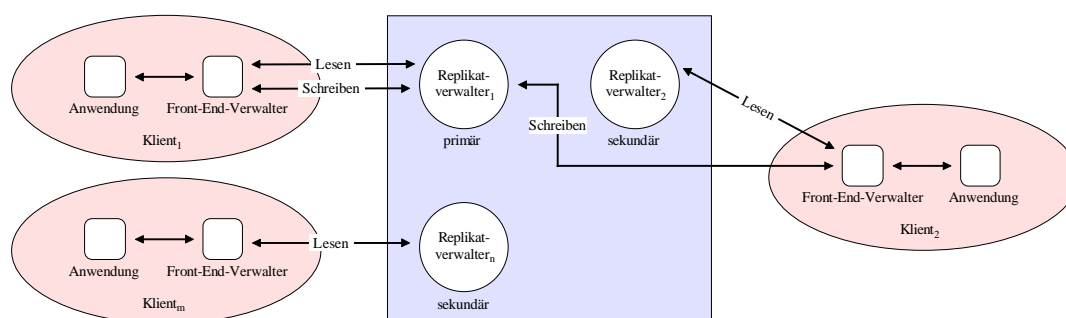


Abbildung 3.2: Primary-Copy-ROWA

Das Replikationskonzept ist einfach zu realisieren und bietet bei vorwiegender Erteilung von Leseaufträgen eine gute Performanz, weshalb es häufig in Verzeichnisdiensten eingesetzt wird. Der Hauptnachteil liegt in der fehlenden Toleranz einer Netzpartitionierung, weil Fail-Stop-Fehler der Kommunikationsverbindungen zu den Replikatverwaltern nicht von Fail-Stop-Fehlern der Netzknoten selbst zu unterscheiden sind. Dies kann zu der Existenz von einem primären Replikatverwalter pro Netzpartition führen, so daß Konflikte entstehen können (Abbildung 3.3).

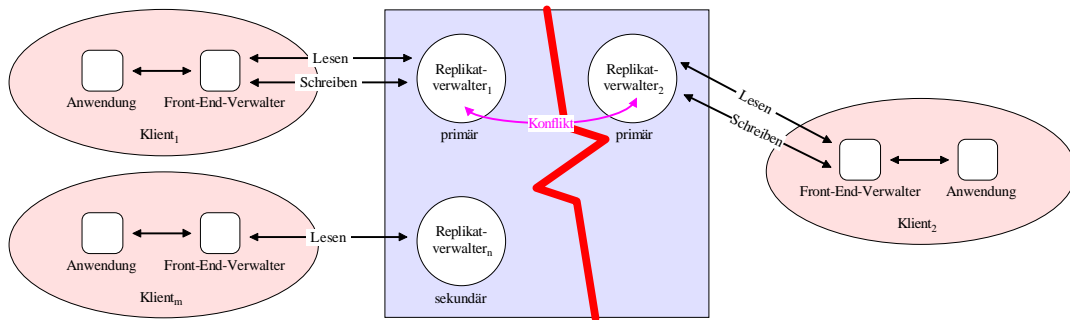


Abbildung 3.3: Primary-Copy-ROWA bei Netzpartitionierung

3.1.2 Replikationskonzepte mit Quorumalgorithmen

Replikationskonzepte, die auf Quorumalgorithmen basieren, beruhen auf dem Prinzip, daß ein Klient für jeden Lese- oder Schreibauftrag die Stimmen einer Abstimmung der Replikerverwalter sammelt, die darüber entscheiden, ob der Auftrag dieses Klienten angenommen oder abgelehnt wird. Erhält ein Klient für einen Auftrag eine vorher festgelegte Mindestanzahl von Stimmen, kann der Auftrag angenommen werden. Die Anzahl der erhaltenen Stimmen wird als *Quorum* bezeichnet.

Es gibt eine Vielzahl von Replikationskonzepten, die in unterschiedlichen Varianten auf Quorumalgorithmen beruhen. Wir wollen an dieser Stelle lediglich auf ein Replikationskonzept auf Basis einer gewichteten Mehrheitsentscheidung [Gif79] eingehen, welches auch heute noch vielfach eingesetzt wird. In diesem Replikationskonzept wird jedem Replikerverwalter eine Anzahl von Stimmen zugewiesen, mit der er sich an Abstimmungen beteiligen darf. Dabei gilt das Prinzip, daß ein Replikerverwalter entweder mit seinen gesamten Stimmen in einer Abstimmung zustimmt oder keine Stimme abgibt. Die Summe der Stimmen aller Replikerverwalter soll durch G repräsentiert werden. Ein Klient muß für die Annahme eines Auftrags ein Quorum von Stimmen sammeln. Die notwendige Mindestanzahl von Stimmen, damit ein Schreibauftrag angenommen wird, wird durch S repräsentiert; analog wird L für einen Leseauftrag verwendet. Replikationskonzepte, die auf Quorumalgorithmen basieren, bieten die Eigenschaft der Ein-Exemplar-Serialisierbarkeit dadurch, daß garantiert wird, daß sich die Quoren zweier beliebiger Schreib- und Leseaufträge überschneiden. Dadurch wird mit der Unterstützung von Versionsvektoren [PPR⁺83] garantiert, daß ein Leseauftrag immer die neueste Version eines Datenelements aus den Replikatbeständen der Replikerverwalter liefert.

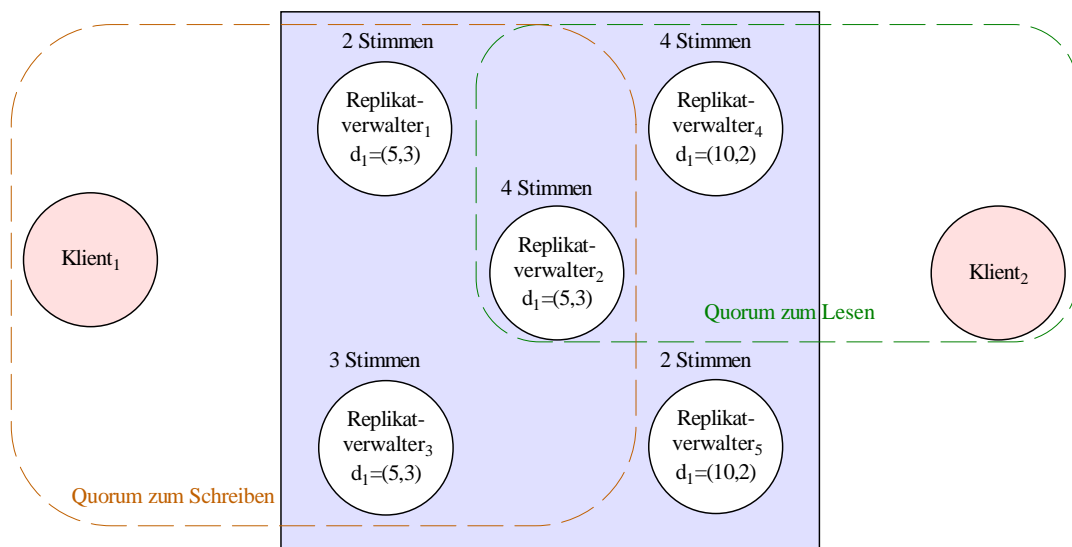
Das Prinzip wird in Abbildung 3.4 visualisiert: Es sei $V := 15$, $S := 9$ und $L := 7$. Das Datenelement d_1 besteht aus einem Tupel¹ der Form (Wert, Versionsnummer). In der Ausgangssituation haben alle Replikativhalter von dem Datenelement d_1 die gleiche Version mit der Versionsnummer 2. Der Klient 1 äußert allen Replikativhaltern den Wunsch, einen Schreibauftrag für das Datenelement d_1 mit der Wertzuweisung 5 zu erteilen. R_1, R_2, R_3 antworten dem Klienten zuerst und das Quorum erfüllt die Bedingung, daß mindestens 9 Stimmen für eine Annahme des Schreibauftrags vorliegen müssen. Daraufhin erteilt der Klient 1 seinen Auftrag allen Replikativhaltern (R_1, R_2, R_3), deren Stimmen in diesem Quorum vorhanden sind. Die Situation nach Ausführung dieses Auftrags ist in der Abbildung dargestellt und nun möchte der Klient 2 das Datenelement d_1 auslesen. Dazu äußert dieser einen Lesewunsch an alle Replikativhalter, wobei R_2 und R_5 als erstes antworten und bei der Antwort die Versionsnummer des zu lesenden Datenelements aus ihrem Replikativbestand mitliefern. Das Quorum enthält genügend Stimmen für eine Annahme des Auftrags und damit kann der Klient 2 den Wert des Datenelements d_1 in der aktuellen Version von R_2 erfragen. Es läßt sich leicht zeigen, daß bei der im Beispiel festgelegten Stimmenverteilung es selbst bei einer Netzpartitionierung nicht möglich ist, Aufträge so zu bearbeiten, daß deren Auswirkungen in Konflikt stehen.

Die Bedingungen für die Bestimmung der Mindestanzahl der Stimmen sind wie folgt:

1. $L + S > G$
2. $S > \lfloor \frac{G}{2} \rfloor$

Die Vorteile dieses Replikationskonzepts liegen in der flexiblen Gewichtung der Stimmen pro Replikativhalter und der Mindestanzahl von Stimmen für die Annahme von Aufträgen. Mit der Zuweisung der Stimmen für jeden Replikativhalter kann deren Bedeutung für Abstimmungen gewichtet werden. Die Mindestanzahl der Stimmen für Aufträge legt fest, wie die Performanz und Verfügbarkeit zwischen Lese- und Schreibaufträgen verteilt wird. So wird eine hohe Verfügbarkeit und Performanz für Leseaufträge erreicht, wenn die Mindestanzahl der Stimmen für deren

¹Die Darstellung eines Datenelements wurde zur besseren Übersichtlichkeit vereinfacht und enthält nicht dessen Kennzeichner.



1. Auftrag Klient 1: Schreibe Wert 5 in d_1
2. Auftrag Klient 2: Lese d_1 (Ergebnis ist 5)

Abbildung 3.4: Gewichteter Quorumalgorithmus

Bearbeitung sehr gering ist. Damit steigt die Mindestanzahl der Stimmen für die Bearbeitung von Schreibaufträgen, so daß deren Performanz und Verfügbarkeit reduziert wird. Viele Replikationskonzepte der ROWA-Klasse sind durch Replikationskonzepte ersetzbar, denen eine entsprechenden Konfiguration eines Quorumalgorithmus zugrunde liegt. So kann das Replikationskonzept Einfaches-ROWA durch einen Quorumalgorithmus mit einer gewichteten Mehrheitsentscheidung ersetzt werden, indem die Konfiguration $S := G$ und $L := 1$ gewählt wird. Ein weiterer Vorteil liegt in der Transparenz bei einer Netzpartitionierung, die keine Konflikte von Aufträgen zur Folge haben kann.

Quorumalgorithmen sind eine gute Wahl für Replikationskonzepte mit starker Konsistenz. Für unsere Anforderungen sind diese jedoch ungeeignet:

- Der Kommunikationsaufwand ist hoch, der durch die Realisierung der starken Konsistenz entsteht. Damit verringert sich die Verfügbarkeit und Performanz im Vergleich zu Read-Any/Write-Any-Replikationskonzepten mit geringem Kommunikationsbedarf, was nur in Anwendungsdomänen gerechtfertigt ist, die starke Konsistenz benötigen.

- Fehlende Unterstützung von mobilen Replikativern. Auf Grund der autonomen Bearbeitung von Aufträgen durch mobile Replikativern bei fehlender Kommunikationsverbindung kann das Prinzip, daß sich alle möglichen Schreibe- und Lesequoren überlagern müssen, nicht eingehalten werden. Daher kann keine starke Konsistenz realisiert werden und es können Konflikte entstehen. Des weiteren müßte eine Rekonfiguration des Systems für jeden mobilen Replikativern erfolgen, der sich in das System ein- oder ausklinkt. Diese Rekonfiguration führt bei einer höheren Anzahl an mobilen Replikativern zu hohem Kommunikations- und Verwaltungsaufwand. Um die Aufträge, die von einem mobilen Replikativern in einer Offline-Phase bearbeitet werden, im System zu verbreiten, ist es notwendig, diese in einer Logdatei zu speichern und abzuspielen, sobald dieser wieder online ist. Damit geht der Vorteil einer relativ einfachen Realisierbarkeit des Replikationskonzepts verloren. Ohne dieses Abspielen wäre das Ergebnis der Ausführung der Aufträge nur sichtbar, wenn der mobile Replikativern online ist, weil keine eigenständige Wissensverbreitung erfolgt. Zusammengefaßt läßt sich die Schlußfolgerung ziehen, daß bei Unterstützung mobiler Replikativern in diesem Replikationskonzept die Vorteile verloren gehen, während die Nachteile erhalten bleiben.

3.1.3 Zusammenfassung der Schlußfolgerungen

Replikationskonzepte mit starker Konsistenz bieten auf Grund ihres Kommunikationsbedarfs eine geringere Verfügbarkeit und Performanz als Read-Any/Write-Any-Replikationskonzepte, die schwache Konsistenz realisieren. Dies ist nur in Anwendungsdomänen gerechtfertigt, in denen Ein-Exemplar-Serialisierbarkeit notwendig ist. Die fehlende Unterstützung für mobile Nutzer ist ein weiterer Grund, warum diese Replikationskonzepte unsere Anforderungen kaum abdecken.

3.2 Schwache Konsistenz

Die Anforderung an die Entwicklung eines Read-Any/Write-Any-Replikationskonzepts mit einem schnellen und kostengünstigen Abgleich bei geringem Kommunikationsbedarf sowie mit einer Unterstützung von mobilen Nutzern prädestiniert die Verwendung von Replikationskonzepten mit schwacher Konsistenz. Diese Replikationskonzepte wurden vor allem in den letzten zehn Jahren entwickelt und sind im Vergleich zu Replikationskonzepten mit starker Konsistenz sehr komplex. Daher beschränken wir uns in diesem Abschnitt auf die Darstellung und Diskussion solcher Aspekte, die Gemeinsamkeiten oder Abgrenzungen von unserer Arbeit aufzeigen. Jedes vorgestellte Replikationskonzept bietet Transparenz für Fail-Stop-Fehler von Netzknoten und Kommunikationsverbindungen inklusive einer Netzpartitionierung.

3.2.1 Bayou

Bayou war ein Projekt des *Xerox Palo Alto Research Center*, das zwischen 1994 und 1997 durchgeführt wurde und die Entwicklung einer Architektur zur Unterstützung von Gruppenarbeit zum Ziel hatte. Der Kern dieser Architektur bildet ein verteiltes persistentes Speichersystem mit einer Replikation der dort abgelegten Daten. Eine Vielzahl einzelner Konzepte² des gesamten Replikationskonzepts ist für unser eigenes Projekt geeignet, weil diese vor allem für ähnliche Anforderungen und eine ähnliche Anwendungsdomäne konzipiert wurden [EMP⁺97]. Die folgenden Konzepte werden als Grundlage unserer Arbeit weiterverwendet:

- Grundprinzip des Abgleichalgorithmus für *inkrementelle* Abgleichsitzungen von Replikativern, wobei lediglich eine vollständige Replikation unterstützt wird [PST⁺97]. Dabei werden Versionsvektoren verwendet, um Abweichungen der Replikatbestände zu erkennen (ein ähnlicher auf Versionsvektoren basierender Ansatz ist in [Hor99] zu finden). Ein sogenanntes *Schreiblog* enthält Informationen über alle Schreiboperationen, die einem Replikativern bekannt sind. Diese Informationen werden benötigt, um in inkrementellen Abgleichsitzungen einem Abgleichpartner nur jene Schreiboperationen zu übermitteln, die diesem unbekannt sind.

²Sprechen wir von einem Konzept, so beziehen wir uns auf einen einzelnen Teil (Teilkonzept) des gesamten Replikationskonzepts, welcher für sich gesehen eigenständig ist.

- Darstellung des Status von Aufträgen. Einem Nutzer wird angezeigt, ob das Ergebnis der Ausführung eines Auftrags nur *vorläufig* oder *endgültig* ist. Das Ergebnis der vorläufigen Ausführung eines Auftrags kann durch nebenläufig bearbeitete Aufträge geändert werden.
- Garantien für Klientensitzungen [TDP⁺94]. Die Einführung von Garantien für Klientensitzungen erhöht die Konsistenz der Sicht eines Klienten. Dies wird durch eine eingeschränkte Auswahl von Replikatverwaltern, die ein Mindestmaß an Wissen über vorhandene Schreibaufträge bezüglich einer Sitzung haben müssen, für eine Annahme von Aufträgen realisiert. Beispielsweise sollte ein Klient nur einen Auftrag zum Lesen an einen solchen Replikatverwalter erteilen, dem die bereits in der Sitzung erteilten Schreibaufträge bekannt sind. Um die Verfügbarkeit des Systems nicht zu verringern, können auch Aufträge an andere Replikatverwalter erteilt werden, wobei einem Klienten angezeigt wird, daß eine Garantie in diesem Fall nicht gegeben werden kann.
- Konzept der leichtgewichtigen Erzeugung von mobilen Replikatverwaltern. Mobile Replikatverwalter müssen für eine flexible Verwendung von mobilen Nutzern ohne administrative Unterstützung erzeugt und entfernt werden können. Bei vielen Replikationskonzepten ist dies nicht möglich, weil diese ausschließlich stationäre Replikatverwalter unterstützen, die selten erzeugt oder entfernt werden.
- Konzept zum Entfernen von Schreiboperationen des Schreiblogs. Selbst bei geringen Kosten für Massenspeicher muß es einem Replikatverwalter möglich sein, nicht mehr benötigte Schreiboperationen aus dem Schreiblog zu entfernen, um den Bedarf an Massenspeicher zu senken. Dies ist beispielsweise für Schreiboperationen sinnvoll, die jedem Replikatverwalter im System bekannt sind, so daß diese in Abgleichsitzungen nicht mehr übertragen werden müssen. Folglich brauchen diese Schreiboperationen nicht im Schreiblog aufbewahrt werden.

Folgende Konzepte fehlen in Bayou, die für die Realisierung unserer Anforderungen notwendig sind:

- Eine Unterstützung von Bäumen. Das Replikationskonzept von Bayou unterstützt lediglich Datenelemente in unstrukturierten Mengen.

- Ein Algorithmus für die Realisierung des Abgleichprozesses, der die Auswahl von Abgleichpartnern für Abgleichsitzungen bei einer Menge von ungefähr zehn Replikativern optimal unterstützt. In Bayou wird die Existenz von mehr als eintausend Replikativern vorausgesetzt. Aus diesem Grund werden epidemische Algorithmen zur Verbreitung der Änderungen der Replikativbestände eingesetzt [DGH⁺87], die für eine Menge von ungefähr zehn Replikativern ungeeignet sind (zu langsame Verbreitung der Änderungen der Replikativbestände, zu hohe Anzahl von auszutauschenden Nachrichten).
- Eine partielle Replikation für die Unterstützung von mobilen Replikativern. Bayou unterstützt lediglich eine vollständige Replikation, die auf Grund der beschränkten Ressourcen mobiler Replikativern ineffizient (bezüglich der Menge zu übertragender Daten für Abgleichsitzungen und des Massenspeicherbedarfs) ist. Es wurde in den Schlußfolgerungen in [EMP⁺97] darauf hingewiesen, daß die Erweiterung um eine partielle Replikation in Bayou ein wichtiger Forschungsgegenstand ist, um mobile Replikativern auch in der Praxis nutzen zu können.
- Eine angemessene Konfliktbehandlung. Bayou unterstützt einen allgemeinen Mechanismus zur Konfliktbehandlung, die von jeder Anwendung zu spezifizieren ist. Dabei lassen sich jedoch nur einzelne Datenelemente und keine Bäume berücksichtigen, die aber für das Diskurssystem notwendig sind.

3.2.2 Ficus

Ficus ist ein Projekt der *University of California Los Angeles*, in dem ein weiträumig verteiltes Dateisystem für eine hohe Anzahl von Replikativern entwickelt wird [PGH⁺98, PGH⁺91]. Das Internet dient als Kommunikationsmedium. Verteilte Dateisysteme sind für diese Arbeit interessant, weil diese ebenso Bäume als Datenstrukturen wie unser Diskurssystem einsetzen.

Der Abgleichprozeß in *Ficus* besteht aus zwei sich ergänzenden Mechanismen [GHM⁺90]: Ändert ein Klient eine Datei und schließt diese, so schickt der Klient eine Änderungsbenachrichtigung (*Update Notification*) an jeden Replikativern, der diese Datei repliziert und sich daraufhin die neue Version der Datei abholt. Dabei wird die vollständige Datei und nicht nur deren Änderungen übertragen. Den

daraus resultierenden *Overhead* von zu übertragenden Daten halten wir für inakzeptabel. Weil nicht gewährleistet wird, daß die Mitteilungen über die Änderung einer Datei auch wirklich bei den adressierten Replikatverwaltern eintreffen (*best effort*), müssen regelmäßig Abgleichsitzungen durchgeführt werden, um schließlich einen vollständigen Abgleich der Replikatbestände zu garantieren. Die durch die Änderungsbenachrichtigungen initiierte sofortige Übertragung geänderter Dateien ist für ein Dateisystem sinnvoll, in welchem eine geringe maximal zulässige Abweichung der Replikatbestände erforderlich ist. In unserem Anwendungsgebiet besteht dieser Bedarf jedoch nicht und der hohe Kommunikationsaufwand durch einen ständigen Nachrichtenaustausch kann nicht motiviert werden. Ficus unterstützt partielle Replikation sowohl für stationäre als auch für mobile Replikatverwalter, in welcher für jede Datei bestimmt werden kann, welche Replikatverwalter diese replizieren [RPR96]. Alle Replikatverwalter, welche dieselbe Datei replizieren, bilden einen virtuellen Ring. Diese virtuellen Ringe bestimmen die Auswahl der Abgleichspartner für Abgleichsitzungen, in denen auch mobile Replikatverwalter in einer Online-Phase einbezogen sind und es bedarf bei n Replikatverwaltern $\mathcal{O}(3n)$ hintereinander durchgeführter Abgleichsitzungen, um die Replikatbestände aller n Replikatverwalter, welche dieselbe Datei replizieren, in Übereinstimmung zu bringen (vorausgesetzt es werden während dieser Abgleichsitzungen keine Aufträge von Klienten bearbeitet). Diese Verbreitungsgeschwindigkeit halten wir für gering. Im Abschnitt 4.3 stellen wir einen originären Algorithmus vor, der mit nebenläufigen Abgleichsitzungen einen schnelleren Abgleich der Replikatbestände in $\mathcal{O}\left(\frac{n}{2} \lceil \log_2 n \rceil\right)$ Sitzungen erreicht. Bei beispielsweise zehn Replikatverwaltern sind in Ficus 30 Sitzungen notwendig, bei uns hingegen nur neun. Weil die Realisierung der partiellen Replikation eng mit den Abgleichalgorithmen verbunden ist, ist eine losgelöste Verwendung dieses Konzepts (beispielsweise auf Grundlage der Konzepte von Bayou) nicht möglich.

Die Konfliktbehandlung in Ficus ist auf Dateisysteme spezialisiert und nicht für unsere Anwendung geeignet. Die Struktur von Bäumen wird nur ungenügend berücksichtigt, weil beispielsweise beim Löschen eines Unterbaumes nicht geprüft wird, ob der gesamte Unterbaum zum Zeitpunkt des Löschens in demselben Zustand ist, in welchem ein Nutzer den Löschauftrag erteilt hat. Der Nutzer hätte den Unterbaum unter Umständen nicht gelöscht, wenn er gewußt hätte, daß nebenläufig ein anderer Unterbaum in den zu löschenden Unterbaum verschoben wurde. Die Konfliktbehandlung beschränkt sich damit auf einzelne Dateien und Verzeichnisse, ohne

tatsächlich die Struktur von Bäumen zu berücksichtigen. Des weiteren unterstützt Ficus keine Anzeige, ob das Ergebnis der Ausführung von Aufträgen nur vorläufig oder endgültig ist.

3.2.3 Coda

Coda war ein Projekt der *Carnegie Mellon University*, in dem ein weiträumig verteiltes Dateisystem als Nachfolger des bekannten *Andrew File System* [MSC⁺86] entwickelt wurde. Die Motivation lag insbesondere in einer Erhöhung der Verfügbarkeit des Dateisystems durch den Einsatz von Replikation [SKK⁺90, KS93].

Im Abgleichprozeß von *Coda* wird eine Variante eines ROWA-A-Replikationskonzepts verwendet, welche schwache Konsistenz unterstützt. Die starke Konsistenz des klassischen ROWA-A-Replikationskonzepts kann durch die Unterstützung der autonomen Arbeit mobiler Nutzer und das Verbergen einer Netzpartitionierung nicht gewährleistet werden. Der Kommunikationsaufwand, um im fehlerfreien Fall eine starke Konsistenz bei stationären Replikatverwaltern bieten zu können, ist jedoch sehr hoch und auf Grund unserer schwachen Anforderungen an die maximal zulässige Abweichung der Replikatbestände nicht zu motivieren. Des weiteren sind die Klienten in diesem Replikationskonzept selbst dafür verantwortlich, daß die Konsistenz der Replikatbestände der stationären Replikatverwalter erhalten bleibt. Diese Entscheidung wurde getroffen, weil die geistigen Väter des Projekts in der Prozessorbelastung der Replikatverwalter den Flaschenhals für ein weiträumig verteiltes Dateisystem sahen. Auf Grund der Anzahl der Replikatverwalter im Projekt *Demos* und der Leistungsfähigkeit heutiger Rechnersysteme sehen wir keinen Anlaß, Softwarekomponenten für die Realisierung des Replikationskonzepts aus einem vertrauenswürdigen stationären Replikatverwalter als Server auf einen nicht vertrauenswürdigen Klienten auszulagern. Die partielle Replikation, die in *Coda* auf Dateiebene angeboten wird, ist so eng mit den Abgleichalgorithmen verbunden, daß eine losgelöste Verwendung (analog zu *Ficus*) nicht möglich ist.

Die Konfliktbehandlung beschränkt sich wie in *Ficus* auf einzelne Dateien und Verzeichnisse, ohne die Struktur von Bäumen zu berücksichtigen. Sitzungsgarantien für Klienten fehlen ebenso wie eine Anzeige, ob das Ergebnis der Ausführung eines Auftrags vorläufig oder endgültig ist.

3.2.4 TACT

TACT ist ein laufendes Projekt der *Duke University* und steht für *Tunable Availability and Consistency Tradeoff* [YV00a, YV00b]. In den bisher vorgestellten Replikationskonzepten war der Grad der maximal zulässigen Abweichung der Replikatbestände durch das Replikationskonzept bestimmt. Eine Änderung dieser maximal zulässigen Abweichung konnte nur durch eine Änderung des Replikationskonzepts selbst verwirklicht werden und war nicht durch eine Rekonfiguration des zugehörigen Systems zu erreichen.

Mit einem steigenden Grad der maximal zulässigen Abweichung steigt die Performanz und Verfügbarkeit, weil der Kommunikationsbedarf verringert wird. Es liegt also ein *Tradeoff* zwischen diesen Eigenschaften vor. Die zunehmende Verbreitung von Internetdiensten im Bereich des *E-Commerce* motiviert eine *gleitende Adaption* der maximal zulässigen Abweichung im laufenden Betrieb, um abhängig von dieser oder der Kommunikationssituation den *optimalen Ausgleich* zwischen der maximal zulässigen Abweichung und der Performanz wählen zu können. Dies soll an einem Beispiel verdeutlicht werden: Das Reservierungssystem der Bundesbahn bietet die Möglichkeit, Platzkarten für einen Zug verbindlich zu reservieren. Eine starke Abweichung der Replikatbestände ist nicht erwünscht, weil ein Buchungskonflikt zur Überbuchung eines Zugs führen kann. Aus diesem Grund ist ein Replikationskonzept mit starker Konsistenz eine gute Wahl, welches wiederum einen hohen Kommunikationsbedarf hat. Eine bessere Alternative liegt in der Möglichkeit, die maximal zulässige Abweichung der Replikatbestände und damit implizit die Performanz dynamisch anpassen zu können. Sind für einen bestimmten Zug eine hohe Anzahl von Plätzen frei, so kann für eine höhere Performanz durch einen geringeren Kommunikationsbedarf die maximal zulässige Abweichung hoch gewählt werden. Mit fallender Anzahl freier Plätze steigt jedoch das Risiko von Überbuchungen und im laufenden Betrieb kann gleitend die maximal zulässige Abweichung verringert werden.

In unserem Anwendungsumfeld läßt sich eine dynamische Anpassung der maximal zulässigen Abweichung der Replikatbestände kaum motivieren. Das Replikationskonzept von TACT ist eher für Internetdienste im elektronischen Handel ausgelegt. TACT ist aktueller Forschungsgegenstand und auf Basis der veröffentlichten

Publikationen ist eine Beurteilung über Anspruch und Realisierbarkeit des skizzierten Replikationskonzepts noch nicht möglich.

3.2.5 Zusammenfassung der Schlußfolgerungen

Von den untersuchten Replikationskonzepten sind die Grundideen und Konzepte aus dem Projekt Bayou am besten für die Umsetzung unserer Anforderungen geeignet. Der inkrementelle Abgleichalgorithmus arbeitet sehr effizient und ist für das erwartete Verhältnis von Lese- zu Schreibaufträgen gut geeignet. Die Unterstützung mobiler Nutzer ist insbesondere durch das Konzept der Garantien für Klientensitzungen und der leichtgewichtigen Erzeugung von Replikerverwaltern weit fortgeschritten. Die Abgleichalgorithmen der untersuchten Replikationskonzepte von Dateisystemen erweisen sich hingegen als ungeeignet. Dies kann insbesondere durch die Anforderungen ihrer Anwendungsdomäne erklärt werden, in der eine geringere maximal zulässige Abweichung der Replikatabstände als in unserem Anwendungsgebiet gefordert wird. Dies hat einen hohen Kommunikationsbedarf zur Folge. Des Weiteren werden im Abgleichprozeß dieser Replikationskonzepte immer die durch Aufträge von Klienten geänderten Dateien und nicht deren Änderungen übermittelt. Die partielle Replikation, die von diesen Replikationskonzepten realisiert wird, ist auf Grund der engen Verknüpfung mit den Abgleichalgorithmen nicht getrennt als eigenständiges Konzept verwendbar.

Für das eigene Replikationskonzept sind damit folgende Konzepte originär zu entwickeln:

- Ein Algorithmus für die Auswahl von Abgleichpartnern für Abgleichsitzungen im Abgleichprozeß, der eine optimale Verbreitung der Änderungen der Replikatabstände für eine Menge von ungefähr zehn Replikerverwaltern realisiert.
- Eine partielle Replikation für mobile Replikerverwalter, die möglichst einfach in die Realisierung des Abgleichprozesses integrierbar ist.
- Eine Unterstützung von Bäumen als strukturierte Daten.
- Eine Konfliktbehandlung, die eine angemessene Berücksichtigung der Struktur von Bäumen erlaubt.

Es sei an dieser Stelle angemerkt, daß die Publikationen zu Replikationskonzepten mit schwacher Konsistenz nur einen Ausschnitt eines jeweiligen Gesamtkonzepts darstellen. Beispielsweise fehlt häufig eine über kurze Erklärungen hinausgehende Spezifikation der Konsistenzbeziehungen, obwohl das Konsistenzmodell einen wichtigen Teil eines Replikationskonzepts darstellt. Die Ausarbeitung eines Gesamtkonzepts wird daher zur vorrangigen Aufgabe in dieser Arbeit.

Kapitel 4

Modell

Inhalt

4.1	System	35
4.2	Konsistenz	44
4.3	Abgleich	60
4.4	Fehler	83
4.5	Mobilität	88
4.6	Nebenläufigkeit	91

In diesem Kapitel wird das entworfene Replikationskonzept beschrieben. Im ersten Abschnitt wird dem Leser ein Überblick über das Konzept aus Sicht des Systems gegeben, ohne dabei in unnötige Tiefen vorzudringen. Eine umfassende Vertiefung findet in den nachfolgenden Abschnitten statt, die detailliert alle Aspekte des Replikationskonzepts behandeln.

4.1 System

Das System besteht aus einer Anzahl von Klienten, die einen Dienst in Anspruch nehmen, der durch Replikatverwalter erbracht wird. Das Internet dient als Kommunikationsmedium. Jeder Replikatverwalter kann mit jedem anderen Replikatverwalter oder Klienten kommunizieren (Abbildung 4.1). Die Klienten erteilen Aufträge an beliebige Replikatverwalter, die diese autonom bearbeiten.

Der Dienst bezieht sich auf einen Datenbestand aus Bäumen, welche die grundlegende Datenstruktur des Diskurssystems bilden. Um die Performanz und Verfügbarkeit des Dienstes zu erhöhen, werden die Bäume als korrespondierende Replikate auf unterschiedlichen Netzknoten, den Replikativern, in Konsistenzbeziehungen als Kopien gehalten. Jeder Replikatiververwalter besitzt des weiteren die Softwarekomponenten, die zur Erbringung des Dienstes erforderlich sind.

Im Abgleichprozeß werden die Replikativbestände der einzelnen Replikatiververwalter abgeglichen. Der Abgleich zwischen stationären Replikatiververwaltern findet in unserem Replikationskonzept durch eine regelmäßig durchgeführte Anzahl von paarweisen Abgleichsitzungen statt, nach welchen zwei Replikatiververwalter den gleichen Replikativbestand bezüglich aller Schreibaufträge aufweisen, die vor einer Abgleichsitzung auf einem der beiden Replikatiververwalter bekannt waren. Schreibaufträge von Klienten, die während einer Abgleichsitzung angenommen werden, können in der laufenden Abgleichsitzung übertragen werden, müssen jedoch nicht. Abgleichsitzungen können nebenläufig zu der Bearbeitung von Aufträgen der Klienten durchgeführt werden (Abbildung 4.2). Um einen effizienten Abgleich mit einer geringen Menge zu übertragender Daten zu realisieren, werden nicht die vollständigen Replikativbestände in einer Abgleichsitzung übertragen, sondern lediglich deren Änderungen (inkrementelle Abgleichsitzungen). Dafür hat ein Replikatiververwalter neben seinem Replikativbestand, der in einer Datenbank verwaltet wird, ein Schreiblog, welches die Änderungen des Replikativbestands enthält. Diese Änderungen entstehen durch die Ausführung von schreibenden Operationen, welche die Replikatiververwalter aus den schreibenden Aufträgen von Klienten erzeugen. Die Operationen, die vom Diskurssystem unterstützt werden, sind in Tabelle 4.1 dargestellt. Weitere Operationen werden im Rahmen dieser Arbeit nicht betrachtet. Für jeden Auftrag wird genau eine Operation erzeugt, die entweder lesend oder schreibend ist (Abbildung 4.3) und neben den Nutzdaten des Auftrags um Verwaltungsdaten ergänzt ist. Syntaktisch oder semantisch inkorrekte Aufträge werden abgewiesen. Die Gruppe der schreibenden Operationen (löschen, erstellen, usw.) wird als *Schreiboperation* bezeichnet. Die Schreiboperationen werden in den Abgleichsitzungen zwischen den Abgleichpartnern ausgetauscht und sind zu diesem Zweck im Schreiblog eines jeden Replikatiververwalters gespeichert. Einem Abgleichpartner werden immer nur solche Schreiboperationen mitgeteilt, die diesem unbekannt sind. Der *Wissensstand*¹ über die dem

¹Man beachte den begrifflichen Unterschied, daß der *Wissensstand* (repräsentiert in der Form eines

Abgleichpartner bekannten Schreiboperationen wird dazu während der Abgleichsitzung ausgetauscht. Eine Schreiboperation, die in einer Abgleichsitzung empfangen wurde, wird sofort in das Schreiblog eingetragen, nicht aber unbedingt unmittelbar im Replikatabstand ausgeführt. Dies würde den Durchsatz des Dienstes erheblich mindern, wie es genauer im Kapitel 5 beschrieben ist. Einen Ausschnitt des Ablaufs einer Abgleichsitzung wird in Abbildung 4.4 dargestellt.

Jeder Replikerverwalter kann seinen Dienst autonom erbringen, so daß lediglich schwache Konsistenz realisiert werden kann. Die nebenläufig bearbeiteten Aufträge von Klienten auf unterschiedlichen Replikerverwaltern können zu Konflikten führen, weil keine sofortige systemweite *Serialisierung* der durch die Aufträge erzeugten Schreiboperationen erfolgt. Eine sofortige systemweite Serialisierung hätte eine hohe Anzahl von auszutauschenden Nachrichten zur Folge, die auch zu einer Verringerung der Verfügbarkeit in Fehlersituationen führen kann.

Daher wird die Serialisierung zu einem späteren Zeitpunkt durch einen besonders ausgezeichneten stationären Replikerverwalter, den *primären Replikerverwalter*², durchgeführt. Systemweit existiert nur ein primärer Replikerverwalter; alle anderen stationären Replikerverwalter sind *sekundär*. Eine Qualifizierung von mobilen Replikerverwaltern in primär und sekundär ist nicht sinnvoll. Der primäre Replikerverwalter serialisiert alle ihm bekannten Schreiboperationen, in dem er sie *bestätigt* und ihnen damit eine Position in der systemweiten Ausführungsreihenfolge zuweist. Die Entscheidung, ein *primary commit scheme* [Sto79] in unserem Replikationskonzept zu verwenden, hat den Vorteil, daß kein Majoritätsquorum für eine Serialisierung notwendig und ein geringer Kommunikationsaufwand vorhanden ist. Der primäre Replikerverwalter benötigt kaum mehr Ressourcen für die Rechenleistung und Kommunikation als ein sekundärer, weil die Serialisierung von Schreiboperationen leichtgewichtig ist (keine Transaktionen) und Bestätigungen in den normalen Abgleichsitzungen wie Schreiboperationen selbst übertragen werden. Dabei wird die Last für die Verbreitung der Bestätigungen, wie bei Schreiboperationen auch, durch

Vektors natürlicher Zahlen) nicht dem Wissen (die Schreiboperationen selbst) eines Replikerverwalters entspricht.

²Damit hat der primäre Replikerverwalter, dessen Bezeichnung aus Bayou [PST⁺97] übernommen wurde, in unserem Replikationskonzept eine andere Aufgabe (das Serialisieren) als in dem Primary-Copy-ROWA-Replikationskonzept, welches in Abschnitt 3.1.1 vorgestellt wurde.

einen transitiven Wissenstransfer auf alle stationären Replikativhalter verteilt. Obwohl in unserem Projekt nur ein Bedarf an ungefähr zehn stationären Replikativhaltern besteht, wäre nach Performanzmessungen im Projekt Bayou [PST⁺97] eine Unterstützung von über eintausend Replikativhaltern mit diesem Ansatz möglich. Der Nachteil eines einzigen Replikativhalters für die Bestätigung von Schreiboperationen ist, daß nach dessen Ausfall vorläufig keine Schreiboperationen bestätigt werden (bis zur manuellen Reparatur). In unserem Anwendungsumfeld ist dies jedoch akzeptabel (eine detaillierte Diskussion erfolgt in Abschnitt 4.4).

Bei der Serialisierung von Schreiboperationen ist zu beachten, daß die von unterschiedlichen Replikativhaltern bearbeiteten Klientenaufträge und die daraus erzeugten Schreiboperationen in keiner kausalen Beziehung stehen, weil eine autonome Bearbeitung ohne Kommunikation mit anderen Replikativhaltern für eine hohe Verfügbarkeit und Performanz erfolgt. Diese Schreiboperationen können somit in beliebiger Reihenfolge serialisiert werden. Schreiboperationen hingegen, die von demselben Replikativhalter erzeugt werden, müssen in der *Reihenfolge ihrer Erzeugung* serialisiert werden, damit ein (potentiell) kausaler Zusammenhang auch in der systemweiten Ausführungsreihenfolge erhalten bleibt (beispielsweise erstellt eine Schreiboperation einen Knoten, der in einer späteren Schreiboperation geändert wird). Die Ausführung einer bestätigten Schreiboperation ist *endgültig*, d.h. die Auswirkung (das Ergebnis der Ausführung) einer bestätigten Schreiboperation kann nur durch spätere bestätigte Schreiboperationen rückgängig gemacht werden. Der primäre Replikativhalter bestätigt Schreiboperationen unmittelbar nach deren Erzeugung, die dann den Status *bestätigt* haben. Auf sekundären Replikativhaltern erhalten die erzeugten Schreiboperationen den Status *unbestätigt*. Weil die unbestätigten Schreiboperationen verschiedener Replikativhalter nebenläufig erzeugt werden und ihre Position in der systemweiten Ausführungsreihenfolge zunächst unbekannt ist, ist deren Ausführung *vorläufig*. Damit ist eine Ausführung zwar für einen Klienten sichtbar, aber das Ergebnis der Ausführung in einem Replikatbestand kann sich noch ändern, wenn beispielsweise einem Replikativhalter durch eine Abgleichsitzung eine Schreiboperation bekannt wird, die gemäß der Serialisierung in seinem Replikatbestand vorher auszuführen ist. Die Schreiboperationen von sekundären Replikativhaltern werden bestätigt, wenn der primäre Replikativhalter Kenntnis über diese Schreiboperationen während einer Abgleichsitzung erhält. Die Daten einer solchen Bestätigung werden wie die Schreiboperationen selbst in Abgleichsitzungen

propagiert, so daß die systemweite Ausführungsreihenfolge allen Replikatverwaltern bekannt wird. Das in diesem Abschnitt erläuterte Prinzip der Serialisierung ist in einem Beispiel in Abbildung 4.5 visualisiert. Zwei Klienten erteilen nebenläufig zwei sekundären Replikatverwaltern Aufträge, die nach Abgleichsitzungen dem primären Replikatverwalter bekannt sind und von diesem bestätigt werden. Das Wissen über diese Bestätigungen wird in späteren Abgleichsitzungen propagiert.

Die nebenläufig erzeugten Schreiboperationen von unterschiedlichen Replikatverwaltern können Konflikte erzeugen, die in einer Konfliktbehandlung aufgelöst werden. Die Konfliktbehandlung von unbestätigten Schreiboperationen ist wie deren Ausführung vorläufig. Die Konfliktbehandlung, die für bestätigte Schreiboperationen durchgeführt wird, ist endgültig.

Zur Unterstützung mobiler Nutzer ist es erforderlich, daß für eine Offline-Arbeit auf ihrem Netzknoten sowohl die Komponente eines Klienten als auch die Komponente eines mobilen Replikatverwalters zur Verfügung steht (siehe auch Abbildung 4.1). Wir gehen in unserem Anwendungsumfeld davon aus, daß im Regelfall ein mobiler Replikatverwalter nur einen Klienten bedient, der auf dem gleichen Netzknoten läuft. Dennoch ist es möglich, daß ein mobiler Replikatverwalter auch zusätzliche Klienten bedient und beispielsweise als lokaler Server in einem *Local Area Network* (LAN) fungiert.

In unserem Replikationskonzept existieren zwei Typen von Replikatverwaltern, nämlich mobile und stationäre. Diese unterscheiden sich vor allem dadurch, daß ein stationärer Replikatverwalter eine ständige Kommunikationsverbindung zum Internet und eine große Menge an Ressourcen zur Verfügung hat. Mobile Replikatverwalter hingegen haben eine über längere Zeiträume unsichere Kommunikationsverbindung und eine beschränkte Kapazität an Massenspeicher. Daher wird einem mobilen Replikatverwalter die partielle Replikation zur Verfügung gestellt, damit nur solche Bäume repliziert werden können, die für die Erfüllung seiner Aufgabe notwendig sind. Des weiteren sind mobile Replikatverwalter nicht an den regelmäßig durchgeführten Abgleichsitzungen der stationären Replikatverwalter beteiligt, weil ein mobiler Replikatverwalter selbst entscheiden soll, wann er mit seinen beschränkten Ressourcen eine Abgleichsitzung durchführen möchte. Mobile und stationäre Replikatverwalter haben einen Kennzeichner, der systemweit eindeutig ist.

Operation ^a	Bedeutung
LeseKnoten(knotenID, attributNamen)	Liest die Attribute mit Namen attributNamen des Knotens mit dem Kennzeichner knotenID.
ErstelleKnoten(knotenID)	Erstellt einen neuen Knoten mit einem Vater, dessen Kennzeichner knotenID ist. Soll ein neuer Baum im Wald erstellt werden, ist die knotenID NIL.
LöscheUnterbaum(knotenID)	Löscht den Unterbaum mit dem Kennzeichner knotenID.
ÄndereKnoten(knotenID, attributNamen & zuweisungen)	Ändert die Attribute mit Namen attributNamen eines Knotens mit dem Kennzeichner knotenID durch die Wertzuweisungen zuweisungen.
VerschiebeUnterbaum(knotenID ₁ , knotenID ₂)	Verschiebt den Unterbaum mit dem Kennzeichner knotenID ₁ unter die Zielposition knotenID ₂ .

^aIm folgenden werden die Operationen dieser Tabelle verkürzt dargestellt. So wird aus ErstelleKnoten(...) erstelle(...), aus LöscheUnterbaum(...) lösche(...) usw.

Tabelle 4.1: Operationen auf Bäumen in Replikatabständen

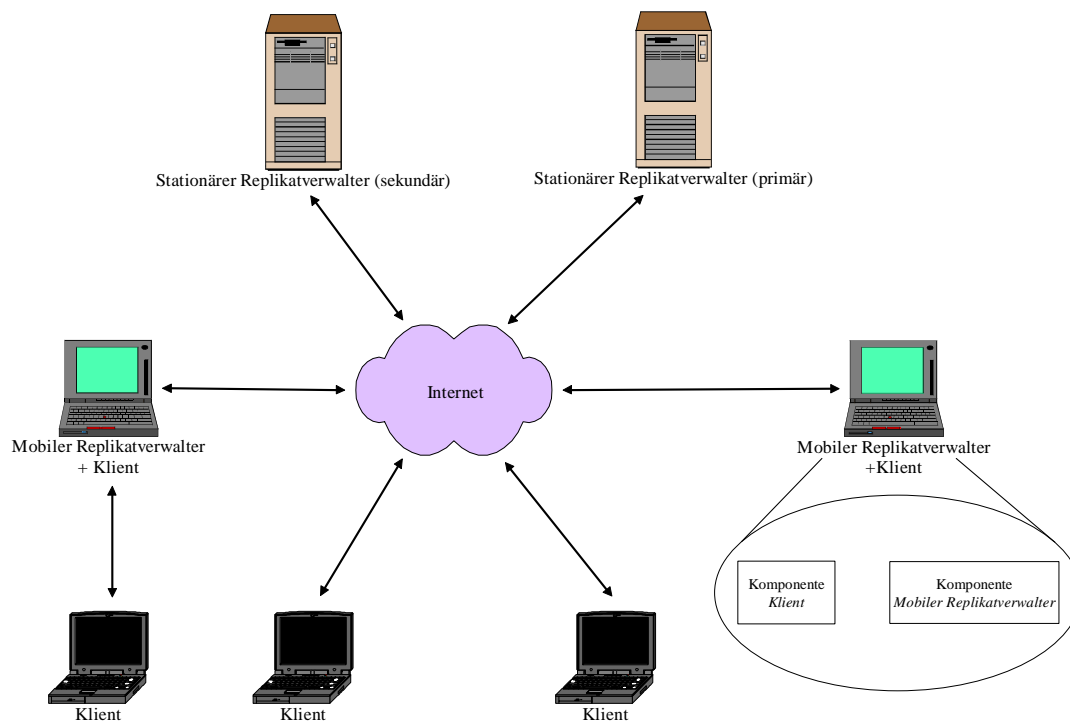


Abbildung 4.1: Systemmodell

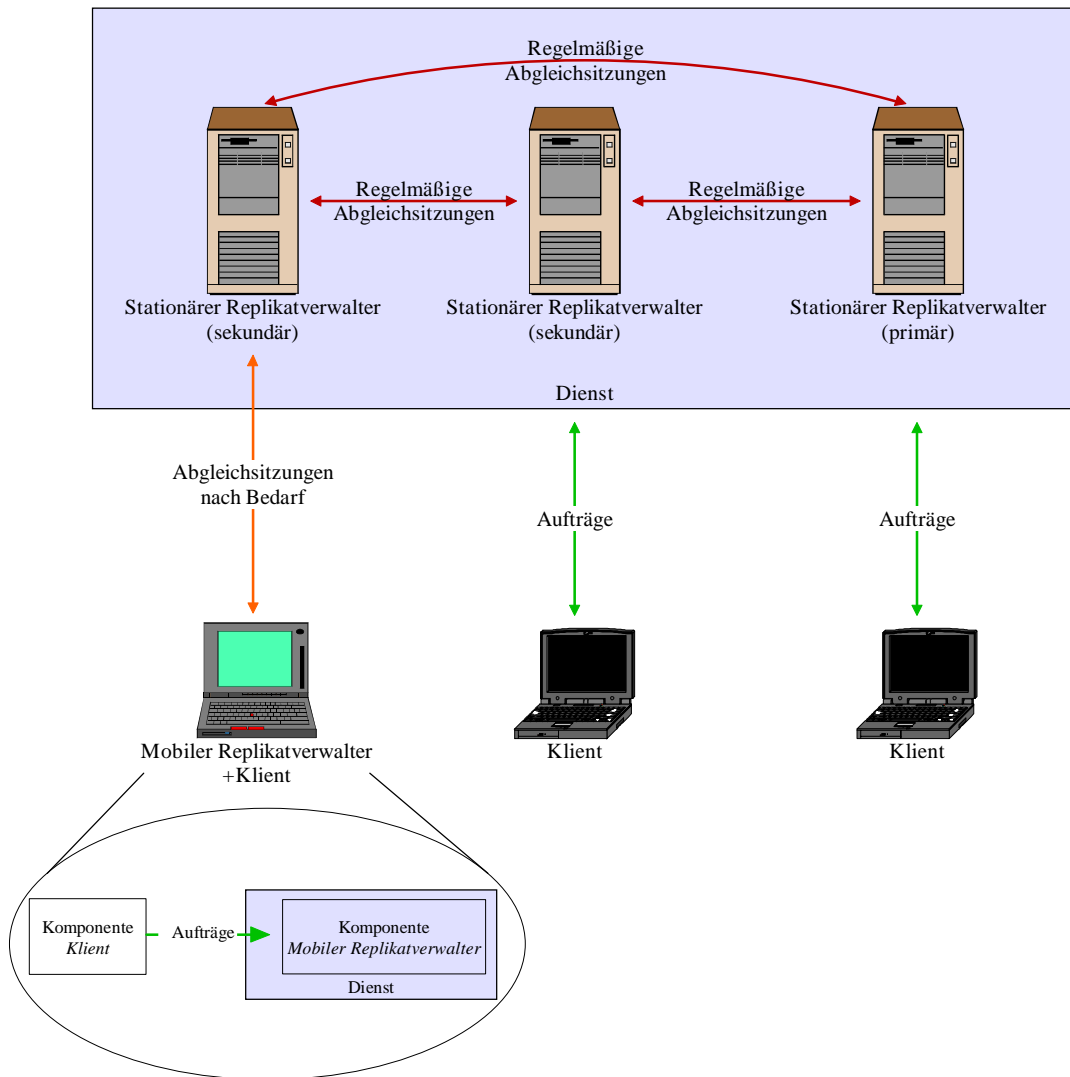


Abbildung 4.2: Durchführung von Abgleichsitzungen bei nebenläufiger Bearbeitung von Klientenaufträgen

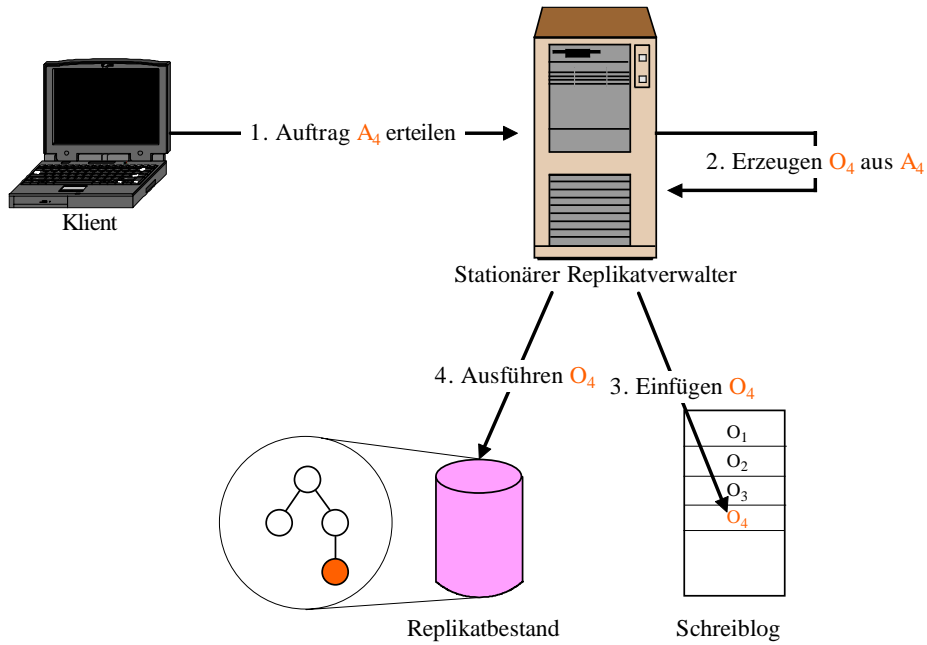


Abbildung 4.3: Bearbeitung von Aufträgen

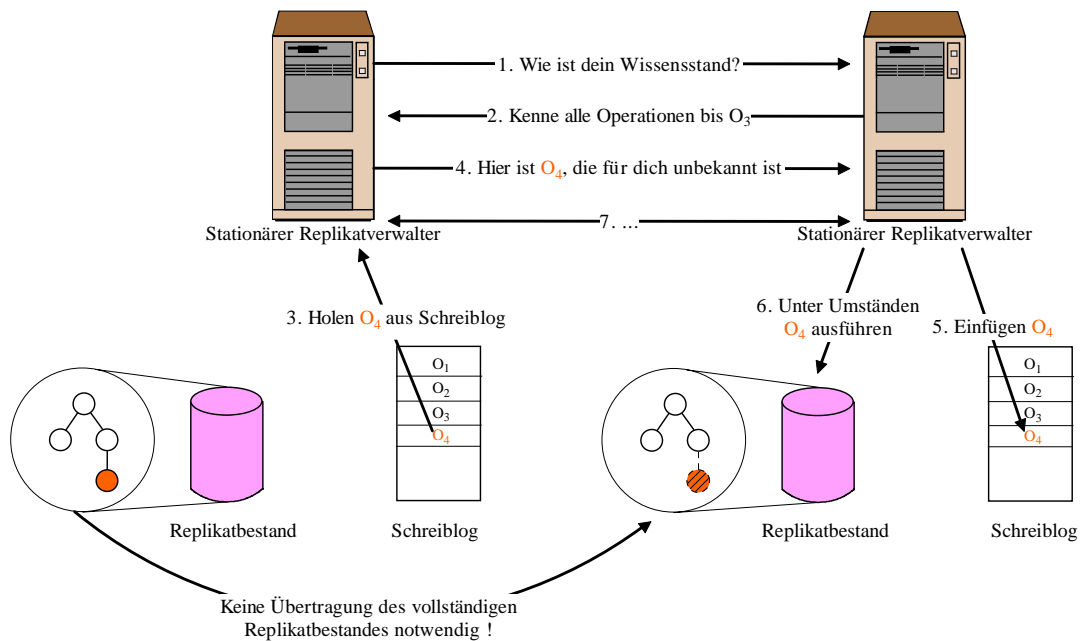


Abbildung 4.4: Ausschnitt einer Abgleichsitzung

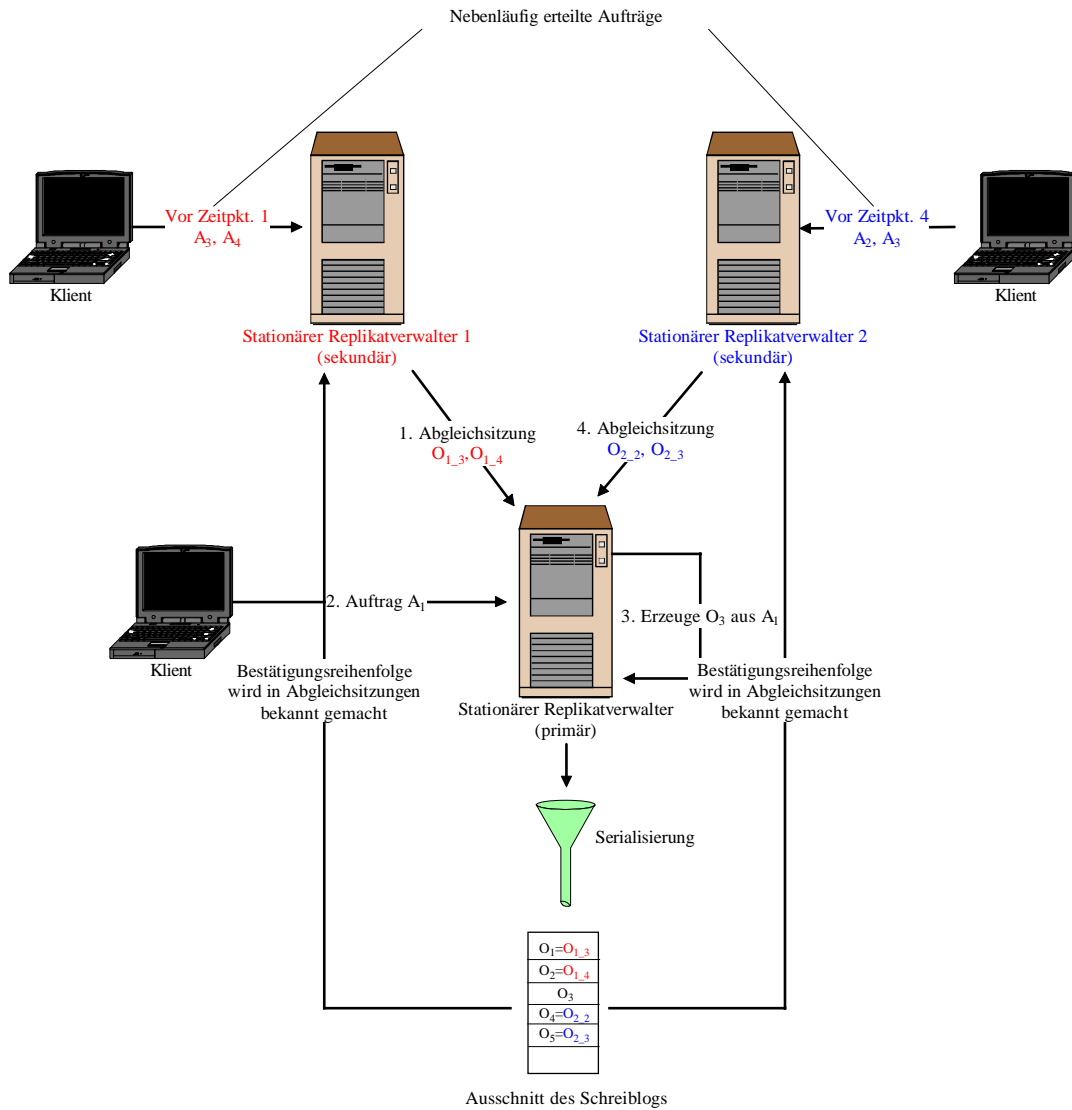


Abbildung 4.5: Serialisierung von Schreiboperationen

4.2 Konsistenz

Die Basis eines Replikationskonzepts besteht aus dem Konsistenzmodell, welches die Konsistenzbeziehungen zwischen den verteilten Replikatbeständen, ihren Veränderungen und den nebenläufig erteilten Änderungsaufträgen spezifiziert. Die Konsistenzbeziehungen des Systems gelten zu jedem Zeitpunkt, also auch in Fehlersituationen. In diesem Abschnitt werden die Konsistenzbeziehungen anschaulich erläutert und formal spezifiziert. Anschließend werden Garantien für Klientensitzungen vorgestellt, die einem Klienten eine konsistentere Sicht auf das System bieten. Weil in einem Replikationskonzept mit schwacher Konsistenz Konflikte auftreten, ist eine Konfliktbehandlung erforderlich, die ebenfalls in diesem Abschnitt erläutert wird. Im Sinne einer fließenden Sprache wird im folgenden der Begriff Operation als Synonym für Schreiboperation verwendet.

Konsistenzbeziehungen

Die Konsistenzbeziehungen werden auf der Grundlage der Prädikatenlogik 1. Stufe spezifiziert (ein zu gewissem Grad ähnlicher Ansatz ist in [CGMW96] zu finden). Dazu führen wir eine *first-order language* (FOL) mit den unten angegebenen Prädikatsymbolen ein, die das Geschehen von *Ereignissen* oder das Bestehen von *Bedingungen* (Zuständen) ausdrückt. Die Prädikatsymbole bezeichnen Prädikate auf folgenden Individuenmengen:

- Menge der Replikatverwalter \mathcal{R} , welche Replikatbestände verwalten, Aufträge annehmen und bearbeiten sowie Operationen erzeugen, ausführen, verbreiten und bestätigen.
- Menge der Schreibaufträge \mathcal{A} , die ein Klient einem Replikatverwalter erteilen kann.
- Menge der Schreiboperationen \mathcal{O} , die von Replikatverwaltern aus Aufträgen erzeugt werden können.
- Menge der Zeitpunkte \mathcal{T} . Die Zeitpunkte $t \in \mathcal{T}$ kann man, wie üblich, durch reelle Zahlen $t \in \mathbb{R}$ modellieren. Für den vorliegenden Beschreibungszweck werden jedoch lediglich drei Beziehungen zwischen den Zeitpunkten verwendet, um das Geschehen von Ereignissen oder das Bestehen von Bedingungen zueinander in eine der Beziehungen $<$ („früher als“), $=$ („zugleich mit“) und $>$ („später als“) zu setzen. Es

reicht daher, die Zeitmenge \mathcal{T} durch die natürlichen Zahlen N zu modellieren. Die Zeitmenge \mathcal{T} wird jedoch als endlich betrachtet, um eine klare Interpretation von Aussagen wie „es existiert ein Zeitpunkt t , zu welchem eine Operation allen Replikativverwaltern bekannt ist“ zu haben und damit die Frage explizit auszuklammern, ob das „irgendwann“ auch einem Zeitpunkt $t = \infty$ entsprechen kann.

Zeitpunkte, die von demselben Replikativverwalter einem Ereignis oder einer Bedingung zugeordnet werden (*time stamps*), können in trivialer Weise in Beziehung gesetzt werden (eine entsprechende Taktung des Weiterzählens vorausgesetzt). In einem partiell synchronen verteilten System ist ein Vergleich von Zeitpunkten, die von verschiedenen Replikativverwaltern zugeordnet wurden, nur zulässig, wenn die verknüpften Ereignisse und Bedingungen über das Senden und Empfangen von Nachrichten in einem kausalen Zusammenhang stehen (*happened-before* Beziehung [Lam78]). Dabei wird die triviale Tatsache zugrunde gelegt, daß eine Nachricht später empfangen wird als sie abgesendet wurde. Ein Vergleich von Zeitpunkten, die von verschiedenen Replikativverwaltern zugeordnet wurden, ist jedoch in unseren spezifizierten Konsistenzbeziehungen nicht erforderlich.

Quantoren auf der Menge \mathcal{T} lassen sich auch ohne Rückgriff auf die Sonderkonstruktion von „temporalen Quantoren“ verwenden, wenn man die Entwicklung des Systems gewissermaßen in der Rückschau betrachtet, um die Vorstellung einer sich ständig erweiternden Individuenmenge zu vermeiden. In der (gedachten) Aufzeichnung des Systemgeschehens sind das Geschehen von Ereignissen und das Bestehen von Bedingungen mit Zeitangaben versehen, die obigen Annahmen entsprechen. Bezüglich dieser Zeitangaben gelten die folgenden formulierten Konsistenzbeziehungen:

$\forall t$ heißt dann „für jeden Zeitpunkt t in der Geschichte des Systems gilt: ...“,

$\exists t$ heißt dann „es gibt einen Zeitpunkt t in der Geschichte des Systems, für den gilt: ...“.

Von der (endlichen) Individuenmenge \mathcal{R} wird angenommen, daß jedes ihrer Individuen (Replikativverwalter) ab irgendeinem Zeitpunkt der Systemgeschichte zum

Netz der Rechnernetzknoten gehört und dieses nicht mehr, außer vorübergehend, verläßt. Vorübergehendes Verlassen bedeutet, daß ein Replikatverwalter in einem Zeitraum nicht verfügbar ist, weil beispielsweise Fail-Stop-Fehler auftreten oder ein mobiler Replikatverwalter offline ist. Durch diese Annahme werden Fälle ausgeschlossen, in denen ein Replikatverwalter ab einem Zeitpunkt nicht (nie) wieder verfügbar wird, wodurch (trivialer Weise) Garantien der Art, daß eine Operation irgendwann jedem Replikatverwalter bekannt ist, verletzt würden.

Um eine lesefreundliche Notation zu verwenden, werden folgende (übliche) Vereinbarungen getroffen: Werden in der Spezifikation n -stellige Prädikate verwendet, um eine Aussage zu formulieren, die sich auf einen Zeitpunkt t bezieht, so handelt es sich eigentlich um ein $(n + 1)$ -stelliges Prädikat, auch wenn das Zeitargument – wie in dieser Arbeit – durch eine besondere Notation [aus $t \in \mathcal{T}$ und $p(O, R, t)$ wird $p_t(O, R)$] hervorgehoben wird. Die Argumente der Prädikate werden einmal eingeführt, und es wird darauf verzichtet, diese vor jedem Ausdruck zu wiederholen [„sei $O \in \mathcal{O}, \dots, t \in \mathcal{T}$, dann gelte ...“]. Vereinbart wird $A \in \mathcal{A}; O, O_1, O_2 \in \mathcal{O}; R, R_1, R_2, R', R_p \in \mathcal{R}$ (wobei das Individuum R_p den primären Replikatverwalter bezeichnet); $t, t_1, t_2, t' \in \mathcal{T}$. Des weiteren wird auf die Darstellung der äußeren Allquantoren einer Aussage ebenso verzichtet [aus $\forall R (\forall t (p_t(R) \implies \dots))$ wird $p_t(R) \implies \dots$] wie auf die Darstellung der Klammerungen von Quantoren mit gleichem Bezug [aus $\dots \forall R (\exists t (p_t(R))) \dots$ wird $\dots \forall R \exists t (p_t(R)) \dots$]. Vergleiche von Zeitpunkten werden zur besseren Lesbarkeit verkürzt dargestellt [aus $\forall t_1 \exists t_2 (p_{t_1}(R) \dots q_{t_2}(R) \wedge (t_1 < t_2))$ wird $\exists t_2 > t_1 (p_{t_1}(R) \dots q_{t_2}(R))$].

Folgende Prädikate bringen das Geschehen eines Ereignisses zum Zeitpunkt t zum Ausdruck:

empfangenAuftrag $_t(A, R)$: Der Replikatverwalter R empfängt den Auftrag A eines Klienten zum Zeitpunkt t .

erzeugenOp $_t(O, R)$: Der Replikatverwalter R erzeugt (aus einem empfangenen Auftrag) die Operation O zum Zeitpunkt t .

bestätigenOp $_t(O, R_p)$: Der primäre Replikatverwalter R_p bestätigt die Operation O zum Zeitpunkt t . Die Bestätigung einer Operation legt dabei lediglich die systemweite Reihenfolge der Ausführung fest und beinhaltet keine Aussage, ob diese Operation mit anderen in Konflikt steht.

ausführenOp $_t(O, R)$: Der Replikatverwalter R führt die Operation O zum Zeitpunkt t aus.

$zurücknehmenOp_t(O, R)$: Der Replikativverwalter R nimmt die Operation O zum Zeitpunkt t zurück. Es ist nur die Rücknahme der vorläufigen Ausführung einer Operation möglich.

Folgende Prädikate bringen das Bestehen einer Bedingung zum Zeitpunkt t zum Ausdruck:

$bearbeitbar_t(A, R)$: Der Auftrag A kann von einem Replikativverwalter R zum Zeitpunkt t bearbeitet werden, wenn er syntaktisch und semantisch korrekt ist.

$unbestätigt_t(O, R)$: Die Operation O ist zum Zeitpunkt t auf dem Replikativverwalter R als unbestätigt bekannt.

$bestätigt_t(O, R)$: Die Operation O ist zum Zeitpunkt t auf dem Replikativverwalter R als bestätigt bekannt.

$bekannt_t(O, R) : \iff$

$unbestätigt_t(O, R) \wedge \neg bestätigt_t(O, R) \vee bestätigt_t(O, R) \wedge \neg unbestätigt_t(O, R)$

Es gilt der Zusammenhang, daß die zum Zeitpunkt t bekannte Operation O auf einem Replikativverwalter R entweder als unbestätigt und nicht bestätigt oder umgekehrt bekannt ist. Weil eine Operation einem Replikativverwalter auch unbekannt sein kann, gilt nicht: $\neg bestätigt_t(O, R) \iff unbestätigt_t(O, R)$.

e -ausgeführt $_t(O, R) : \iff$

$\exists t_1 < t (ausführenOp_{t_1}(O, R) \wedge bestätigt_{t_1}(O, R))$

Auf dem Replikativverwalter R war die Operation O zum Zeitpunkt t_1 der Ausführung als bestätigt bekannt und damit ist die Operation zum Zeitpunkt t endgültig ausgeführt (siehe auch Konsistenzbeziehung 4.10).

v -ausgeführt $_t(O, R) : \iff$

$\exists t_1 < t (ausführenOp_{t_1}(O, R) \wedge unbestätigt_{t_1}(O, R) \wedge \forall t_2 (t_1 < t_2 < t \implies \neg zurücknehmenOp_{t_2}(O, R)))$

Auf dem Replikativverwalter R wurde die Operation O zum Zeitpunkt t_1 ausgeführt und bis zum Zeitpunkt t nicht zurückgenommen, so daß die Operation auf R zum Zeitpunkt t vorläufig ausgeführt ist.

Die folgenden Konsistenzbeziehungen wurden zur besseren Übersichtlichkeit in drei Themengruppen unterteilt. Zu jeder formalen Konsistenzbeziehung wird eine kurze textuelle Erläuterung gegeben.

Erzeugung, Status und Bestätigung einer Operation

„Die Annahme eines Klientenauftrags und damit die Erzeugung einer Operation erfolgt unter bestimmten Bedingungen“, d.h.: Sobald ein Auftrag A auf einem Replikatverwalter R zu einem Zeitpunkt t empfangen wird und dieser Auftrag bearbeitbar ist, wird daraus eine Operation O erzeugt. Eine Abbildung $f: \mathcal{A} \times \mathcal{R} \times \mathcal{T} \rightarrow \mathcal{O}$ stellt dabei den Zusammenhang zwischen der Erzeugung einer Operation und dem zugehörigen Auftrag her (ein einfaches Kopieren der Daten des Auftrags in die Datenstruktur der zu erzeugenden Operation mit der Ergänzung um einen Annahmestempel aus einer Annahmenummer und dem Kennzeichner des erzeugenden Replikatverwalters). Ein Auftrag kann nur bearbeitet werden, wenn er syntaktisch und semantisch korrekt ist. Dazu gehört insbesondere, daß ein Replikatverwalter nur dann einen Auftrag zur Bearbeitung annimmt, wenn die Bezugsbäume des Auftrags (beispielsweise ein zu löschender Unterbaum) auf diesem zum Zeitpunkt t vorhanden sind.

$$\text{empfangenAuftrag}_t(A, R) \wedge \text{bearbeitbar}_t(A, R) \implies \text{erzeugenOp}_t(O, R) \quad (4.1)$$

„Eine erzeugte Operation ist zunächst unbestätigt“, d.h.: Eine auf einem Replikatverwalter R erzeugte Operation O ist auf diesem zum Zeitpunkt t der Erzeugung als unbestätigt bekannt. Dies gilt auch für den primären Replikatverwalter, der jedoch vor Erzeugung der nächsten Operation eine Bestätigung durchführt.

$$\text{erzeugenOp}_t(O, R) \implies \text{unbestätigt}_t(O, R) \quad (4.2)$$

„Jede unbestätigte Operation ist irgendwann bestätigt“, d.h.: Eine auf einem Replikatverwalter R zum Zeitpunkt t_1 als unbestätigt bekannte Operation O wird auf diesem zu einem späteren Zeitpunkt t_2 als bestätigt bekannt sein. Die Bestätigung einer Operation legt die Position in der systemweiten Ausführungsreihenfolge (Bestätigungsreihenfolge) fest. Weil die Ausführung einer unbestätigten Operation nur vorläufig ist, muß einem Nutzer garantiert sein, daß eine Bestätigung erfolgt und damit eine endgültige Ausführung möglich ist.

$$\text{unbestätigt}_{t_1}(O, R) \implies \exists t_2 > t_1(\text{bestätigt}_{t_2}(O, R)) \quad (4.3)$$

„Die Bestätigungsreihenfolge von Operationen entspricht der Erzeugungsreihenfolge“, d.h.: Wird von dem primären Replikativverwalter R_p zum Zeitpunkt t eine Operation O_2 bestätigt, die zum Zeitpunkt t_2 von dem Replikativverwalter R' erzeugt wurde, dann sind alle Operationen O_1 , die vor O_2 von R' erzeugt wurden, von R_p bereits bestätigt (also zum Zeitpunkt t auf R_p als bestätigt bekannt). Die Erzeugungsreihenfolge von Operationen spiegelt potentielle kausale Beziehungen zwischen den einzelnen Operationen (transitiv) wider (beispielsweise die Antwort auf einen Diskussionsbeitrag). Diese Reihenfolge muß daher auch bei der Bestätigung erhalten bleiben. Weil zwischen den Operationen, die von unterschiedlichen Replikativverwaltern erzeugt wurden, keine kausale Beziehung besteht (es gibt keine Kommunikation der Replikativverwalter über solche Erzeugungsvorgänge), können diese Operationen in beliebiger Reihenfolge bestätigt (serialisiert) werden. Weil auch die Übertragung von unbestätigten Operationen in Erzeugungsreihenfolge erfolgt (Konsistenzbeziehung 4.7), kann die Einhaltung der Bestätigungsreihenfolge sehr einfach realisiert werden, indem der primäre Replikativverwalter die Operationen, die er in einer Abgleichsitzung erhält, umgehend in der Empfangsreihenfolge bestätigt.

$$\begin{aligned} & \text{bestätigenOp}_t(O_2, R_p) \wedge \text{erzeugenOp}_{t_2}(O_2, R') \\ & \implies \forall O_1 (\exists t_1 < t_2 (\text{erzeugenOp}_{t_1}(O_1, R')) \implies \text{bestätigt}_t(O_1, R_p)) \end{aligned} \quad (4.4)$$

„Ist eine Operation auf einem Replikativverwalter einmal als bestätigt bekannt, so darf sie ihren Status nicht zu unbestätigt wechseln“, d.h.: Ist eine Operation O zu einem Zeitpunkt t_1 auf einem Replikativverwalter R als bestätigt bekannt, so gibt es keinen späteren Zeitpunkt t_2 , zu welchem sie auf R als unbestätigt bekannt ist. Die Ausführung einer bestätigten Operation kann nur dann endgültig sein, wenn diese ihren Status nicht zu unbestätigt wechseln und damit als unbestätigte Operation zurückgenommen werden kann.

$$\text{bestätigt}_{t_1}(O, R) \implies \forall t_2 > t_1 (\text{bestätigt}_{t_2}(O, R)) \quad (4.5)$$

Abgleichprozeß

„Jede Operation ist irgendwann allen Replikativverwaltern bekannt“, d.h.: Ist eine Operation O zum Zeitpunkt t auf einem Replikativverwalter R bekannt, so gibt es auf jedem Replikativverwalter R' einen Zeitpunkt t' , zu welchem diese Operation auf R'

bekannt ist. Damit wird garantiert, daß eine Wissensverbreitung stattfindet und jeder Replikatverwalter Kenntnis von den durch Klientenaufträgen erzeugten Operationen erhält.

$$\text{bekannt}_t(O, R) \implies \forall R' \exists t' (\text{bekannt}_{t'}(O, R')) \quad (4.6)$$

„Unbestätigte Operationen werden einem Replikatverwalter in der Erzeugungsreihenfolge bekannt gemacht“, d.h.: Ist auf einem Replikatverwalter R zum Zeitpunkt t eine Operation O_2 als unbestätigt bekannt, die zum Zeitpunkt t_2 von einem Replikatverwalter R' erzeugt wurde, dann sind alle Operationen O_1 , die vor O_2 von R' erzeugt wurden, ebenfalls auf R zum Zeitpunkt t als unbestätigt bekannt. Die potentiell kausalen Beziehungen der Operationen sind auch bei der Übertragung zu berücksichtigen. Dadurch kann in Fehlerfällen bei einem unplanmäßigen Abbruch einer Abgleichsitzung auf Recovery-Maßnahmen verzichtet werden, weil der Replikatbestand die korrekten kausalen Beziehungen der Operationen widerspiegelt. Des weiteren wird durch die geordnete Übertragung sichergestellt, daß eine effiziente Repräsentation des Wissens möglich ist.

$$\begin{aligned} & \text{unbestätigt}_t(O_2, R) \wedge \text{erzeugenOp}_{t_2}(O_2, R') \\ & \implies \forall O_1 (\exists t_1 < t_2 (\text{erzeugenOp}_{t_1}(O_1, R')) \implies \text{unbestätigt}_t(O_1, R)) \end{aligned} \quad (4.7)$$

„Bestätigte Operationen bzw. die Bestätigungen von unbestätigten Schreiboperationen werden einem Replikatverwalter in der Bestätigungsreihenfolge bekannt gemacht“, d.h.: Ist auf einem Replikatverwalter R zum Zeitpunkt t eine Operation O_2 als bestätigt bekannt, die zum Zeitpunkt t_2 von dem primären Replikatverwalter R_p erzeugt wurde, dann sind alle Operationen O_1 , die vor O_2 von R_p bestätigt wurden, ebenfalls auf R zum Zeitpunkt t als bestätigt bekannt. Die Begründung ist analog zur Konsistenzbeziehung 4.7.

$$\begin{aligned} & \text{bestätigt}_t(O_2, R) \wedge \text{bestätigenOp}_{t_2}(O_2, R_p) \\ & \implies \forall O_1 (\exists t_1 < t_2 (\text{bestätigenOp}_{t_1}(O_1, R_p)) \implies \text{bestätigt}_t(O_1, R)) \end{aligned} \quad (4.8)$$

Ausführung im Replikatbestand eines Replikatverwalters

„Jede Operation ist irgendwann endgültig ausgeführt“, d.h.: Ist auf einem Replikatverwalter R eine Operation O zum Zeitpunkt t bekannt, dann existiert ein Zeitpunkt

t' , zu welchem diese Operation auf R endgültig ausgeführt ist. Der Zeitpunkt für die Ausführung einer Operation wird einem Replikativverwalter überlassen, damit dieser für hohe Performanz eine gruppierte Ausführung von Operationen durchführen kann. Einem Nutzer muß jedoch gewährleistet sein, daß eine aus seinem Auftrag erzeugte Operation endgültig ausgeführt wird.

$$\text{bekannt}_t(O, R) \implies \exists t'(e\text{-ausgef\u00fchrt}_{t'}(O, R)) \quad (4.9)$$

„Nur vorläufig ausgeführte Operationen können zurückgenommen werden“ und somit „Wurde eine bestätigte Operation ausgeführt, so ist deren Ausführung endgültig“, d.h.: Wird eine Operation O auf einem Replikativverwalter R zum Zeitpunkt t zurückgenommen, so ist diese zum Zeitpunkt t vorläufig im Replikativbestand von R ausgeführt. Die Rücknahme der Ausführung von unbestätigten Schreiboperationen ist notwendig, weil deren Positionen in der Bestätigungsreihenfolge, also in der systemweiten Ausführungsreihenfolge, unbekannt sind. Wird die Bestätigung einer unbestätigten Operation bekannt, so wird deren Ausführung zurückgenommen und diese Operation, die nun als bestätigt bekannt ist, an der Position der Bestätigungsreihenfolge, die abweichend von der Position der vorherigen Reihenfolge der Ausführung sein kann, endgültig ausgeführt.

$$\text{zur\u00fccknehmenOp}_t(O, R) \implies v\text{-ausgef\u00fchrt}_t(O, R) \quad (4.10)$$

„Eine bestätigte Operation wird vor allen unbestätigten Operationen ausgeführt“, d.h.: Wird eine als bestätigt bekannte Operation O zum Zeitpunkt t auf einem Replikativverwalter R ausgeführt, so gibt es auf R zum gleichen Zeitpunkt keine vorläufig ausgeführten Operationen O' . Das Ergebnis der Ausführung einer bestätigten Operation soll endgültig sein. Damit die bestätigten Operationen nicht von einer Änderung der Ausführungsreihenfolge von unbestätigten Operationen berührt werden, muß deren vorläufige Ausführung vor der endgültigen Ausführung einer bestätigten Operation zurückgenommen werden.

$$\text{ausf\u00fchrenOp}_t(O, R) \wedge \text{best\u00e4tigt}_t(O, R) \implies \neg \exists O' (v\text{-ausgef\u00fchrt}_t(O', R)) \quad (4.11)$$

„Bestätigte Operationen werden in der Bestätigungsreihenfolge ausgeführt“, d.h.: Wird auf einem Replikerverwalter R zum Zeitpunkt t eine bestätigte Operation O_2 ausgeführt, die zum Zeitpunkt t_2 von dem primären Replikerverwalter R_p bestätigt wurde, dann sind alle Operationen O_1 , die vor O_2 von R_p bestätigt wurden, bereits auf R endgültig ausgeführt. Diese Konsistenzbeziehung ist notwendig, um die Ausführung in der Bestätigungsreihenfolge zu sichern.

$$\begin{aligned} & \text{ausführenOp}_t(O_2, R) \wedge \text{bestätigt}_t(O_2, R) \wedge \text{bestätigenOp}_{t_2}(O_2, R_p) \\ & \implies \forall O_1(\exists t_1 < t_2(\text{bestätigenOp}_{t_1}(O_1, R_p)) \implies e\text{-ausgeführt}_t(O_1, R)) \end{aligned} \quad (4.12)$$

„Unbestätigte Operationen werden in Erzeugungsreihenfolge ausgeführt“, d.h.: Wird auf einem Replikerverwalter R zum Zeitpunkt t eine unbestätigte Operation O_2 ausgeführt, die zum Zeitpunkt t_2 von dem Replikerverwalter R' erzeugt wurde, dann sind alle Operationen O_1 , die vor O_2 von R' erzeugt wurden, bereits auf R vorläufig ausgeführt. Diese Konsistenzbeziehung ist notwendig, um den potentiell kausalen Zusammenhang der Operationen, die von demselben Replikerverwalter erzeugt wurden, bei der Ausführung zu erhalten.

$$\begin{aligned} & \text{ausführenOp}_t(O_2, R) \wedge \text{unbestätigt}_t(O_2, R) \wedge \text{erzeugenOp}_{t_2}(O_2, R') \\ & \implies \forall O_1(\exists t_1 < t_2(\text{erzeugenOp}_{t_1}(O_1, R')) \implies v\text{-ausgeführt}_t(O_1, R)) \end{aligned} \quad (4.13)$$

Sitzungsgarantien

Neben der Konsistenz des Systems, spezifiziert durch die Konsistenzbeziehungen, kann ein Klient Anforderungen an eine zusätzliche Konsistenz bezüglich seiner *Sicht* haben, die abhängig von den Replikerverwaltern ist, welche die Aufträge des Klienten bearbeiten. Insbesondere mobile Nutzer ohne einen eigenen mobilen Replikerverwalter tendieren in Abhängigkeit von ihrer Position dazu, verschiedene Replikerverwalter für Aufträge zu verwenden. Des weiteren kann ein Front-End-Verwalter eines Klienten unterschiedliche Replikerverwalter in Abhängigkeit der Kommunikationssituation und der Lastverteilung im System ansprechen. Ohne das Konzept der Garantien für Klientensitzungen (nicht zu verwechseln mit Abgleichsitzungen der Replikerverwalter) würde ein Anwender mit Inkonsistenzen³ bezüglich der Sicht seiner eigenen bisherigen Aufträge während einer Sitzung konfrontiert, die durch

³Aus Sicht des Anwenders, nicht des Systems.

abweichende Replikalbestände der einzelnen Replikalverwalter verursacht werden. Falls ein Nutzer beispielsweise einen Auftrag zum Erstellen eines Knotens erteilt und anschließend dieser neue Knoten von einem anderen Replikalverwalter gelesen werden soll, der aber von diesem Schreibauftrag noch keine Kenntnis erlangt hat, so scheint dem Nutzer sein Auftrag zu fehlen. Um solche Effekte auszuschließen, werden vom System Garantien für Klientensitzungen (kurz Sitzungsgarantien) übernommen, die wie folgt aussehen:

- *Read Your Writes*. Diese Garantie versichert, daß die Auswirkungen der bisherigen Schreiboperationen einer Sitzung bei jedem Auftrag zum Lesen der gleichen Sitzung berücksichtigt werden. Der Sinn dieser Garantie soll an einem Beispiel verdeutlicht werden: Ein Nutzer erstellt während einer Bahnfahrt einen Beitrag. Während der Fahrt wird der vom Klienten verwendete Replikalverwalter gewechselt und der Nutzer muß feststellen, daß beim automatischen Einlesen der Beiträge sein neuer Beitrag verschwunden ist. Dies wurde dadurch möglich, daß der zuletzt ausgewählte Replikalverwalter noch keine Kenntnis von der Erstelle-Operation des Auftrags hatte. Durch die angesprochene Garantie wird die Menge der verfügbaren Replikalverwalter für Leseaufträge eines Klienten so eingeschränkt, daß Leseaufträge nur solchen Replikalverwaltern erteilt werden, die jede durch die Sitzung erzeugte Schreiboperation kennen.
- *Monotonic Reads*. Aufträge zum Lesen werden nur solchen Replikalverwaltern erteilt, die jene Schreiboperationen kennen, deren Auswirkungen bei vorherigen Aufträgen zum Lesen wahrgenommen wurden. Damit wird erreicht, daß ein Nutzer beim Wechseln zu einem anderen Replikalverwalter nicht plötzlich weniger Auswirkungen von Schreiboperationen sieht als er vorher gesehen hat.
- *Monotonic Writes*. Ein Auftrag zum Schreiben darf nur solchen Replikalverwaltern erteilt werden, welche jene Schreiboperationen kennen, deren Auswirkungen in der Sitzung wahrgenommen wurden. Damit bleibt die Kausalität zwischen einer durch den Nutzer initiierten Schreiboperation und den in der Sitzung bis dahin bekannten Schreiboperationen erhalten.

Eine Sitzungsgarantie gewährleistet, daß entweder die Bearbeitung eines Auftrags gemäß der Garantie erfolgt oder angezeigt wird, daß die Bearbeitung des Auftrags nur ohne die Garantie erfolgen kann. Damit wird vermieden, daß eine Einschränkung der Menge verwendbarer Replikerverwalter zur Realisierung der Sitzungsgarantien zu einer geringeren Verfügbarkeit führen könnte. Die Implementierung dieser Sitzungsgarantien erfolgt vollständig auf der Klientenseite in einem *Session Manager*. Dies hat den Vorteil, daß Replikerverwalter keine Daten über den Sitzungsverlauf mit Klienten verwalten müssen. Weil das Konzept der Sitzungsgarantien von Bayou ohne Änderungen für unser Replikationskonzept geeignet ist, möchten wir in dieser Arbeit auf deren formale Darstellung und Realisierung verzichten. Wir verweisen dazu auf [TDP⁺94].

Konfliktbehandlung

Um das Konsistenzmodell zu vervollständigen, müssen neben den Konsistenzbeziehungen und Sitzungsgarantien auch Konflikte in das Modell mit einbezogen werden. In Replikationskonzepten mit schwacher Konsistenz wird die Erhöhung von Performanz und Verfügbarkeit mit dem Verlust der Ein-Exemplar-Serialisierbarkeit erkaufte. Daraus folgt, daß Konflikte zwischen Operationen möglich und mit steigender Anzahl von nebenläufigen Operationen auf verschiedenen Replikerverwaltern sogar wahrscheinlich sind. In der Anwendungsdomäne unseres Diskurssystems ist es akzeptabel, daß Konflikte auftreten können, wenn diese auf jedem Replikerverwalter in gleicher deterministischer Weise aufgelöst werden. Eine intelligente Behandlung von Konflikten erfordert im allgemeinen eine Berücksichtigung des Anwendungskontexts.

In unserem Replikationskonzept wird zwischen bestätigten und unbestätigten Operationen unterschieden. Die Ausführung einer bestätigten Operation ist endgültig und eine Konfliktbereinigung wurde gegebenenfalls durchgeführt. Eine bestätigte Operation wird niemals wieder zurückgenommen und mit einem anderen Ergebnis neu ausgeführt. Die Ausführung einer unbestätigten Operation hingegen ist stets vorläufig. Dieser Unterschied sollte einem Nutzer angezeigt werden, damit er ein verlässliches Bild von dem Zustand der Daten des Systems erhält. Beispielsweise können Knoten, deren Attribute durch unbestätigte Operationen geändert worden sind, in

einer besonderen Farbe dargestellt werden. Damit erkennt ein Nutzer zu jedem Zeitpunkt, ob die Informationen eines Knotens als vorläufig oder endgültig einzustufen sind.

In den untersuchten Replikationskonzepten liegt ein Konflikt vor, wenn die Ergebnisse der Ausführung von zwei oder mehr Schreiboperationen unvereinbar sind. Dies ist in Replikationskonzepten, die Datenelemente ohne Beziehungen betrachten, völlig ausreichend, weil Konflikte potentiell nur auf einem *einzigem* Datenelement pro Operation⁴ auftreten können. Dies gilt jedoch nicht bei Einbeziehung der Struktur von Bäumen, durch die der potentielle Konfliktbereich von Operationen auf eine Menge von Knoten, nämlich Unterbäumen, ausgeweitet wird. Der *größere potentielle Konfliktbereich* und eine größere Vielfalt möglicher Konflikte führen zu einer aufwendigeren Konfliktbehandlung, so daß die Konsistenz von Bäumen schwieriger zu gewährleisten ist.

Die Auswirkungen des potentiell größeren Konfliktbereichs von Bäumen gegenüber Mengen, deren Datenelemente keine Beziehungen untereinander haben, soll an einem Beispiel mit einer Lösche-Operation aufgezeigt werden, wie es in der Abbildung 4.6 dargestellt ist: Ein Nutzer löscht den Unterbaum 12. Allerdings wußte dieser bei Auftragserteilung nicht, daß zu diesem Zeitpunkt bereits zwei bestätigte Operationen vorhanden waren, die dem verwendeten Replikatverwalter unbekannt waren. Eine Erstelle-Operation erstellte den Knoten 16 mit dem Vater 14, eine Verschiebe-Operation verschob den Unterbaum 20 unter den Knoten 15 in den zu löschenden Unterbaum. Hätte der Nutzer den Unterbaum 12 nicht gelöscht, wenn er systemweit vollständiges Wissen über die Struktur des Unterbaumes zum Erteilungszeitpunkt des Lösche-Auftrags besessen hätte? Wie kann erkannt werden, daß es eine Art von Konflikt zwischen der Lösche-Operation des Knotens 12 und der Verschiebe-Operation gibt, die sich auf echte Nachfolger in der Struktur des Baumes bezieht?

Dieses Beispiel zeigt, daß der bisherige Konflikt-Begriff nicht weitreichend genug gefaßt ist. Dieser soll daher durch die Nutzersicht zum Zeitpunkt der Erteilung von Aufträgen erweitert werden, wie es folgende Fragen ausdrücken: Bezogen auf welchen Zustand agierte ein Nutzer bei dem Erteilen eines Auftrags? Liegt dieser Zustand bei Ausführung der zum Auftrag zugehörigen Operation vor?

⁴Es sei an die eingeschränkten Typen von Operationen erinnert, welche in dieser Arbeit betrachtet werden.

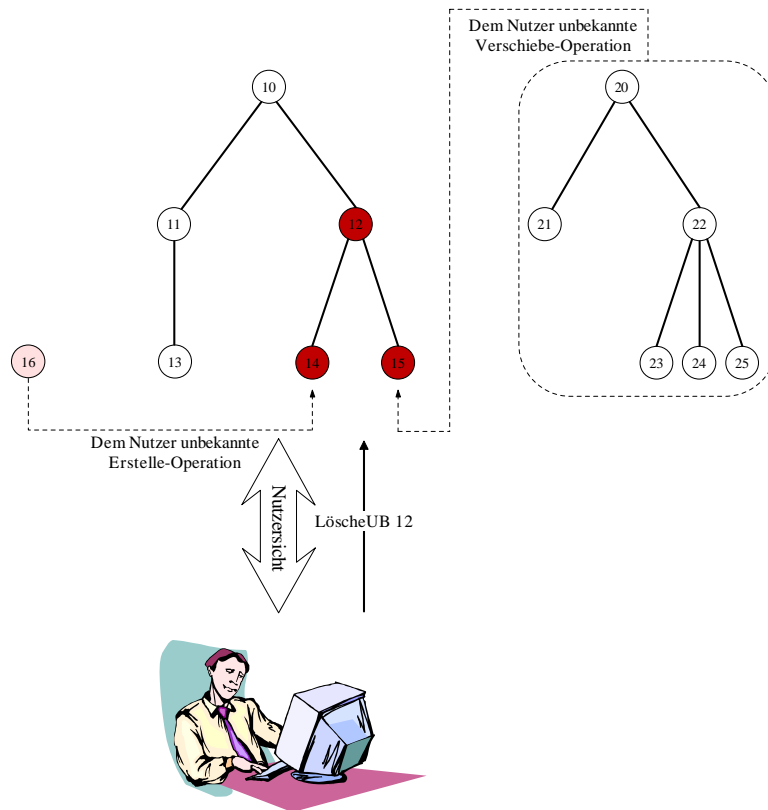


Abbildung 4.6: Sicht eines Nutzers beim Löschen eines Unterbaumes

Zur Beantwortung dieser Fragen muß zu jeder Operation die Nutzersicht zum Zeitpunkt der Erteilung eines Klientenauftrags festgehalten werden, und dies nicht nur für einen einzelnen Knoten, sondern je nach Operationstyp für vollständige Unterbäume. Vor der Ausführung einer Operation muß überprüft werden, ob der vorliegende Unterbaum in dem Replikatbestand konsistent mit der Nutzersicht bei Erteilung des Klientenauftrags ist. Dies führt zu dem Begriff der Kontextdiskrepanz und der damit verbundenen Erweiterung des Konflikt-Begriffs:

KONTEXTDISKREPANZ, KONFLIKT

Eine Schreiboperation bezieht sich auf die Datenstruktur eines Baumes und hat als Operanden maximal zwei Unterbäume. Eine **Kontextdiskrepanz** liegt vor, wenn die Bezugsbäume zum Zeitpunkt der Ausführung einer Schreiboperation und zum Zeitpunkt der Erstellung des zugehörigen Klientenauftrags voneinander abweichen. Ein **Konflikt** für die Ausführung von Schreiboperationen liegt vor, wenn mindestens ein nachfolgend beschriebener Fall eintritt:

- I) *Die Ergebnisse der Ausführung von zwei oder mehr Schreiboperationen sind unvereinbar.*
- II) *Es liegt eine Kontextdiskrepanz vor, die für eine Anwendung inakzeptabel ist.*

Ein Anwendungsentwickler spezifiziert für seine Anwendung, wann eine Kontextdiskrepanz akzeptabel oder inakzeptabel ist. Beispielsweise wäre es inakzeptabel, wenn durch eine Kontextdiskrepanz ein nicht intendierter Informationsverlust auftritt (wie im vorherigen Beispiel). Als akzeptabel hingegen könnte gelten, daß eine Kontextdiskrepanz zu keinem Informationsverlust führt, wie beispielsweise eine Abweichung der Bezugsbäume beim Verschieben.

Mit Hilfe der eingeführten Begriffe soll eine schlichte Konfliktbehandlung für die Operationen unseres Diskurssystems aufgestellt werden. Eine aufwendigere Behandlung von Konflikten ist denkbar, jedoch nicht Gegenstand dieser Arbeit.

Eine Konfliktbehandlung wird autonom sowohl auf dem primären Replikatverwalter als auch auf den sekundären Replikatverwaltern durchgeführt. Auf eine systemweite Koordination der Konfliktbehandlung wird verzichtet, damit die Anforderung an eine autonome Bearbeitung von Klientenaufträgen durch die Replikatverwalter erfüllt wird. Die fehlende systemweite Koordination stellt kein Problem dar, solange eine deterministische Konfliktbehandlung vorliegt, die bei gleichem Wissensstand von unterschiedlichen Replikatverwaltern (bezüglich aller Schreiboperationen, deren Position in der systemweiten Ausführungsreihenfolge vor der Schreiboperation ist, für die ein Konflikt vorliegt) das gleiche Ergebnis in den Replikatbeständen erzeugt. An dieser Stelle soll die Operationssemantik, die in Tabelle 4.1 auf Seite 40 beschrieben ist, präzisiert werden. Die dort dargestellte Bedeutung der Ausführung einer Operation gilt nur für den konfliktfreien Fall, wo das Ergebnis einer solchen Ausführung widerspruchsfrei ist. So kann eine Ändere-Operation nicht zu einer Änderung führen, wenn zum Ausführungszeitpunkt der Bezugsknoten dieser Operation bereits gelöscht ist. Aus diesem Grund gilt im Fall eines Konflikts eine andere Operationssemantik, in der die „eigentliche“ Aufgabe einer Operation, beispielsweise das Ändern, durch eine andere Aktion, beispielsweise das Versenden einer E-Mail über eine nicht mögliche Änderung des Knotens, ersetzt wird. Auch bei Konflikten wird somit eine Operation „ausgeführt“, d.h. zu einem Abschluß gebracht. Hier sei daran erinnert, daß die Bestätigung einer Operation in unserem Replikationskonzept deren

Position in der systemweiten Ausführungsreihenfolge festlegt und nicht über deren Abbruch oder Ausführung entscheidet, wie dies im Kontext verteilter Transaktionen der Fall ist.

In Tabelle 4.2 sind die Operationen in der Reihenfolge ihrer Ausführung eingetragen, die in unserer Konfliktbehandlung miteinander in Konflikt stehen können. Dabei sind drei wesentliche Fälle zu betrachten. Der Fall A entspricht der Behandlung von Konflikten, wie sie in den bisher untersuchten Replikationskonzepten beschrieben wurden (I). Die Fälle B und C stellen eine Kontextdiskrepanz dar, die aus Sicht unserer Anwendung zu behandeln ist (II).

1. Op. \ 2. Op.	Erstelle	Lösche	Ändere	Verschiebe
Erstelle	KK ^a	Fall B	KK	KK
Lösche	Fall A	KK	Fall A	Fall A
Ändere	KK	Fall B	Fall C	KK
Verschiebe	KK	Fall B	KK	KK

^a ‚KK‘ steht für ‚Keine Konflikte‘.

Tabelle 4.2: Konflikte zwischen verschiedenen Schreiboperationen

Fall A Eine Operation bezieht sich auf Unterbäume, die bereits gelöscht sind. So kann beispielsweise eine Operation die Aufgabe haben, einen Knoten unter einem Vater zu erstellen, der bereits gelöscht ist. Weil die Aufgabe der Operation in diesem Konfliktfall nicht zu erfüllen ist, ergibt die Ausführung der Operation auf den sekundären Replikatverwaltern keine Ergebnisse, während der primäre Replikatverwalter bei Bestätigung der Operation eine E-Mail an den Nutzer verschickt, daß seine Operation auf Grund eines Konflikts nicht das gewünschte Ergebnis erbringen konnte. Die E-Mails werden nur von dem primären Replikatverwalter versendet, damit ein Nutzer nur einmalig eine Nachricht für die endgültige Konfliktbehandlung erhält.

Fall B Ein Unterbaum soll gelöscht werden, der bei Ausführung der Operation nicht dem Unterbaum entspricht, für den ein Nutzer den Lösche-Auftrag erteilt hat. Es liegt also eine Kontextdiskrepanz vor. Daher muß eine Entscheidung getroffen werden, ob der Löschvorgang einer Lösche-Operation trotz eines

Unterschieds stattfinden soll oder nicht. Für ein Unterlassen des Löschens spricht, daß vielleicht wichtige Informationen verloren gehen (z.B. wurde ein Unterbaum in die zu löschende Struktur geschoben) und dies nicht im Sinne des Nutzers ist. Auf der anderen Seite möchte vielleicht ein Moderator einer Diskussion einen großen Unterbaum löschen, auch wenn auf Grund der hohen Knotenanzahl weiterhin unbekannte Schreiboperationen für den Unterbaum vorliegen. Deswegen wird dem Nutzer bei der Erteilung eines Auftrags zum Löschen die Möglichkeit gegeben, seine Entscheidung für den Konfliktfall zu spezifizieren: Ein Parameter legt fest, ob ein Unterbaum unbedingt oder nur bedingt zu löschen ist. Im ersten Fall wird der Unterbaum unabhängig von einer Kontextdiskrepanz gelöscht, im zweiten hingegen nicht.

Fall C Es liegt eine Kontextdiskrepanz vor, weil ein Attribut eines Bezugsknotens, welches durch eine Ändere-Operation einen neuen Wert erhalten soll, einen anderen Wert als bei Auftragserteilung hat. In diesem Fall wird einem Informationsverlust dadurch vorgebeugt, daß die Attributwerte in einer hier nicht näher spezifizierten Weise zusammengeführt werden.

Es bleibt die Frage zu klären, wie effizient eine Kontextdiskrepanz festzustellen ist. Eine Möglichkeit wäre das Anheften der kompletten Bezugsbäume als Kopie an eine Operation. Dies würde allerdings zu hohem Kommunikationsaufwand beim Übertragen der Operationen während der Abgleichsitzungen führen. Diese Lösung ist daher ungeeignet. Eine effiziente und elegante Möglichkeit zur Lösung dieses Problems bietet der Einsatz von *Prüfsummen* auf der rekursiven Struktur, deren Zustand festzuhalten ist. Diese Idee ist angelehnt an die Verwendung von *Hash*-Funktionen, die im Gebiet der Informationssicherheit für die Bildung von digitalen Signaturen eingesetzt werden. Es sollte ein Prüfsummenalgorithmus mit hoher Bitzahl der Prüfsumme verwendet werden, so daß mit sehr hoher Wahrscheinlichkeit durch einen Vergleich der Prüfsummen eine Kontextdiskrepanz festgestellt werden kann. Dazu wird bei Bedarf jeder Operation bei Erzeugung eine Prüfsumme angehängt, welche die Sicht des Nutzers bei Auftragserteilung repräsentiert. Diese wird später beim Ausführen der Operation mit der Prüfsumme des aktuellen Unterbaumes verglichen. Der Einsatz einer Prüfsumme ist nur bei solchen Operationen notwendig, bei denen eine kompakte Repräsentation einer größeren Datenstruktur benötigt wird. Beispielsweise kann dies im Fall einer Lösche-Operation ein vollständiger Unterbaum sein.

Der Einsatz einer Prüfsumme kann aber auch bei Ändere-Operationen angebracht sein, wenn dort der Zustand eines Attributs mit größerer Datenmenge (beispielsweise ein Dokument) festzuhalten ist.

Die Tabelle 4.3 beschreibt, welche Knoten und deren Attribute für eine Prüfsummenbildung zu berücksichtigen sind, um Kontextdiskrepanzen und Konflikte erkennen zu können. Der Operator \uplus symbolisiert dabei die Berechnung der Prüfsumme.

Operation	Prüfsumme
Erstelle(kID)	Keine Prüfsumme notwendig ^a
Lösche(kID)	\uplus (Unterbaum kID ohne vater von kID)
Ändere(kID, attributnamen & zuweisungen)	\uplus (Attribute mit attributnamen von kID)
Verschiebe(kID ₁ , kID ₂)	Keine Prüfsumme notwendig ^b

^aBei Ausführung der Operation wird überprüft, ob ein Knoten mit dem Vater kID vorhanden ist. Diese Information ist bereits in der Erstelle-Operation enthalten.

^bBei Ausführung der Operation wird überprüft, ob die Knoten mit dem Kennzeichner kID₁ und kID₂ vorhanden sind. Diese Informationen sind bereits in der Verschiebe-Operation enthalten.

Tabelle 4.3: Prüfsummenbildung für Schreiboperationen

4.3 Abgleich

Dieser Abschnitt beginnt mit einem Überblick über den Abgleichprozeß. Anschließend werden verteilte Algorithmen diskutiert, deren Verwendung im Abgleichprozeß denkbar ist. Danach wird das Konzept der Abgleichsitzungen vorgestellt, in denen Replikatverwalter paarweise ihr Wissen austauschen. Es folgt die Entwicklung eines Algorithmus, der die Replikatverwalter für Abgleichsitzungen auswählt. Danach werden die Auswirkungen einer partiellen⁵ Replikation auf den Abgleichprozeß untersucht.

⁵Die Bezeichnung Replikatverwalter ohne nähere Qualifikation wird im Interesse einer flüssigen Sprache in diesem Abschnitt, außer im Teilbereich „Abgleichsitzungen bei partieller Replikation“, lediglich für stationäre Replikatverwalter verwendet.

Der Wissensaustausch der Replikatverwalter wird in sogenannten Abgleichsitzungen⁶ durchgeführt. Diese Sitzungen finden immer genau zwischen zwei Replikatverwaltern statt, welche für diese Sitzung als Abgleichpartner bezeichnet werden. Nach einer Sitzung, in der bidirektional das Wissen ausgetauscht wird, haben beide Replikatverwalter das gleiche Wissen über die Schreiboperationen, die vor Beginn der Sitzung bekannt waren. Schreiboperationen, die während einer Sitzung auf einem der Replikatverwalter erzeugt werden, können übermittelt werden, müssen jedoch nicht. Eine Übermittlung solcher Schreiboperationen findet nur statt, wenn die Schreiboperationen vor dem Zeitpunkt, zu welchem sie in der Übertragungsreihenfolge zu übertragen sind, vorliegen. Anderenfalls werden sie in der nächsten Abgleichsitzung übertragen. Im Abgleichprozeß wird zwischen mobilen und stationären Replikatverwaltern unterschieden. Während stationäre Replikatverwalter lediglich eine vollständige Replikation unterstützen, ist mobilen Replikatverwaltern auf Grund ihrer beschränkten Ressourcen die partielle Replikation erlaubt. Mobile Replikatverwalter initiieren selbstständig nach Bedarf Abgleichsitzungen mit beliebigen stationären Replikatverwaltern. Stationäre Replikatverwalter sind hingegen in regelmäßig durchgeführten Abgleichsitzungen aktiv. In einer *Abgleichrunde* findet eine Anzahl von Sitzungen statt, in der jeder verfügbare stationäre Replikatverwalter genau in einer Sitzung aktiv ist. Gibt es beispielsweise zehn stationäre Replikatverwalter, so gibt es in einer Runde fünf Sitzungen. Nach einer Anzahl von Runden, dem *Abgleichzyklus*, ist das Wissen aller stationären Replikatverwalter ausgetauscht und jeder stationäre Replikatverwalter hat das Wissen eines beliebigen anderen stationären Replikatverwalters bezüglich mindestens aller Schreiboperationen, die vor der ersten Runde des Abgleichzyklus bekannt waren. Die Anlässe für Zyklen werden durch die *Abgleichpolitik* festgelegt. Ereignisse führen dabei zu neuen Zyklen (z.B. alle zehn Minuten oder wenn mehr als hundert neue Schreiboperationen auf einem stationären Replikatverwalter erzeugt wurden, usw.). Damit ist es möglich, den durchschnittlichen Grad an Abweichung der Replikatbestände über die Anzahl und Häufigkeit der Zyklen anzupassen. Eine solche Politik ist anwendungsspezifisch und nicht Gegenstand dieser Arbeit.

⁶In diesem Abschnitt kurz mit Sitzung bezeichnet, weil keine Verwechslungsgefahr mit einer Klientensitzung besteht.

An dieser Stelle soll diskutiert werden, warum Abgleiche zwischen Replikatverwaltern paarweise durchgeführt werden und warum *Multicast*- und *Commit*-Protokolle nicht für unsere Anforderungen geeignet sind.

Für den Abgleichprozeß sind folgende Anforderungen zu erfüllen:

- Für eine geringe Anzahl auszutauschender Nachrichten soll der Abgleich inkrementell erfolgen.
- Die Realisierung der Algorithmen soll einfach sein:
 - Das Abgleichverfahren soll möglichst ohne Änderungen zum Abgleich von mobilen und stationären Replikatverwaltern gleichermaßen verwendbar sein.
 - Es sollen keine Recovery-Maßnahmen für Replikatverwalter notwendig sein, die durch Fail-Stop-Fehler von Kommunikationsverbindungen und Netzknoten nicht verfügbar gewesen sind.
- Das entworfene Konsistenzmodell stellt folgende Anforderungen:
 - Die Übertragung von Schreiboperationen erfolgt geordnet in der Reihenfolge, wie es die Konsistenzbeziehungen 4.7 und 4.8 vorschreiben.
 - Es besteht lediglich die Notwendigkeit, den Abgleich nach bestem Bemühen (*best effort*) durchzuführen. Wenn keine Fehler im System vorliegen, haben alle Replikatverwalter nach einem Abgleich mindestens den gleichen Replikatbestand bezüglich aller Schreiboperationen, die vor Beginn des Abgleichs bekannt waren. Dies gilt nicht für Situationen mit Fail-Stop-Fehlern.
- Die folgenden Optimierungskriterien sollen gelten:
 - Das wichtigste Optimierungskriterium ist eine geringe Anzahl von Schritten des Algorithmus, so daß eine schnelle Verbreitung der Schreiboperationen erfolgt.
 - Das zweite Optimierungskriterium ist eine geringe Anzahl auszutauschender Nachrichten.

- Weil die Menge der stationären Replikativhalter in einer geringen Größenordnung liegt (um die zehn), ist der fehlerfreie Fall der Regelfall. Daher sind die Abgleichalgorithmen für diesen Fall zu optimieren.
- Es können nur technische Mittel in den Algorithmen verwendet werden, die hardwareseitig zur Verfügung stehen. Im Internet gibt es nur in einigen Subnetzen Multicasts (beispielsweise ist IP-Multicast im MBONE-Netz, <http://www.mbone.de>, vorhanden), so daß dieses Mittel in unserem Umfeld nicht zur Verfügung steht.

Für die Diskussion der Komplexität von Algorithmen soll folgende Modellierung für unser (partiell) synchrones System gelten:

- Das Senden und Empfangen einer Nachricht entspricht jeweils einem Schritt. Wenn beispielsweise eine Nachricht zum Zeitpunkt t versendet wird, dann wird diese zum Zeitpunkt $t + 1$ Schritte empfangen. Eine Antwort des Empfängers auf diese Nachricht kann somit frühestens zum Zeitpunkt $t + 2$ Schritte erfolgen.
- Jede Schreiboperation wird als einzelne Nachricht verschickt. Damit wird ein Maß für den zeitlichen Aufwand für das Verschicken einer Schreiboperation festgesetzt. Wird auf eine solche Annahme verzichtet, so entspräche beispielsweise die zeitliche Komplexität für das Verschicken von tausend Schreiboperationen einer Komplexität für das Verschicken einer einzelnen Schreiboperation.
- Die lokale Berechnung von Ergebnissen auf einem Netzknoten erfordert keinen Schritt, weil der Zeitimpuls („tick“) für die Taktung so gewählt wird, daß das Senden oder Empfangen einer Nachricht sowie eine Berechnung innerhalb eines Schritts möglich sind.

Das Problem für den Abgleichprozeß sei wie folgt (vereinfacht) beschrieben: Es existieren n stationäre Replikativhalter im System. Jeder Replikativhalter R_i hat eine Menge \mathcal{O}_i von Schreiboperationen, die keinem anderen Replikativhalter bekannt sind. Für eine einfachere Komplexitätsanalyse wird festgelegt, daß $\forall i (|\mathcal{O}_i| = u)$ gilt, daß also auf jedem Replikativhalter die gleiche Anzahl unbekannter Operationen u vorhanden ist. Ziel des systemweiten Abgleichs ist, daß die

Schreiboperationen, die vor dem Beginn des Abgleichs vorhanden sind, am Ende des Abgleichs allen Replikativverwaltern bekannt sind, sofern keine Fehler im System vorliegen. In Situationen mit Fail-Stop-Fehlern von Kommunikationsverbindungen bzw. Netzknoten sind die Schreiboperationen zwar in der durch die Konsistenzbeziehungen festgelegten Reihenfolge bekannt zu machen, aber es ist nicht notwendig, daß alle funktionstüchtigen Replikativverwalter das gleiche Wissen nach einem systemweiten Abgleich bezüglich aller Schreiboperationen haben, die vor Beginn des Abgleichs bekannt waren. Der Abgleich wird also nach (mehr oder weniger) bestem Bemühen durchgeführt.

Es gibt drei Klassen von Algorithmen, die sich auf den ersten Blick für eine Lösung dieses Problems anbieten:

Multicast-Algorithmen: Multicast-Algorithmen, wie beispielsweise in *ISIS* [BJ87] oder in [CS95] bieten sich an, um unbekannte Schreiboperationen von einem Replikativverwalter an jeden seiner Partner gleichzeitig zu verschicken. Das Internet unterstützt nur partiell Multicast auf Hardwarebasis, so daß ein Multicast in der Regel durch eine Menge von *Unicasts* zu ersetzen ist. Multicast-Algorithmen haben den Nachteil, daß sie zwar prinzipiell für die Verbreitung von Schreiboperationen zwischen den stationären Replikativverwaltern geeignet sind, nicht jedoch für die Verbreitung an mobile Replikativverwalter. Diese sind nicht ständig verfügbar und erhalten somit nicht die Schreiboperationen, die per Multicast verbreitet werden. Für den Abgleich mobiler Replikativverwalter muß deshalb ein anderes Verfahren für einen Abgleich eingesetzt werden. Dies widerspricht unserer Anforderung, daß mit einem Abgleichmechanismus mit möglichst wenig Änderungen sowohl der Abgleich zwischen stationären Replikativverwaltern als auch mit mobilen zu realisieren ist. Ein weiterer Nachteil ist der Bedarf an einer Recovery-Maßnahme, die ebenfalls unseren Anforderungen einer einfachen Realisierbarkeit widerspricht: Ist ein stationärer Replikativverwalter auf Grund eines Fail-Stop-Fehlers der Kommunikationsverbindung oder seines Netzknotens nicht verfügbar, während Multicasts stattfinden, so muß ihm das fehlende Wissen später durch eine Recovery-Maßnahme mitgeteilt werden. Des weiteren benötigen verlässliche Multicast-Algorithmen alle τ Zeitpunkte mindestens n Nachrichten (theoretisch gezeigt in [Chr91]), um in der Gruppenverwaltung festzustellen, ob einzelne Replikativverwalter nicht verfügbar sind und damit die Gruppenzugehörigkeit neu bestimmt werden muß. Dafür ist auch der Einsatz von Konsens-Algorithmen notwendig [CS95]. Ein solcher verlässlicher Multicast-Algorithmus ist

auch für unsere Anwendung trotz des lediglich besten Bemühens notwendig, weil garantiert werden muß, daß ein stationärer Replikatverwalter, der in einem Intervall nicht verfügbar gewesen ist und in dieser Zeit mindestens eine Schreiboperation „verpaßt“ hat, keine späteren Schreiboperationen bei einem Multicast empfängt, bevor er die verpaßten Schreiboperationen kennt (Konsistenzbeziehungen 4.7 und 4.8). Der hohe Kommunikationsaufwand für eine solche Gruppenverwaltung ist für unsere Anwendung nicht gerechtfertigt. Eine andere Situation hingegen liegt bei Replikationskonzepten mit starker Konsistenz vor, die häufig Multicast-Algorithmen einsetzen: Dort werden alle von einem Replikatverwalter erzeugten Schreiboperationen an seine Partner verschickt und sofort in deren Replikatbestand integriert. Ist ein Replikatverwalter nicht verfügbar, wird dies in der Gruppenverwaltung festgestellt und bei nachfolgender Verfügbarkeit eine Recovery-Maßnahme durchgeführt. Der Aufwand ist hier zur Realisierung der starken Konsistenz gerechtfertigt.

Konsensalgorithmen: Konsensalgorithmen, wie beispielsweise der Algorithmus von Lamport für die Erzielung eines Konsens bei Byzantinischen Fehlern [LSP82] oder das wohlbekannte *2-Phase-Commit-Protocol* [Gra78, BHG87] sind ebenfalls nicht für unsere Anforderungen geeignet. Auch wenn das Problem der Konsensalgorithmen auf dem ersten Blick dem unseren ähnlich erscheint, so sind diese jedoch verschieden: Das Ziel für einen Konsensalgorithmus besteht darin, daß sich alle Replikatverwalter bezüglich der *Meinung zu einem Thema einigen*. Für das vorliegende Problem bedeutet dies, daß $u * n$ verschiedene Themen vorliegen, zu welcher jeder Replikatverwalter die Meinung „kenne ich“ oder „kenne ich nicht“ hat. Vor dem Abgleich hat zu jedem Thema (d.h. unbekannte Schreiboperation) lediglich ein Replikatverwalter die Meinung „kenne ich“. Ziel ist es, diese Meinung in Form der Schreiboperation selbst den anderen Replikatverwaltern bekannt zu machen und damit zu verbreiten. Einen Konsens zu einem *bestimmten* Zeitpunkt zu erreichen, daß alle Replikatverwalter zu einer Schreiboperation die gleiche Meinung haben, wird in unserem Abgleichprozeß nach bestem Bemühen nicht angestrebt. Es wird lediglich garantiert, daß durch regelmäßige Abgleiche aller Replikatverwalter eine Schreiboperation irgendwann überall bekannt wird und die Schreiboperationen in einer bestimmten Reihenfolge zu übermitteln sind. Ein 2-Phase-Commit-Protokoll benötigt $u * n * (4 * (n - 1))$ Nachrichten, um einen Konsens herzustellen, der in unserem

Problem nicht erforderlich ist, so daß der dafür notwendige Kommunikationsaufwand nicht zu rechtfertigen ist. Auf Grund der abweichenden Problemstellung der Konsensalgorithmen haben Multicast-Algorithmen einen weitaus engeren Bezug zu unserem Problem.

Epidemische Algorithmen für Abgleichprozesse: Epidemische Algorithmen für Abgleichprozesse verwenden paarweise Abgleichsitzungen, die sowohl mobile als auch stationäre Replikativhalter gleichermaßen abgleichen können. Des weiteren sind keine Recovery-Maßnahmen notwendig, weil bei Beginn einer Sitzung der Wissensstand des Partners erfragt wird und diesem dann passend die unbekannt Schreiboperationen in der vorgegebenen Reihenfolge übermittelt werden, selbst wenn eine vorherige Sitzung unplanmäßig abgebrochen wurde. Diese Eigenschaften sind auch ein Grund, warum epidemische Algorithmen mit Abgleichsitzungen in Replikationskonzepten mit schwacher Konsistenz bei einer Anzahl von mindestens mehreren hundert Replikativhaltern, wie im Projekt Bayou [PST⁺97], eingesetzt werden.

Die Idee epidemischer Algorithmen, die Wissen „wie eine Krankheit“ verbreiten, ist wie folgt: Regelmäßig wählt jeder stationäre Replikativhalter einen anderen *zufällig* für eine Sitzung als Partner aus. Diese beiden Replikativhalter tauschen in einer solchen Sitzung jene Schreiboperationen aus, welche dem Partner unbekannt sind. Für die Schreiboperationen, die einem Partner nicht übermittelt werden müssen, weil diese dort bereits bekannt sind, gilt der Partner als *infiziert*. Damit der Verbreitungsprozeß zu einem Ende kommt, verliert ein Replikativhalter bei der Verbreitung von Schreiboperationen an Interesse, wenn seine Partner die Schreiboperationen bereits kennen. Die Wahrscheinlichkeit, daß ein Replikativhalter bestimmte Schreiboperationen zu einem Zeitpunkt t kennt, steigt exponentiell mit t , wobei nicht garantiert werden kann, daß jeder Replikativhalter die Schreiboperationen zu dem Zeitpunkt t erhalten hat. Es läßt sich jedoch zeigen, daß bei geschickter Wahl der Algorithmenparameter eine sehr hohe Wahrscheinlichkeit erreicht werden kann, daß die Schreiboperationen zum Zeitpunkt t allen Replikativhaltern bekannt sind [DGH⁺87].

Epidemische Algorithmen sind für eine hohe Anzahl von Replikativhaltern (üblicherweise einige hundert) gut geeignet. Dabei wird angenommen, daß bei einer

hohen Anzahl von Replikatverwaltern der Fehlerfall der Regelfall ist. Die zufällige Auswahl von Partnern bietet eine gute Möglichkeit, mit Fehlern nach bestem Bemühen umzugehen, indem eine unplanmäßig abgebrochene Sitzung nicht wieder aufgenommen wird und in der nächsten Runde ein zufälliger und damit meist anderer Replikatverwalter ausgewählt wird. Der Nachteil ist jedoch, daß durch die zufällige Auswahl der Replikatverwalter das Wissen nicht optimal ausgetauscht wird, weil ein Partner bereits viele Schreiboperationen kennen kann, die in der aktuellen Abgleichsitzung verbreitet werden sollten. Ein weiterer Nachteil ist, daß nur Wahrscheinlichkeitstheoretische Aussagen darüber gemacht werden können, ob die Replikatverwalter eine Menge bestimmter Schreiboperationen zu einem Zeitpunkt t kennen. Bei einer geringeren Anzahl von Replikatverwaltern, wie in unserem Anwendungsumfeld vorhanden, ist der fehlerfreie Fall der Regelfall und es kann deterministisch bestimmt werden, wie das Wissen zu verteilen ist, damit dieses bei möglichst gleicher Last der Netzknoten und Kommunikationsverbindungen in wenigen Schritten verbreitet werden kann. Des Weiteren können Aussagen gemacht werden, nach wie vielen Schritten das Wissen garantiert verbreitet ist. Ein solcher Algorithmus entspräche damit einem epidemischen Algorithmus, der zufällig nur die optimalen Partner für eine Wissensverbreitung auswählt.

Abgleichsitzungen

Das Ziel eines Zyklus ist ein vollständiger Wissensaustausch zwischen allen Replikatverwaltern. Dazu wird eine Anzahl von paarweisen Abgleichsitzungen durchgeführt, in denen zwei Replikatverwalter ihr Wissen bidirektional austauschen. Um diesen Austausch so effizient wie möglich zu gestalten, werden nur die Operationen übertragen, deren Ausführung die Änderungen an den Bäumen der Replikatbestände verursacht hat und nicht die geänderten Bäume oder Knoten selbst. Dadurch reduziert sich die Menge zu übertragender Daten. Im Gegensatz zu Dateisystemen, in welchen eine Serie von Operationen einen häufig temporären Charakter hat (beispielsweise bei *Compilern*, die temporäre Dateien anlegen und unmittelbar danach wieder löschen), sind solche Serien von Operationen bei dem Klientenverhalten in unserem Diskurssystem nicht zu erwarten. Dies kann aus den Erfahrungen mit den Vorgängern des zu realisierenden Diskurssystems geschlossen werden.

Operationen werden aus den Schreibaufträgen von Klienten erzeugt, die bei einzelnen Replikativverwaltern nebenläufig eingegangen sind. Auf jedem Replikativverwalter existiert eine logische Zeit, die jeder Schreiboperation bei Erzeugung zugewiesen wird. Diese logische Zeit wird *Annahmenummer* der Schreiboperation genannt und besteht aus dem Kennzeichner des Replikativverwalters und einem einfachen lokalen Zähler, der bei jeder Erzeugung einer Schreiboperation, ausgelöst durch einen angenommenen Klientenauftrag, um Eins erhöht wird. Der primäre Replikativverwalter bestätigt zudem jede Schreiboperation unmittelbar nach deren Erzeugung und vergibt dazu eine *Bestätigungsnummer*. Die Bestätigungs- und Annahmenummer einer bestätigten Schreiboperation sind gleich, wenn der primäre Replikativverwalter diese erzeugt hat. Die Bestätigungs- und Annahmenummer einer bestätigten Schreiboperation sind unterschiedlich, wenn ein sekundärer Replikativverwalter diese Schreiboperation erzeugt hat und der primäre Replikativverwalter diese später bestätigt. Eine solche Bestätigung führt ebenso wie die Annahme eines Klientenauftrags zu einer Erhöhung der logischen Zeit des primären Replikativverwalters.

Schreiboperationen auf jedem einzelnen Replikativverwalter sind serialisiert. Die unbestätigten Schreiboperationen der sekundären Replikativverwalter werden nebenläufig erzeugt und haben zuerst keine Position in der systemweiten Ausführungsreihenfolge (Bestätigungsreihenfolge). Eine Position wird erst durch die Bestätigung des primären Replikativverwalters festgelegt (eine Bestätigung erfolgt nach Konsistenzbeziehung 4.5 garantiert). Die Bestätigungsreihenfolge berücksichtigt die potentiell kausalen Beziehungen von Schreiboperationen, die von demselben Replikativverwalter erzeugt wurden (Konsistenzbeziehung 4.4). Die Schreiboperationen, die von unterschiedlichen Replikativverwaltern erzeugt wurden, stehen auf Grund der nebenläufigen und autonomen Erzeugung in keiner kausalen Beziehung, so daß diese in beliebiger Reihenfolge serialisiert werden können. Der primäre Replikativverwalter serialisiert ihm unbekannte, unbestätigte Schreiboperationen während der Abgleichsitzungen. Weil die Reihenfolge zur Übertragung von unbestätigten Schreiboperationen bereits deren potentiell kausale Beziehungen berücksichtigt (Konsistenzbeziehung 4.7), kann der Bestätigungsvorgang auf dem primären Replikativverwalter sehr einfach realisiert werden: Der primäre Replikativverwalter serialisiert die Schreiboperationen in der Reihenfolge, in der sie ihm bekannt werden. Um eine hohe Verbreitungsgeschwindigkeit des Wissens über die Bestätigungsreihenfolge zu bekommen, überträgt ein sekundärer Replikativverwalter sein Wissen in einer Abgleichsitzung mit

dem primären Replikatverwalter zuerst. Dadurch kann der primäre Replikatverwalter die soeben erhaltenen unbestätigten Schreiboperationen sofort bestätigen und die Bestätigungsdaten (kurz Bestätigung) im zweiten Teil der Abgleichsitzung seinem Partner übermitteln. Die Bestätigungen werden ebenso wie die Schreiboperationen selbst in den Abgleichsitzungen (auch zwischen zwei sekundären Replikatverwaltern) verbreitet.

In einer Sitzung sollen dem Partner nur solche Schreiboperationen und Bestätigungen übertragen werden, welche diesem unbekannt sind. Aus diesem Grund muß ein Austausch über den Wissensstand des jeweiligen Partners erfolgen. Dazu ist eine effiziente Repräsentation des Wissens eines Replikatverwalters erforderlich. Die Reihenfolge, in der unbestätigte und bestätigte Schreiboperationen sowie Bestätigungen ausgetauscht werden (Konsistenzbeziehungen 4.7 und 4.8), hat eine angenehme Eigenschaft: Es können keine „Lücken“ in der Reihenfolge der bekannten Schreiboperationen entstehen. Die bestätigten Schreiboperationen und Bestätigungen werden geordnet nach der Bestätigungsnummer übermittelt. Danach werden für jeden Replikatverwalter die unbestätigten Schreiboperationen nach deren Erzeugungsreihenfolge übertragen. Damit können die bekannten bestätigten Schreiboperationen durch eine einzige Zahl, *der höchsten bekannten Bestätigungsnummer* und die bekannten unbestätigten Schreiboperationen durch eine Zahl pro Replikatverwalter für die höchste bekannte Annahmenummer, dem *Annahmevektor*, charakterisiert werden. Beispiel: Es existieren drei Replikatverwalter. R_1 hat insgesamt 10 unbestätigte Schreiboperationen bis zu seiner logischen Zeit 10 erzeugt. Der primäre Replikatverwalter R_2 besitzt 20 bestätigte Schreiboperationen bis zu seiner logischen Zeit 20. R_3 hat 4 unbestätigte Schreiboperationen bis zu seiner logischen Zeit 4 erzeugt und führt nacheinander eine Sitzung mit R_1 und R_2 durch. Danach besitzt R_3 das folgende Wissen: Seine eigene logische Zeit ist 4, die höchste bekannte Annahmenummer von R_1 ist 10 und die höchste bekannte Bestätigungsnummer ist 20. Dabei ist garantiert, daß R_3 bei einem Wissensstand, in welchem die höchste bekannte Bestätigungsnummer 20 beträgt, alle bestätigten Schreiboperationen empfangen hat, deren Bestätigungsnummer kleiner oder gleich 20 ist. Eine Lücke, in der beispielsweise die Schreiboperationen 4 und 12 fehlen, kann es nicht geben. Die geordnete Übermittlung der Schreiboperationen und Bestätigungen in einer Sitzung hat des weiteren den Vorteil, daß die Konsistenz selbst dann ohne Recovery-Maßnahmen erhalten bleibt, wenn eine Sitzung unplanmäßig abgebrochen wird.

Zusammengefaßt läuft eine Sitzung zwischen einem Replikatverwalter R_1 und R_2 wie folgt ab:

1. R_1 fragt R_2 nach dessen Wissensstand, der durch den Annahmevektor und der höchsten bekannten Bestätigungsnummer charakterisiert wird.
2. R_1 ermittelt alle Schreiboperationen und Bestätigungen, die R_2 nicht kennt.
3. R_1 sendet R_2 alle bestätigten, R_2 unbekanntem Schreiboperationen sowie Bestätigungen in der Bestätigungsreihenfolge.
4. R_1 sendet R_2 alle unbestätigten, R_2 unbekanntem Schreiboperationen in der Erzeugungsreihenfolge.
5. R_1 und R_2 führen die Schritte 1-4 in vertauschten Rollen aus, damit ein bidirektionaler Wissensaustausch nach Beendigung der Sitzung vollzogen ist.

Partnerwahl in Abgleichrunden und -zyklen

Neben dem Ablauf einer Sitzung ist auch die Verbreitung des Wissens über Änderungen der Replikatbestände im System interessant. Dazu ist eine Auswahl der Partner der Sitzungen für jede Runde und die Anzahl der Runden für einen Zyklus von Bedeutung. Das Ziel besteht in einer möglichst kostengünstigen Verbreitung des Wissens, charakterisiert insbesondere durch eine geringe Anzahl von Schritten.

Die Problembeschreibung von Seite 63 soll an dieser Stelle zur Erinnerung kurz wiederholt werden: Es existieren n stationäre Replikatverwalter. Jeder Replikatverwalter R_i besitzt eine Menge \mathcal{O}_i von Schreiboperationen, die vor dem systemweiten Abgleichprozeß keinem anderen stationären Replikatverwalter bekannt sind. Es gilt $\forall i (|\mathcal{O}_i| = u)$. Das Ziel des systemweiten Abgleichs liegt darin, daß die Schreiboperationen (das Wissen), die vor dem Beginn des Abgleichs vorhanden sind, am Ende des Abgleichs allen Replikatverwaltern bekannt sind. Es seien keine Fehler im System vorhanden (Fehlersituationen werden ausführlich in 4.4 diskutiert).

Der folgende Algorithmus wählt die Replikatverwalter für Abgleichsitzungen durch eine *Permutation* der potentiellen Partner für jede Runde in einem Zyklus aus. Das Prinzip zur Bestimmung der Partner ist wie folgt: Es wird versucht, daß jeder Replikatverwalter mit einem Partner kommuniziert, dessen Wissen möglichst *verschieden* von dem eigenen ist. Damit wird die Last möglichst gleichmäßig auf alle

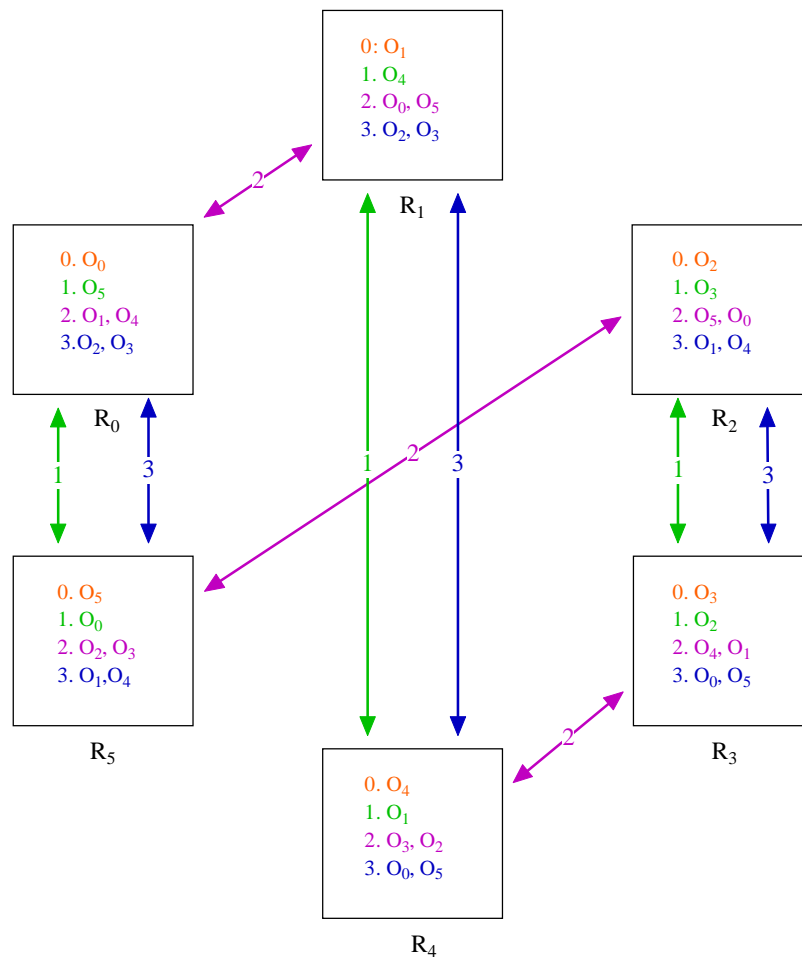


Abbildung 4.7: Partnerwahl in einem Zyklus

Netzknoten im System verteilt und das Wissen verbreitet sich besonders schnell. Ein Beispiel zur Darstellung der Funktionsweise des Algorithmus ist in Abbildung 4.7 für sechs Replikativverwalter gezeigt: Vor Beginn des Zyklus hat jeder Replikativverwalter eine Menge von Schreiboperationen (orange dargestellt), die allen anderen Replikativverwaltern unbekannt sind. In der ersten Runde (grün) finden Sitzungen zwischen (R_0, R_5) , (R_1, R_4) sowie (R_2, R_3) statt. Dabei erhält R_0 beispielsweise die Schreiboperationen O_5 . Im nächsten Zyklus (violett) tauschen sich (R_0, R_1) , (R_2, R_5) sowie (R_3, R_4) aus. Nach dieser Runde hat beispielsweise R_2 das Wissen über die Schreiboperationen O_0, O_2, O_3, O_5 . Im letzten Schritt⁷ (blau) erfolgen dann Sitzungen zwischen (R_0, R_5) , (R_1, R_4) und (R_2, R_3) , wonach jeder Replikativverwalter alle Schreiboperationen kennt.

⁷Es ist Zufall, daß die Partner in der ersten und dritten Runde identisch sind.

Das Prinzip der Permutation zur Ermittlung des Partners ist in Abbildung 4.8 dargestellt. In der ersten Runde werden die Partner in einer festen Reihenfolge ausgewählt. Danach erfolgt in jeder Runde eine Verschiebung der Partner (rot dargestellt), wobei unten „hinausgeschobene“ Replikatverwalter oben wieder eingesetzt werden (blau dargestellt). Die Verschiebung erfolgt um eine Zweierpotenz von Positionen, die abhängig von der Rundenanzahl ist und dafür sorgt, daß genau jene Replikatverwalter kommunizieren, die möglichst wenig Wissen teilen.

Replikatverwalter	Partner Runde 1	Partner Runde 2	Partner Runde 3
0	5	1	5
1	4	0	4
2	3	5	3
3	2	4	2
4	1	3	1
5	0	2	0

Verschiebung zur vorherigen Runde

Übertrag der Replikatverwalter, welche 'hinausgeschoben' wurden

Abbildung 4.8: Permutationsprinzip der Partnerwahl

Der Algorithmus, der die systemweite Auswahl der Partner bei einer gegebenen Anzahl von Replikatverwaltern berechnet, soll hier im Pseudocode angegeben werden:

```
void BerechnePartner(n)
{
  int v=-1; int mehrAufwand=0;
  int r, k, partner;
  if(n==ungeradeZahl()) mehrAufwand++;
  for(r=1, r<=⌈log2(n) + mehrAufwand⌉, r++) //Rundenanzahl
  {
    v=v+2r-1; //Verschiebungswert für Permutation
    for(k=0, k<=n-1, k++)
    {
      partner=Modulus(Modulus(n-1+v, n)-k+n, n); //Durchführung P.
      println("Austausch von "+k+", "+partner);
    }
  }
}
```

Zuerst erfolgt eine Prüfung, ob eine gerade oder ungerade Anzahl von Replikativverwaltern vorliegt. Liegt eine ungerade Anzahl vor, so ist eine Zusatzrunde notwendig, weil in jeder Runde ein Replikativverwalter keinen Partner für eine Sitzung findet. Aus diesem Grund ist es für die Wissensverbreitung ökonomischer, wenn eine gerade Anzahl von Replikativverwaltern vorliegt, weil damit jeder Replikativverwalter in jeder Runde an einer Sitzung teilnehmen kann. Danach erfolgt die Berechnung der Rundenanzahl und für jede Runde wird eine zyklische Verschiebung der Partner durchgeführt. Mit diesem Algorithmus kann jeder einzelne Replikativverwalter bestimmen, wann er mit welchem Partner eine Sitzung durchzuführen hat. Die Taktung der Runden kann einfach umgesetzt werden. Ist ein Replikativverwalter mit einer Sitzung fertig und kontaktiert seinen Partner für die nächste Runde, der mit seiner Sitzung der vorherigen Runde noch beschäftigt ist, so nimmt dieser die Sitzung in einem Wartemodus an, bis die vorherige Sitzung beendet ist.

Die zeitliche Komplexität des Algorithmus wird als Schritte und der Aufwand als Anzahl von Nachrichten angegeben, um diese mit anderen Algorithmen vergleichen zu können. Ein anderes Maß der Komplexität ist die Anzahl von Runden und Sitzungen, die unabhängig von der Anzahl der unbekanntenen Schreiboperationen u pro Replikativverwalter ist:

$$\begin{aligned}
 \text{AnzahlSchritte}(n, u) &= \begin{cases} 4u(n-1) + 2c \lceil \log_2 n \rceil & : n \text{ gerade} \\ 6u(n-1) + 2c(\lceil \log_2 n \rceil + 1) & : n \text{ ungerade} \end{cases} \\
 \text{AnzahlNachrichten}(n, u) &= \begin{cases} u(n^2 - n) + c \frac{n}{2} \lceil \log_2 n \rceil & : n \text{ gerade} \\ u(n^2 - n) + c \frac{n}{2} (\lceil \log_2 n \rceil + 1) & : n \text{ ungerade} \end{cases} \\
 \text{AnzahlRunden}(n) &= \begin{cases} \lceil \log_2 n \rceil & : n \text{ gerade} \\ \lceil \log_2 n \rceil + 1 & : n \text{ ungerade} \end{cases} \\
 \text{AnzahlSitzungen}(n) &= \begin{cases} \frac{n}{2} \lceil \log_2 n \rceil & : n \text{ gerade} \\ \lfloor \frac{n}{2} \rfloor (\lceil \log_2 n \rceil + 1) & : n \text{ ungerade} \end{cases}
 \end{aligned}$$

Die Anzahl der Schritte und Nachrichten ist abhängig von der Anzahl n der Replikatverwalter und den unbekanntem Schreiboperationen u . Die Konstante c steht stellvertretend für die konstante Anzahl von Nachrichten, die in einer Abgleichsitzung für den Austausch der Wissensstände der Partner notwendig sind. In einer optimalen Realisierung ist $c = 2$. In unserer Realisierung gilt $c = 4$, was jedoch im Hinblick darauf, daß in einer Abgleichsitzung hunderte von Schreiboperationen ausgetauscht werden, unerheblich ist.

An dieser Stelle soll an einem Zahlenbeispiel verdeutlicht werden, wie wenig Schritte unser Algorithmus für einen systemweiten Abgleich benötigt. Dazu vergleichen wir ihn mit einem einfachen Ansatz, in welchem die Replikatverwalter nacheinander ein Multicast von allen unbekanntem Schreiboperationen durchführen. Nehmen wir eine typische Situation, in der 100 unbekanntem Schreiboperationen ($u = 100$) auf jedem der zehn Replikatverwalter ($n = 10$) vorliegen. In jeder Runde versendet ein Replikatverwalter $u * (n - 1)$ Nachrichten an $n - 1$ Replikatverwalter. Damit benötigt jede Runde $2u * (n - 1)$ Schritte, weil ein Multicast an $n - 1$ Replikatverwalter durch $n - 1$ Unicasts zu ersetzen ist und sowohl das Senden als auch das Empfangen genau einen Schritt benötigen. Insgesamt sind $n * 2u * (n - 1)$ Schritte notwendig, also 18000 Schritte (und 9000 Nachrichten). Dabei wurde nicht berücksichtigt, daß der Multicast-Algorithmus eines häufigen Nachrichtenaustauschs für die Gruppenverwaltung (nicht verfügbare Replikatverwalter werden aus der Gruppe ausgeschlossen) Bedarf, damit ausgefallenen Replikatverwaltern bei erneuter Verfügbarkeit keine Schreiboperationen bekannt werden, wenn vorherige unbekannt sind (Konsistenzbeziehungen 4.7 und 4.8). Unser originärer Algorithmus benötigt 3624 Schritte bei $c = 4$ sowie 9080 Nachrichten und performiert bezüglich des hauptsächlichen Optimierungskriteriums, einer geringen Anzahl von Schritten, wesentlich besser. Auf Grund des signifikanten Unterschieds der Schrittzahl der beiden Algorithmen sei erwähnt: Der Multicast-Algorithmus performiert bei dieser Komplexitätsbetrachtung vor allem deshalb schlecht, weil in unserem Szenario ein echter Multicast weder im Modell noch auf Hardwareebene vorhanden ist. Die Unicasts werden hintereinander ausgeführt, und die Last wird nicht gleichmäßig über alle Netzknoten verteilt, so daß möglichst jeder Netzknoten zwischen zwei „ticks“ ausgelastet ist. Könnte ein echter Multicast genutzt werden, der auch implizit eine bessere Lastverteilung auf die Netzknoten zur Folge hätte, wäre die Anzahl der Schritte $2u * n$, also 2000 in diesem Beispiel. Dennoch würden auch hier die Nachteile erhalten bleiben, daß ein (hoher)

Kommunikationsaufwand für die Gruppenverwaltung erforderlich ist (dieser ist nicht in der Komplexitätsbetrachtung berücksichtigt), Recovery-Maßnahmen erforderlich sind und der Algorithmus nur für den Abgleich von stationären, nicht aber mobilen Replikatverwaltern eingesetzt werden kann.

Zusammenfassend läßt sich feststellen, daß unser eigener Algorithmus für die vorhandenen Anforderungen optimal geeignet ist. Er ist einfach zu realisieren und nutzt das Konzept von paarweisen Abgleichsitzungen, mit welchem sowohl stationäre als auch mobile Replikatverwalter bei hoher Flexibilität abgeglichen werden können. Recovery-Maßnahmen sind nicht erforderlich. Die Anforderung an einen Abgleichprozeß mit lediglich bestem Bemühen macht es möglich, mit einer geringen Anzahl von Schritten und damit einem geringen Zeitaufwand einen inkrementellen Abgleich durchzuführen.

Allerdings soll nicht verschwiegen werden, daß Fragen zum eigenen Algorithmus offen bleiben: Weder konnte die Korrektheit des Algorithmus selbst noch die Korrektheit der Abschätzung der Komplexität im Rahmen dieser Arbeit bewiesen werden. Beides wurde jedoch in einer Simulation für bis zu eintausend Replikatverwalter überprüft.

Abgleichsitzungen bei partieller Replikation

Im vorliegenden Anwendungsumfeld wird die partielle Replikation durch die Unterstützung mobiler Nutzer motiviert, die im allgemeinen nur beschränkte Ressourcen zur Verfügung haben. Damit werden neben komfortabler Offline-Bearbeitung auch neue Formen der Moderation erschlossen: Ein Moderator kann beispielsweise eine Diskussion auf sein Notebook replizieren und während einer Tagung oder Besprechung eine Sitzungsmoderation durchführen, deren Ergebnisse dann bei der nächsten Netzverbindung verbreitet werden. Diese Anwendungsfälle motivieren den Einsatz partieller Replikation für mobile Replikatverwalter, nicht jedoch für stationäre. Dennoch soll an einem Aspekt verdeutlicht werden, warum eine *uneingeschränkte* partielle Replikation von Bäumen, die sowohl für stationäre als auch mobile Replikatverwalter zur Verfügung steht, schwer in dem bisher vorgestellten Replikationskonzept zu realisieren ist.

In unserem Replikationskonzept war bisher ein primärer Replikatverwalter für die Festlegung einer systemweiten Bestätigungsreihenfolge ausreichend. In einem System mit partieller Replikation ist dies von Nachteil, weil der primäre Replikatverwalter jede Schreiboperation im System bestätigen muß, obwohl dieser unter Umständen selbst nur eine Teilmenge aller systemweiten vorhandenen Bäume repliziert. Daher ist es angebracht, wenn jeder Baum eines Waldes einen eigenen primären Replikatverwalter hat. Damit wird der Overhead, daß alle systemweiten Schreiboperationen von einem einzigen primären Replikatverwalter bestätigt werden, auf einzelne Bäume beschränkt. Daraus folgt der Bedarf an einer logischen Zeit mit eigenen Bestätigungsnummern pro Baum. Damit tritt jedoch eine Reihe von Problemen auf, deren Lösung nicht trivial ist: Mit der Verschiebe-Operation kann jeder Unterbaum in einen anderen Baum des Waldes verschoben werden. Wer ist der primäre Replikatverwalter, wenn die primären Replikatverwalter von Quell- und Zielknoten unterschiedlich sind? Wie einigen sich diese? Wie kann eine konsistente systemweite Bestätigungsreihenfolge garantiert werden? Wie werden die Bestätigungsnummern unterschiedlicher primärer Replikatverwalter zusammengeführt? Welche Problemlösungen gibt es, wenn sich zwei Verschiebe-Operationen widersprechen, deren Quell- und Zielpositionen zu drei unterschiedlichen primären Replikatverwaltern gehören? Darf ein primärer Replikatverwalter mobil sein?

Die genannten Fragen sollen an dem Beispiel in Abbildung 4.9 illustriert werden. Das Szenario besteht aus insgesamt fünf Replikatverwaltern, wobei von vieren die Replikatbestände graphisch dargestellt sind. R_5 ist ein mobiler Replikatverwalter, der den Unterbaum 12 replizieren möchte. Unter jedem Replikatverwalter ist dessen Kennzeichner dargestellt, über den Bäumen des Waldes wird der primäre Replikatverwalter für den jeweiligen Baum benannt. Die Replikatverwalter R_1 bis R_3 sind stationär, die anderen mobil. Die Beschriftung eines Knotens repräsentiert seinen Kennzeichner, nicht den Inhalt. Die Verschiebe-Operationen, die nebenläufig ausgeführt werden, sind gestrichelt inklusive ihrer Position in der späteren systemweiten Bestätigungsreihenfolge abgebildet.

1. Auf dem Replikatverwalter R_4 verschiebt ein Klient den Unterbaum 23 unter den Zielknoten 32. R_4 ist zu diesem Zeitpunkt nicht online. Wer ist der neue primäre Replikatverwalter für 23? Ist es schon R_4 , oder immer noch R_3 ?

Darf R_4 jetzt Klientenaufträge zum Schreiben für den Unterbaum 23 annehmen und bestätigen, wo R_3 noch nicht weiß, daß er nicht mehr der primäre Replikatverwalter ist und somit weiterhin Schreiboperationen für diesen Unterbaum bestätigt? Sollte R_4 eine solche Verschiebe-Operation nur erlauben, wenn dieser online ist und sich mit R_3 absprechen kann?

2. Auf R_2 wird der Unterbaum 22 unter den Knoten 17 verschoben, wodurch R_1 der neue primäre Replikatverwalter wird. Wie wird diese Information R_1 mitgeteilt und woher erfährt R_3 , daß er für diesen Unterbaum nicht mehr zuständig ist, wo die Verschiebe-Operation auf einem Replikatverwalter erzeugt und ausgeführt wurde, der weder für den einen, noch für den anderen Unterbaum der primäre Replikatverwalter ist? Ist es die Aufgabe von R_2 , zwischen R_3 und R_1 eine Vermittlerrolle zu spielen? Was würde geschehen, wenn R_1 ein mobiler Replikatverwalter wäre, der gerade offline ist? Bei der nächsten Netzverbindung von R_4 spricht dieser sich mit R_3 ab, daß er auf Grund der Verschiebe-Operation 1 der neue primäre Replikatverwalter für den Unterbaum 23 ist. Durch die Verschiebe-Operation 2 ist R_3 dafür aber nicht mehr zuständig. Bedeutet dies, daß sich R_3 für jeden Knoten den „ablösenden“ primären Replikatverwalter merken muß, um ihn später bei Bedarf einem anderen Replikatverwalter, in diesem Beispiel R_4 , mitteilen zu können?
3. Auf dem Replikatverwalter R_4 wird der Unterbaum 16 unter den Knoten 31 verschoben. Es gelten die bisher angesprochenen Probleme.
4. Der Replikatverwalter R_5 hat den Unterbaum 12 zur Replikation ausgewählt. Weil die Schreiboperationen eins bis vier R_1 noch unbekannt sind, weiß R_5 nichts von der starken Veränderung der Baumstruktur. Hätte ein Nutzer einen anderen Unterbaum zur Replikation ausgewählt, wenn er diesen Wissen besessen hätte? Würden für eine Realisierung der partiellen Replikation Verwaltungsstrukturen über den primären Replikatverwalter eines Baumes angelegt, so würden diese R_1 als primären Replikatverwalter für jeden Knoten des Unterbaumes 12 ausweisen. Welche Probleme ergeben sich dann aus der Tatsache, daß diese Zuordnung auf Basis eines längst überholten Wissens erfolgt?

Die Probleme zur Realisierung einer uneingeschränkten, partiellen Replikation sind sehr komplex. Mehrere Strategien scheinen zur Lösung der Probleme geeignet.

Zum einen kann die Funktionalität von Schreiboperationen so eingeschränkt werden, daß bestimmte Eigenschaften gegeben sind. Es könnte beispielsweise nur eine Verschiebung von Unterbäumen innerhalb des übergeordneten Baumes erlaubt sein, damit es keinen Wechsel der primären Replikatverwalter gibt. Die resultierenden Einschränkungen für die Funktionalität des Systems sind nach Meinung des Autors aber nicht tragbar. Eine andere Möglichkeit liegt in einer häufigeren Kommunikation der Replikatverwalter, die zusätzliches Wissen über Verwaltungsstrukturen austauschen, die zur Realisierung der partiellen Replikation genutzt werden.

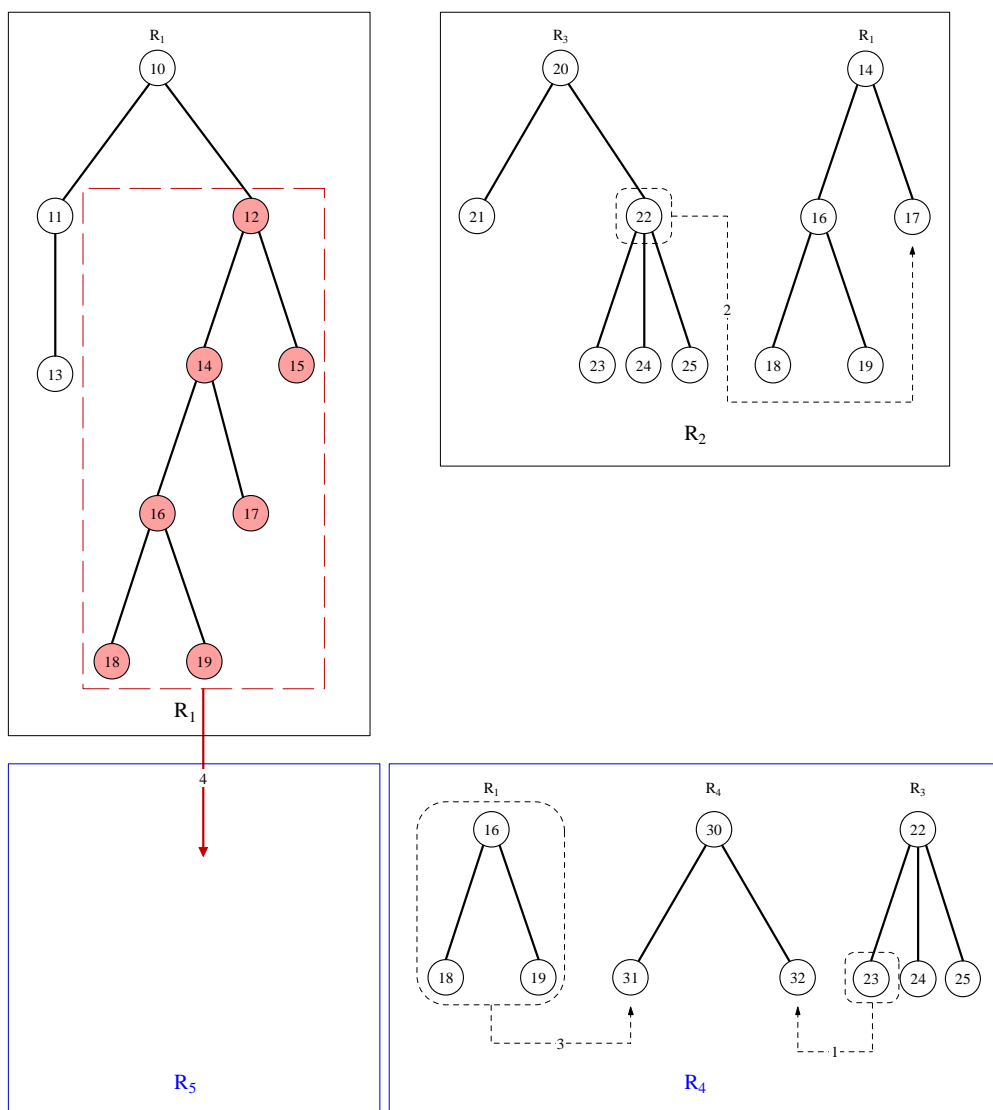


Abbildung 4.9: Verschiebe-Operationen bei uneingeschränkter, partieller Replikation

Im vorliegenden Anwendungsumfeld ist eine Motivation für eine uneingeschränkte, partielle Replikation nicht gegeben. Der Realisierungsaufwand ist sehr hoch und steht in keinem Verhältnis zum Nutzen. Die Daten im Projekt Demos haben selten einen ausschließlich lokalen Bezug und sollen von daher auf allen Replikativern repliziert sein, um vor allem eine maximale Verfügbarkeit des Dienstes zu gewährleisten. Die folgende Einschränkung führt damit zu einer optimal angepassten Lösung für unsere Anwendung in Form einer *eingeschränkten* partiellen Replikation: Die partielle Replikation wird ausschließlich für mobile Replikativern angeboten, nicht für stationäre (siehe Abbildung 4.10). Dabei darf ein mobiler Replikativern nur Sitzungen mit einem stationären Replikativern durchführen und diesem nur selbst erzeugte Operationen übermitteln. Diese Vereinfachung führt zu einem gravierenden Unterschied: In jeder Sitzung ist garantiert, daß ein Partner (der stationäre) vollständiges Wissen über alle Schreiboperationen bis zu dem Wissensstand hat, der durch seinen Annahmevektor und der höchsten bekannten Bestätigungsnummer repräsentiert ist. Damit können weiterhin die einzelnen Teile unseres Replikationskonzepts in der bisherigen Form genutzt werden und eine Unterstützung von partieller Replikation für mobile Replikativern kann einfach integriert werden. So gilt weiterhin das Konzept der logischen Zeiten wie bisher vorgestellt, so daß ein einziger primärer Replikativern für die Vergabe aller Bestätigungsnummern im System ausreichend ist.

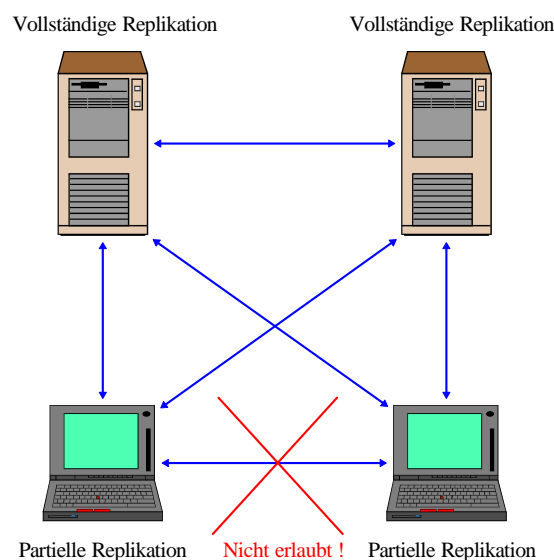


Abbildung 4.10: Eingeschränkte Auswahl an Partnern von mobilen Replikativern für Sitzungen

Ein mobiler Replikatorverwalter kann bei partieller Replikation beliebige Unterbäume aus dem Wald eines stationären Replikatorverwalters zu sich replizieren oder diese Replikate wieder bei sich entfernen (dereplizieren). Dabei sind folgende Einschränkungen sinnvoll:

- Es darf nur ein *vollständiger* Unterbaum, beginnend an seinem Wurzelknoten und endend an den Blättern, repliziert werden. Damit wird eine zu starke Partitionierung der Baumstruktur verhindert.
- Jeder Knoten darf nur einmal auf jedem mobilen Replikatorverwalter vorhanden sein. Die mehrfache Replikation eines Knotens erhöht weder die Performanz, noch die Verfügbarkeit. Dafür ist beim Replizieren eines Unterbaumes zu überprüfen, ob bereits einer seiner Knoten als Replikate auf dem mobilen Replikatorverwalter vorhanden ist.
- Es ist nur das Dereplizieren eines vollständigen Baumes aus dem Wald des mobilen Replikatorverwalters erlaubt. Damit kann ein replizierter Unterbaum nur als Ganzes derepliziert werden.

Mit diesen sinnvollen Einschränkungen wäre die Realisierung von Sitzungen mit partieller Replikation unproblematisch, wenn diese nicht durch die Auswirkungen von Verschiebe-Operationen erschwert würde: Ein stationärer Replikatorverwalter *ermittelt* während einer Sitzung für einen mobilen Replikatorverwalter jene Schreiboperationen, die sich auf dessen Replikatebestand beziehen. Dies funktioniert für Erstelle-, Ändere- und Lösche-Operationen, weil deren Bezugsknoten entweder zu dem Replikatebestand des mobilen Replikatorverwalters gehören und daher übermittelt werden oder nicht. Anders sieht es hingegen bei Verschiebe-Operationen aus, wie es in einem Beispiel in Abbildung 4.11 dargestellt ist: Für einen mobilen Replikatorverwalter scheinen Unterbäume neu erstellt oder gelöscht, obwohl sie lediglich verschoben wurden und dem mobilen Replikatorverwalter auf Grund der partiellen Replikation unbekannt sind. Daher sollen zwei *lokale* Schreiboperationen für mobile Replikatorverwalter eingeführt werden, die dieses Problem lösen: Eine Schreiboperation (ErstelleLokal(knotenListe)) erstellt einen vollständigen lokalen Unterbaum auf einem mobilen Replikatorverwalter, so daß ein verschobener Unterbaum, dessen Knoten außerhalb des Replikatebestands des mobilen Replikatorverwalters liegen, lokal neu erzeugt wird. Die andere Schreiboperation (LöscheLokal(knotenID)) löscht

lokal einen Unterbaum, der durch eine Verschiebe-Operation unter einen Zielknoten verschoben wurde, der außerhalb des Replikabestands des mobilen Replikativverwalters liegt. Für das dargestellte Beispiel würde der stationäre Replikativverwalter die Verschiebe-Operationen folgendermaßen in lokale Schreiboperationen für den mobilen Partner übersetzen: Aus Verschiebe(15, 21) wird LöscheLokal(15), aus Verschiebe(22,17) wird ErstelleLokal(22, Attribute von 22, . . . , 25, Attribute von 25).

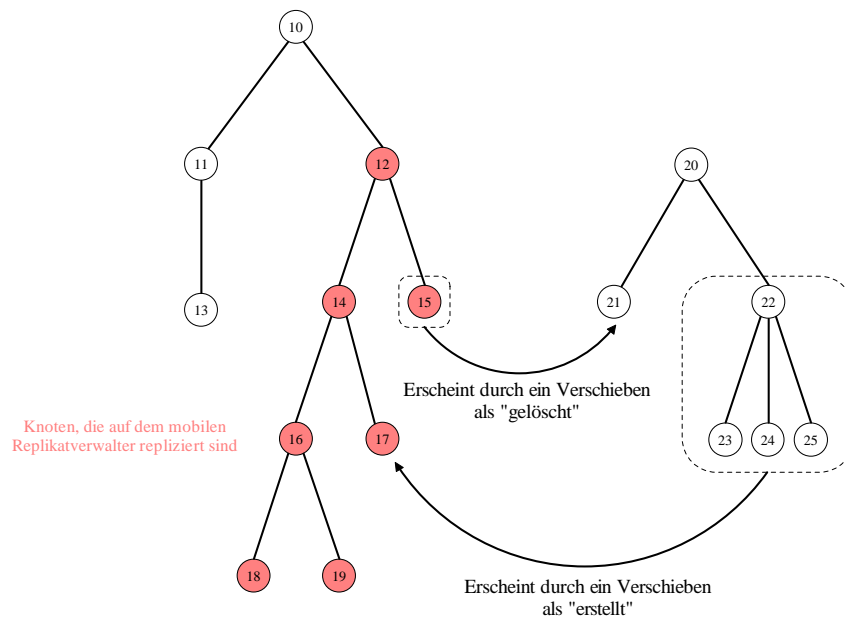


Abbildung 4.11: Verschiebe-Operationen aus Sicht eines mobilen Replikativverwalters mit partieller Replikation

Damit wird eine Sitzung zwischen einem mobilen Replikativverwalter R_m mit partieller Replikation und einem stationären Replikativverwalter R_s wie folgt durchgeführt:

1. R_m erfragt den Wissensstand von R_s über jene Schreiboperationen, die R_s von R_m kennt.
2. R_m übermittelt R_s jene Schreiboperationen, die R_m erzeugt hat und die R_s unbekannt sind.
3. R_s erfragt den Wissensstand von R_m . Dabei übermittelt R_m die Kennzeichner der Wurzelknoten aller Unterbäume, die dieser repliziert hat.

4. R_s ermittelt alle Schreiboperationen und Bestätigungen, die sich auf die von R_m replizierten Unterbäume beziehen und diesem unbekannt sind. Dabei werden Verschiebe-Operationen in lokale Schreiboperationen für R_m umgesetzt.
5. R_s übermittelt die ermittelten Schreiboperationen an R_m .

Unsere partielle Replikation nutzt die (rekursive) Struktur von Bäumen, die bisher lediglich den Nachteil einer aufwendigeren Konfliktbehandlung hatte, als Vorteil aus. Dies wird durch die Einschränkung möglich, daß ein mobiler Replikatverwalter Unterbäume ausschließlich vollständig replizieren darf. Wenn ein mobiler Replikatverwalter in einer Abgleichsitzung seinem Partner die notwendigen Informationen über seinen Replikatbestand übermitteln soll, so muß dieser lediglich die Kennzeichner der Wurzelknoten der replizierten Unterbäume übertragen und nicht, wie im Fall einer unstrukturierten Menge, die Kennzeichner aller einzelner Knoten. Der stationäre Replikatverwalter kann damit für seinen Partner ermitteln, welche einzelnen Knoten zu diesem Unterbaum gehören und folglich die Schreiboperationen bestimmen, die sich auf diesen beziehen.

Mobile Replikatverwalter werden nicht in den Abgleichzyklen und der Abgleichpolitik berücksichtigt, damit sie auf Grund ihrer beschränkten Ressourcen selbst entscheiden können, wann sie eine Sitzung initiieren. Als Partner einer Sitzung kommt jeder stationäre Replikatverwalter in Frage, obwohl der primäre zu bevorzugen ist, weil dann einem mobilen Replikatverwalter sofort die Bestätigungen seiner bis dahin unbestätigten Schreiboperationen mitgeteilt werden können.

Zusammenfassend sollen die Vor- und Nachteile der vorgestellten Lösung aufgeführt werden:

- + Die Lösung ist einfach.
- + Ein primärer Replikatverwalter im System ist ausreichend. Die Vergabe der Bestätigungsnummern bleibt von den Erweiterungen der partiellen Replikation unberührt.
- + Gute Integration in das vorgestellte Replikationskonzept.
- + Verwendung der Struktur von Bäumen zur Reduktion des Kommunikationsaufwands.

- + Keine Einschränkungen der Funktionalität des Systems für die Realisierung einer partiellen Replikation.
- + Unterstützung aller Anforderungen des Anwendungsumfelds.
- Keine partielle Replikation für stationäre Replikatverwalter.
- Kein Abgleich von mobilen Replikatverwaltern untereinander.

4.4 Fehler

Ein Grund für den Einsatz von Replikation im Projekt Demos ist die Erhöhung der Verfügbarkeit des zu erbringenden Dienstes. Um die Güte der Umsetzung dieser Anforderung zu beurteilen, ist ein Fehlermodell erforderlich, welches das Systemverhalten in Fehlersituationen beschreibt. In unserem Fehlermodell werden ausschließlich stationäre Replikatverwalter betrachtet, ohne Klienten und mobile Replikatverwalter. Jeder stationäre Replikatverwalter ist mit jedem anderen verbunden. In dem System können zwei verschiedene Fehlerklassen auftreten: Fail-Stop-Fehler und Byzantinische Fehler, die jeweils bei Netzknoten und Kommunikationsverbindungen auftreten können. Dabei wird angenommen, daß keine Korrelation von Fehlern von unterschiedlichen Netzknoten und unterschiedlichen Kommunikationsverbindungen besteht. Die Definition der Fehlerklassen wurde bereits in Abschnitt 2.1 vorgenommen.

Die Erkennung und Behandlung von Fail-Stop-Fehlern steht im Zentrum dieses Fehlermodells. Bei den vorliegenden physischen Gegebenheiten können, wie bei den meisten *Real-World*-Systemen, keine garantierte maximale Reaktionszeit eines Prozesses und keine garantierte maximale Nachrichtenlaufzeit des Kommunikationssystems bestimmt werden. Somit liegt kein synchrones verteiltes System vor, in dem eine Fehlererkennung einfach zu realisieren wäre. Auf der anderen Seite können unsere Abgleichalgorithmen nicht in einem asynchronen Systemmodell klassischer Definition betrachtet werden, weil ohne einen Bezug auf die oben genannten Eigenschaften eine Erkennung von Fail-Stop-Fehlern nicht möglich ist. Aus diesem Grund sind asynchrone Systemmodelle in der Praxis oft nicht verwendbar (man denke an den „Klassiker“ [FLP85]). Neben diesen beiden Extremen, die insbesondere für eine

theoretische Betrachtung verteilter Systeme bedeutsam sind, gibt es das Modell der partiell synchronen Systeme, welches real existierende Systeme treffender beschreibt [Lyn96] (auch als zeitabhängiges asynchrones System in [CS95] bezeichnet). Hier wird die Annahme getroffen, daß die Festlegung einer maximalen Reaktions- und Nachrichtenlaufzeit („*Timeouts*“ in der Welt der Programmierer) derart möglich ist, daß diese während der meisten Betriebszeit gelten und damit real existierende Fail-Stop-Fehler erkannt werden können. Diese Erkennung ist jedoch nicht verläßlich, weil auch scheinbare Fail-Stop-Fehler ausgewiesen werden, obwohl ein Netzknoten oder eine Kommunikationsverbindung funktioniert und eine Antwort lediglich stark verzögert ist. Die Erkennung scheinbarer Fail-Stop-Fehler stellt kein Problem für unser Replikationskonzept dar, weil der Abgleichprozeß nach bestem Bemühen erfolgt und keine sich widersprechenden Ergebnisse auftreten können. Wir setzen für eine geeignete Erkennung von Fail-Stop-Fehlern voraus, daß die Werte für die maximale Reaktionszeit eines Prozesses und für die maximale Nachrichtenlaufzeit so gewählt werden, daß die Wahrscheinlichkeit des Auftretens scheinbarer Fail-Stop-Fehler gering ist und eine Ausfallerkennung in angemessener Zeit erfolgt.

Fehler von Kommunikationsverbindungen

Für einen korrekten Ablauf der Abgleichsitzungen ist es erforderlich, daß das verwendete Kommunikationssystem alle Nachrichten in Sendereihenfolge ausliefert. Byzantinische Fehler von Kommunikationsverbindungen werden auf Transportebene des Netzwerkprotokolls behandelt. Dies bedeutet insbesondere, daß aus Sicht unseres Replikationskonzepts keine Nachrichten verändert und verdoppelt werden, sowie verloren gehen. Das Internet erfüllt mit dem verläßlichen *TCP*-Protokoll diese Anforderungen und behandelt Byzantinische Fehler von Kommunikationsverbindungen bis zu einem gewissen, in der Praxis ausreichenden Grad. Weitere Aspekte von Byzantinischen Fehlern von Kommunikationsverbindungen werden im Rahmen dieser Arbeit nicht behandelt.

Fail-Stop-Fehler von Kommunikationsverbindungen führen häufig zu einer Netzpartitionierung. Einige Replikationskonzepte, wie beispielsweise das Primary-Copy-ROWA-Replikationskonzept, sind nicht für diese Fehlersituationen geeignet. Ein System mit einem solchen Replikationskonzept ist beispielsweise nur verfügbar, wenn ein primärer Replikatverwalter für die Annahme von Schreibaufträgen der Klienten

verfügbar ist. Scheint dieser auszufallen, wird ein neuer primärer Replikatverwalter im System gewählt. Im Fall einer Netzpartitionierung können daher mehrere primäre Replikatverwalter in unterschiedlichen Netzpartitionen entstehen, so daß nebenläufig ausgeführte Schreiboperationen die Konsistenz des Systems verletzen können. Replikationskonzepte mit Quorumalgorithmen arbeiten hingegen auch bei einer Netzpartitionierung korrekt.

In unserem Replikationskonzept wird ausschließlich eine paarweise Kommunikation in Form von Abgleichsitzungen verwendet. Eine systemweit koordinierte Kommunikation der Replikatverwalter, wie beispielweise bei der Wahl eines primären Replikatverwalters im Primary-Copy-ROWA-Replikationskonzept, ist nicht notwendig. Damit reduziert sich das Problem auf eine mögliche oder unmögliche paarweise Kommunikation zu einem Zeitpunkt, in der eine Netzpartitionierung jedoch ohne Bedeutung ist.

Im entwickelten Replikationskonzept sind Fail-Stop-Fehler von Kommunikationsverbindungen unter zwei Gesichtspunkten zu betrachten:

- Ausfall während einer Abgleichsitzung. Wird die Verbindung zwischen zwei Replikatverwaltern unterbrochen und damit eine laufende Sitzung, wird versucht, eine neue Sitzung mit diesem Partner in derselben Runde zu initiieren. Eine (unfreiwillig) abgebrochene Sitzung erfordert keine Recovery-Maßnahmen, weil zu keinem Zeitpunkt eines Abbruchs Inkonsistenzen entstehen können. Dies wird durch eine intelligente Reihenfolge für die Übermittlung der Schreiboperationen und deren Bestätigungen unter Berücksichtigung der potentiell kausalen Beziehungen erreicht, wie es in den Konsistenzbeziehungen spezifiziert ist. Nach einem Sitzungsabbruch bleibt ein konsistenter Replikatbestand zurück, der lediglich das Wissen einer jüngeren logischen Zeit im Vergleich zum Wissen nach einer vollständig durchgeführten Sitzung widerspiegelt. Eine neue Sitzung kann also wie im fehlerfreien Fall ohne eine Recovery-Maßnahme durchgeführt werden. Falls eine neue Sitzung mit dem alten Partner nicht initiiert werden kann, wird zur nächsten Runde übergegangen.
- Partnerwahl sowie Wissensverbreitung in einem Abgleichzyklus bei Netzpartitionierung. Eine vollständige Wissensverbreitung mit einem geringen Zeitaufwand und einer geringen Anzahl von Runden wird durch den in Abschnitt

4.3 vorgestellten Algorithmus zur Partnerwahl erreicht, der jedoch für den fehlerfreien Fall optimiert ist. Hier soll ein einfacher Algorithmus vorgestellt werden, der die Auswahl der Partner im Fehlerfall übernimmt. Versucht ein Replikatverwalter mehrmals erfolglos, seinen Partner bei Beginn einer neuen Runde zu kontaktieren, so schaltet dieser in den *Fehlermodus*: Die Anzahl der noch ausstehenden Runden des aktuellen Zyklus wird um einen vorher festgelegten Faktor erhöht, weil für einen Wissensaustausch im Fehlerfall eine höhere Rundenanzahl notwendig ist als im fehlerfreien Fall bei der Verwendung des optimalen Algorithmus. Der Replikatverwalter wählt *zufällig* einen anderen Replikatverwalter aus und initiiert eine Sitzung mit diesem, falls der Partner dazu bereit ist. Danach wird zur nächsten Runde übergegangen. Findet der Replikatverwalter nach einer Anzahl festgesetzter Versuche für eine Runde keinen Partner, so wird ebenfalls zur nächsten Runde übergegangen. Sind alle Runden durchlaufen, wird der Zyklus als abgeschlossen betrachtet. Die Robustheit gegenüber einer Netzpartitionierung wird durch die Zufallskomponente erreicht.

Existieren $n > 1$ Replikatverwalter, so sind diese durch $\sum_{i=1}^{n-1} i$ Kommunikationsverbindungen miteinander verbunden. Selbst wenn jede dieser Kommunikationsverbindungen ausfällt, bleibt das System so lange verfügbar, wie ein Klient wenigstens einen Replikatverwalter kontaktieren kann. Lediglich die Wissensverbreitung über neue Schreiboperationen und deren Bestätigungen ist bei vollständigem Ausfall aller Kommunikationsverbindungen zwischen den Replikatverwaltern unterbunden. Damit steigt die Wahrscheinlichkeit für eine größere Anzahl von Konflikten in den nebenläufigen angenommenen Klientenaufträgen, die jedoch später in der Konfliktbehandlung aufgelöst werden. Ein inkonsistenter Zustand kann selbst in dieser Extremsituation nicht eintreten.

Fehler von Netzknoten

Die Berücksichtigung von Byzantinischen Fehlern von Netzknoten liegt auf Grund der hohen Schwierigkeit und der hohen Anzahl offener Fragen dieses Themenbereichs außerhalb des Rahmens dieser Arbeit. Aus diesem Grund werden ausschließlich Fail-Stop-Fehler von Netzknoten diskutiert und unter drei Aspekten betrachtet:

- Ausfall während einer Abgleichsitzung. Fällt ein Netzknoten während einer laufenden Sitzung aus, so muß gewährleistet sein, daß die Atomarität der Entgegennahme einer Schreiboperation mit einer eventuellen atomaren Ausführung und Änderung des Wissensstands, repräsentiert durch den Annahmevektor und die höchste bekannte Bestätigungsnummer, gewährleistet ist. Es darf beispielsweise nicht möglich sein, daß eine Schreiboperation von einem Partner entgegengenommen und persistent im Schreiblog festgehalten wird, dieses Wissen aber nicht korrekt im Annahmevektor repräsentiert wird, weil der Replikatverwalter vor der Änderung des Annahmevektors ausgefallen ist. Zur Gewährleistung der Atomarität gibt es einige Techniken, wie beispielsweise *Intention Lists* [Lam81], *Shadow Paging* [GMB⁺81] oder *Write-ahead Logging* [PS83], die hier nicht näher erläutert werden. Zur Realisierung der Atomarität können auch bereits verfügbare Bibliotheken, wie beispielsweise *Arjuna* [PSWL95], verwendet werden. In der Realisierung unseres Replikationskonzepts wird die Existenz einer Komponente vorausgesetzt, welche die Atomarität von Befehlsserien auf dem Rechnersystem eines Replikatverwalters mit Hilfe der oben genannten Techniken garantiert.
- Ausfall während eines Abgleichzyklus. Fällt ein Replikatverwalter während eines laufenden Zyklus aus und wird innerhalb dieses wieder betriebsbereit, so nimmt er nicht mehr an dem laufenden Zyklus teil. Fällt ein Replikatverwalter während eines Zyklus aus und bleibt innerhalb dieses ausgefallen und somit für keinen potentiellen Partner ansprechbar, so gilt das bisher gesagte für die Auswahl von Partnern bei Fehlern von Kommunikationsverbindungen. Ein Replikatverwalter kann nicht unterscheiden, ob sein potentieller Partner durch einen Fehler der Kommunikationsverbindung oder durch einen Fehler des Netzknotens nicht ansprechbar ist, so daß die gleiche Fehlerbehandlung verwendet wird.
- Ausfall des primären Replikatverwalters. In vielen Replikationskonzepten mit einem ausgezeichneten Replikatverwalter (beispielsweise Primary-Copy-ROWA) führt der Ausfall des primären Replikatverwalters zu einer Rekonfiguration des Systems, damit weiterhin der Dienst erbracht werden kann. Die Existenz eines funktionstüchtigen primären Replikatverwalters ist zwingend

für die Verfügbarkeit des Systems erforderlich. In unserem Replikationskonzept ist dies nicht der Fall: Der Ausfall des primären Replikatverwalters hat lediglich die Auswirkung, daß die neu erzeugten Schreiboperationen vorläufig unbestätigt bleiben. Damit erhöht sich die Anzahl potentieller Konflikte, weil die nebenläufig erzeugten Schreiboperationen nicht in der üblichen Zeitspanne von dem primären Replikatverwalter serialisiert werden. Dies führt aus Sicht der Konsistenz zu keinem Problem, weil die Konfliktbehandlung eine beliebige Anzahl von Konflikten auflösen kann. Aus diesem Grund ist keine sofortige Bestimmung eines neuen primären Replikatverwalters, eventuell mit zusätzlichen Recovery-Maßnahmen, für die Erhaltung der Verfügbarkeit des Systems erforderlich. Das System arbeitet weiter wie im Konsistenzmodell beschrieben, so daß die Nutzer lediglich mit häufigeren Konflikten zwischen ihren nebenläufigen Schreiboperationen sowie einer *längeren Bestätigungszeit* rechnen müssen. Ein Administrator kann beispielsweise den ausgefallenen primären Replikatverwalter am Tag nach dem Ausfall ersetzen oder reparieren. Ein Ausfall wird also manuell behoben.

Mit steigender Anzahl von Ausfällen wird die Funktionalität des Systems (hauptsächlich in Form der Performanz) nur allmählich verringert. Weil jeder Replikatverwalter autonom den Dienst erbringen kann, bleibt die Verfügbarkeit des Systems selbst bei $n - 1$ Netzknotenausfällen erhalten (graceful degradation).

Zusammenfassend läßt sich die Schlußfolgerung ziehen, daß unser Replikationskonzept eine hohe Verfügbarkeit auch bei einer hohen Anzahl von Fail-Stop-Fehlern von Kommunikationsverbindungen und Netzknoten bietet. Der Ausfall des primären Replikatverwalters hat keine Auswirkungen auf die Verfügbarkeit und auch im Fall einer Netzpartitionierung sind Inkonsistenzen ausgeschlossen.

4.5 Mobilität

Die Unterstützung von mobilen Nutzern beeinflußt den Entwurf eines Replikationskonzepts erheblich. Die Motivation für die Unterstützung dieser Zielgruppe wurde bereits ausführlich besprochen. Damit ein mobiler Nutzer offline arbeiten kann, wurde das Konzept des mobilen Replikatverwalters eingeführt, der im Regelfall die folgenden Eigenschaften aufweist:

- Verfügt lediglich über eine geringe Menge an Massenspeicherplatz und eine über längere Zeiträume unsichere Kommunikationsverbindung mit geringer Bandbreite.
- Wechselt zwischen Online- und Offline-Phasen, in welchen der Replikatverwalter mit dem Internet verbunden ist oder nicht.
- Ändert seine geographische Position.
- Bedient nur einen Klienten, der die gleiche Hardware wie der Replikatverwalter verwendet. Dennoch ist es möglich, daß ein mobiler Replikatverwalter auch mehrere Klienten, beispielsweise als lokaler Server in einem LAN, bedient.

In diesem Abschnitt soll erläutert werden, wie Einträge aus dem Schreiblog auch bei einer Unterstützung von mobilen Replikatverwaltern entfernt werden können und wie Replikatverwalter leichtgewichtig erzeugt werden können. Einige wesentliche Konzepte zur Unterstützung mobiler Nutzer sind bereits in vorherigen Abschnitten erläutert worden:

- Die Unterstützung von partieller Replikation wird in Abschnitt 4.3 ab Seite 75 behandelt.
- Die Garantien für Klientensitzungen werden in Abschnitt 4.2 ab Seite 52 dargestellt.

Entfernen von Einträgen aus dem Schreiblog

Es wurde bisher die Annahme getroffen, daß jede Schreiboperation in dem Schreiblog eines Replikatverwalters dauerhaft gespeichert ist und daraus nicht wieder entfernt wird. Selbst bei geringen Kosten für Massenspeicher ist es jedoch nicht einzusehen, warum eine Vielzahl von Einträgen des Schreiblogs von bestätigten Schreiboperationen nicht entfernt wird, wenn diese Schreiboperationen bereits allen Replikatverwaltern bekannt sind und in Abgleichsitzungen nicht wieder benötigt werden. Stationäre Replikatverwalter weisen in der Regel ähnliche Replikatbestände auf, weil sie regelmäßig Abgleichsitzungen durchführen und daher die Wahrscheinlichkeit gering ist, daß nach einem längeren Zeitraum (zum Beispiel einem Monat)

eine Schreiboperation nicht allen stationären Replikalverwaltern bekannt ist. Dies gilt jedoch nicht für mobile Replikalverwalter, die unter Umständen lange Zeiträume vom Netz getrennt sind. Dennoch ist es nicht sinnvoll, daß die stationären Replikalverwalter jede Schreiboperation für eine mögliche inkrementelle Abgleichsitzung mit einem mobilen Replikalverwalter aufbewahren, der vielleicht nicht mehr existiert und von seinem Nutzer nicht ordnungsgemäß abgemeldet wurde.

Daher sollten die Replikalverwalter in regelmäßigen Abständen (beispielsweise alle paar Monate) ältere Schreiboperationen entfernen, die mit sehr hoher Wahrscheinlichkeit allen Replikalverwaltern bekannt sind. Dadurch entsteht jedoch die (geringe) Gefahr, daß eine spätere inkrementelle Abgleichsitzung mit einem Partner nicht mehr möglich ist, weil eine bereits entfernte Schreiboperation in der Sitzung für den Abgleich des Partners benötigt wird. In einem solchen Fall ist ein *vollständiger Transfer* des Replikalbestands mit dem Schreiblog notwendig, um den Partner auf einen aktuellen Stand zu bringen. Es existiert somit ein Tradeoff zwischen den Kosten des Massenspeicherbedarfs auf einem Replikalverwalter und den Kommunikationskosten, die durch einen Partner entstehen können, dessen Replikalbestand zu stark von dem eigenen abweicht. Aus diesem Grund sollte ein Replikalverwalter mit Bedacht und nur gelegentlich alte Einträge aus dem Schreiblog löschen. Im Regelfall kann davon ausgegangen werden, daß niemals ein vollständiger Transfer zwischen zwei stationären Replikalverwaltern notwendig ist. Lediglich bei mobilen Replikalverwaltern, die Monate vom Netz getrennt waren, könnte ein solcher vollständiger Transfer erforderlich sein. Die Feststellung, ob ein vollständiger Transfer notwendig ist oder nicht, kann durch einen einfachen Vergleich der höchsten bekannten Bestätigungsnummer des Partners mit der höchsten Bestätigungsnummer der entfernten Operationen getroffen werden.

Erzeugen von Replikalverwaltern

Für die Unterstützung mobiler Nutzer ist es erforderlich, daß mobile Replikalverwalter dynamisch in einer leichtgewichtigen Weise im System erzeugt und entfernt werden können, damit eine flexible Verwendung erfolgen kann. Bei üblichen Replikationskonzepten, die insbesondere ausschließlich stationäre Replikalverwalter unterstützen, welche selten erzeugt oder entfernt werden, fehlt eine solche Möglichkeit

und ein Systemadministrator muß den Erzeugungsvorgang durchführen. Nachfolgend wird beschrieben, wie in unserem Replikationskonzept mobile und stationäre Replikatverwalter leichtgewichtig erzeugt werden.

Jeder Replikatverwalter wird mit einem *systemweit eindeutigen* Kennzeichner identifiziert. Der erste Replikatverwalter des Systems hat den Kennzeichner 0. Für die Erzeugung eines neuen Replikatverwalters R_{neu} schickt dieser einen Auftrag an einen bereits existierenden, stationären Replikatverwalter (R_{alt}), der den Erzeugungsvorgang steuert. R_{alt} erstellt zuerst einen systemweit eindeutigen Kennzeichner für R_{neu} , der aus der Annahmenummer des Erzeugungsauftrags besteht. In der Annahmenummer ist auch der Kennzeichner von R_{alt} enthalten, so daß für jeden erzeugten Replikatverwalter dessen „Ahnenreihe“ rekursiv bestimmt werden kann. Die Erzeugung von R_{neu} wird wie eine gewöhnliche Schreiboperation in das Schreiblog von R_{alt} eingetragen und später in Abgleichsitzungen übertragen, damit das Wissen über die Existenz des neuen Replikatverwalters im System verbreitet wird. Nach dem Eintragen wird dem Replikatverwalter R_{neu} der Replikatbestand und das Schreiblog von R_{alt} übermittelt. Falls der zu erzeugende Replikatverwalter ein mobiler ist, werden nur die zu replizierenden Unterbäume mit deren Schreiboperationen übermittelt. Im Fall eines neuen stationären Replikatverwalters wird der vollständige Replikatbestand und das vollständige Schreiblog übermittelt. Anschließend wird R_{neu} die logische Zeit zur Bildung von Annahmenummern durch R_{alt} zugewiesen. R_{neu} ist jetzt einsatzbereit.

Das Entfernen eines Replikatverwalters wird analog zum Erzeugen durchgeführt, wobei dem im System verbleibenden Replikatverwalter die noch unbekanntenen Schreiboperationen des zu entfernenden Replikatverwalters zu übermitteln sind.

4.6 Nebenläufigkeit

Für eine hohe Performanz, charakterisiert durch geringe Antwortzeiten und hohen Durchsatz, ist eine nebenläufige Bearbeitung unterschiedlicher Aufgaben auf jedem Replikatverwalter erforderlich. In diesem Abschnitt soll erläutert werden, welche Typen von Aufgaben ein stationärer Replikatverwalter in unserem Replikationskonzept nebenläufig bearbeiten kann, um eine hohe Performanz zu realisieren.

- **Bearbeitung von Klientenaufträgen.** Die Hauptaufgabe eines stationären Replikatverwalters liegt in der Erbringung des Dienstes für die Klienten. Durch eine effiziente Realisierung des Replikationskonzepts kann eine Vielzahl von Klienten nebenläufig bedient werden.
- **Abgleichsitzungen mit mobilen Replikatverwaltern.** Mobile Replikatverwalter bestimmen selbstständig, wann sie eine Abgleichsitzung mit einem stationären Replikatverwalter durchführen. Auf Grund der hohen Anzahl von mobilen Replikatverwaltern im System, die insbesondere zu bestimmten Tageszeiten einen unabhängigen Bedarf an Abgleichsitzungen haben (beispielsweise in den Abendstunden, wenn mobile Nutzer den letzten Diskussionsstand im Büro für die Offline-Arbeit zu Hause abgleichen), ist es notwendig, daß ein stationärer Replikatverwalter eine Vielzahl von Abgleichsitzungen mit mobilen Replikatverwaltern nebenläufig durchführen kann. Zusätzlich sind auch die Klientenaufträge zu bedienen, damit die Verfügbarkeit des Systems nicht reduziert wird.
- **Abgleichsitzungen mit stationären Replikatverwaltern.** Die Häufigkeit von Abgleichzyklen zwischen stationären Replikatverwaltern bestimmt, wie hoch der Grad an Übereinstimmung der Replikatbestände im System ist. Daher wurde in dem Replikationskonzept und seiner Realisierung darauf geachtet, daß ein stationärer Replikatverwalter mit einem stationären Partner eine Abgleichsitzung durchführen kann, während nebenläufig Klientenaufträge bearbeitet und Abgleichsitzungen mit einer Vielzahl von mobilen Replikatverwaltern durchgeführt werden. Daher können die stationären Replikatverwalter im System bei uneingeschränkter Verfügbarkeit des Dienstes durch häufige Abgleichzyklen eine hohe Übereinstimmung der Replikatbestände erzielen. Dabei reicht es aus, wenn ein stationärer Replikatverwalter lediglich mit einem einzigen stationären Partner eine Abgleichsitzung gleichzeitig durchführen kann, weil nur eine einzige Abgleichsitzung von einem stationären Replikatverwalter in einer Abgleichrunde vorgesehen ist.

In Abbildung 4.12 ist exemplarisch dargestellt, welche Aufgaben ein Replikatverwalter nebenläufig bearbeiten kann. Die Nebenläufigkeit einzelner Prozesse der Realisierung, die auf der Nebenläufigkeit der zu bearbeitenden Aufgaben auf Modellebene basiert, ist in Abschnitt 5.4 beschrieben.

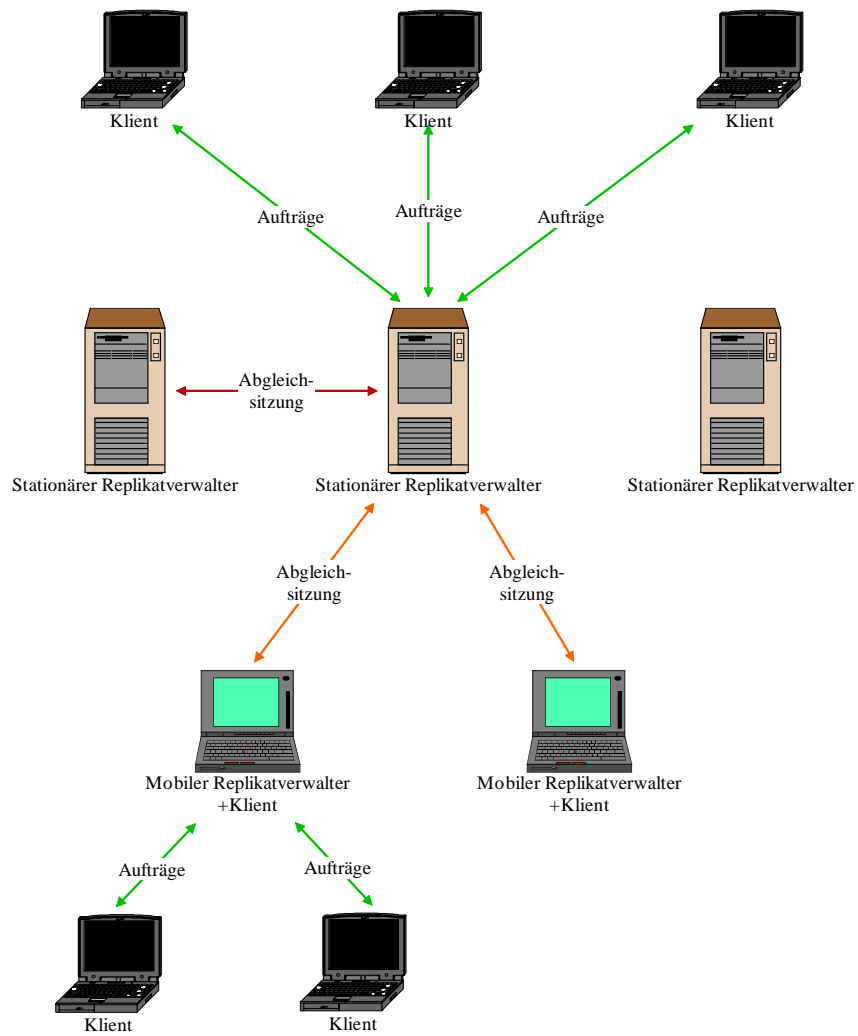


Abbildung 4.12: Nebenläufige Bearbeitung von Aufgaben

Kapitel 5

Realisierung

Inhalt

5.1 Darstellungsmittel	96
5.2 System	98
5.3 Replikatverwalter	100
5.4 Nebenläufigkeit und der Verteilungsprozeß	106
5.5 Prozesse zur Realisierung von Abgleichsitzungen	110

Für eine Realisierung des entwickelten Replikationskonzepts wurde im Rahmen der Diplomarbeit ein implementierungsnahes SDL-Modell in einem Softwareentwicklungswerkzeug entworfen, welches das System und die entwickelten Algorithmen inklusive ihrer Nebenläufigkeiten beschreibt. In diesem Kapitel soll dieses SDL-Modell keineswegs vollständig, sondern lediglich in Ausschnitten dargestellt werden. Dabei wurden solche Teile zur Darstellung ausgewählt, die zum einen wichtig und zum anderen schwierig zu realisieren sind. Dem Leser soll in diesem Kapitel ein Eindruck gegeben werden, welche Komponenten ein Replikatverwalter benötigt, wie diese interagieren, welche Nebenläufigkeiten existieren und wie ausgewählte Aspekte algorithmisch realisiert werden.

5.1 Darstellungsmittel

Zur Realisierung des Replikationskonzepts mit seinen Algorithmen wird in dieser Arbeit die Spezifikationssprache SDL verwendet. Diese wurde standardisiert von *International Telegraph and Telephone Consultative Committee* (CCITT) und eignet sich besonders für die Darstellung von ereignisgesteuerten Systemen. SDL basiert auf endlichen Automaten, genauer gesagt, erweiterten endlichen Automaten. Ein endlicher Automat verfügt über eine endliche Menge von Zuständen, die er einnehmen, einer endlichen Menge von Eingangssignalen, die er bearbeiten, und einer endlichen Menge von Ausgangssignalen, die er erzeugen kann. Die Zustandsübergänge werden durch Eingabesignale ausgelöst, Ausgangssignale nur während der Zustandsübergänge erzeugt. Im Gegensatz zum herkömmlichen endlichen Automaten verfügt der erweiterte endliche Automat auch über Variablen, die einen unendlichen Wertebereich haben können. Zustandsübergänge können damit nicht nur von dem aktuellen Eingangssignal, sondern auch von Bedingungen und Variablen abhängig gemacht werden. Während eines Zustandsüberganges können Variablen ihre Werte ändern [CCI88].

Im folgenden soll für die Darstellung der Sprache SDL ihre graphische Syntax (SDL/GR) verwandt werden. Es sei angemerkt, daß unsere Darstellung im Sinne von SDL nicht vollständig oder Standard konform ist. Eigene Anpassungen erlauben es, die Algorithmen übersichtlicher und bequemer darzustellen.

Die wichtigsten Symbole von SDL sind in den Abbildungen 5.1, 5.2, 5.3 dargestellt. Es gibt in SDL vier verschiedene Ebenen, nämlich *Systeme*, *Blöcke*, *Prozesse* und *Prozeduren*. Systeme, Blöcke und Prozesse kommunizieren über *Kanäle*, über die *Nachrichten* verschickt werden. Jeder *Prozeß* und jede *Prozedur* besitzt ein *Startsymbol*, bei dem die Ausführung beginnt, sowie ein *Endsymbol*, nach welchem der Prozeß oder die Prozedur beendet ist. In einem Zustand wird ein Zustandsübergang durch ein *Eingabesignal* ausgelöst, der *Tasks*, *Entscheidungen*, *Prozeduraufrufe*, *Prozeßerzeugungen mit Prozeßstart* oder *Signalausgaben* enthalten kann, bevor der Automat in einen neuen Zustand gelangt. Ein *Konnektor* verbindet zwei Diagramme miteinander, die aus Platzmangel getrennt werden mußten. Die Spezifikation der Anzahl von Instanzen eines Prozesses nach dem Systemstart sowie die maximale Instanzenanzahl erfolgt in einem Prozeßdiagramm in der rechten oberen Ecke, wobei die maximale Instanzenanzahl in Klammern angegeben wird.

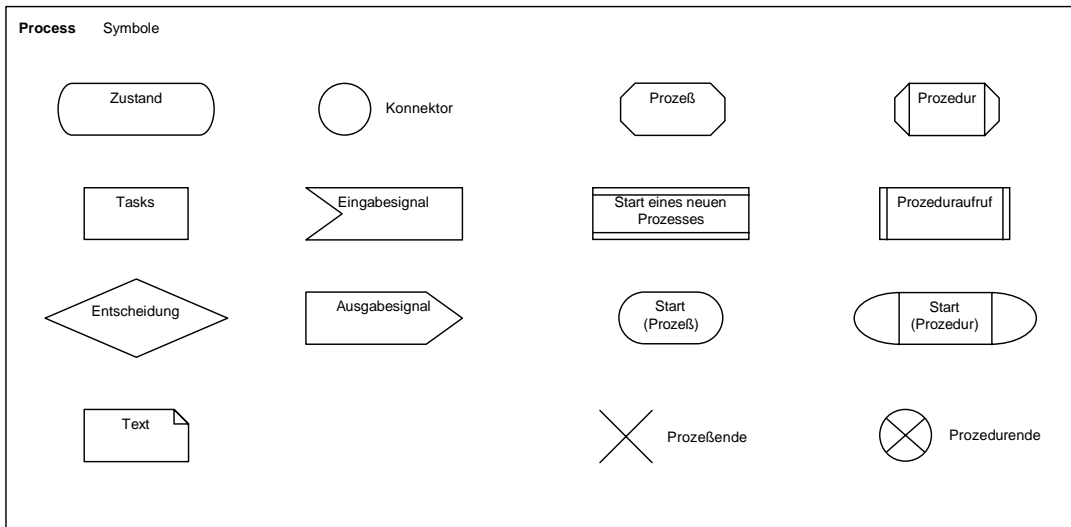


Abbildung 5.1: Übersicht einiger Symbole in SDL

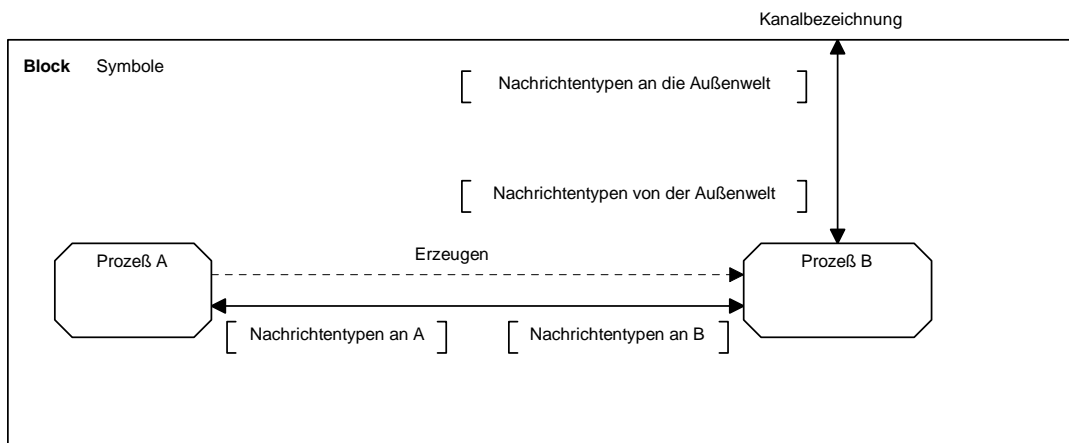


Abbildung 5.2: Übersicht einiger Symbole in einem SDL-Block

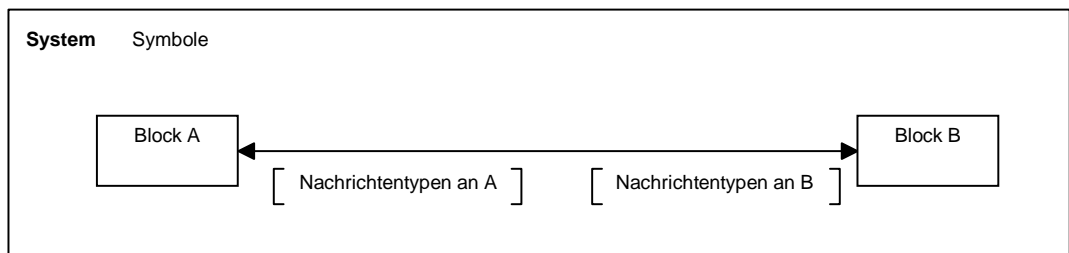


Abbildung 5.3: Übersicht einiger Symbole in einem SDL-System

Die Datenstrukturen, Nachrichten sowie Kodefragmente werden in einer an *Java* angelehnten Form in der Schriftart *Typewriter* dargestellt. Für die Beschreibung der Objekte wird ein Schlüsselwort *persistent* eingeführt, welches ein Objekt kennzeichnet, das persistent auf einem Speichermedium vorhanden ist. Existiert zu diesem Objekt ein Semaphor¹ (*guarded*), so befindet sich das Semaphor lediglich im flüchtigen Speicher. Die in den SDL-Diagrammen integrierten Kodefragmente beziehen sich auf lokale Objekte, wenn der Objektbezeichner klein geschrieben ist; ansonsten wird Bezug auf ein globales Objekt genommen. In den Diagrammen werden Abkürzungen verwendet, die sich auch auf englische Ausdrücke beziehen, wenn deren deutsche Übersetzung sprachlich holprig erscheint oder diese sich besser und eindeutiger abkürzen lassen.

5.2 System

Die Systemarchitektur des Replikationskonzepts ist vereinfacht als SDL-System in Abbildung 5.4 dargestellt. Die Klienten und Replikatverwalter kommunizieren über das Internet unter Verwendung von Kanälen. In den Klammern angegeben sind die Nachrichtentypen, die über einen Kanal übertragen werden können.

Die Nachrichtentypen haben folgende Bedeutung:

- *AufNT* ist ein Nachrichtentyp, der einen Auftrag zum Lesen oder Schreiben enthält, welcher von einem Klienten an einen Replikatverwalter erteilt wird.
- *AufAntNT* ist ein Nachrichtentyp, der die Antwort eines Replikatverwalters auf einen Auftrag enthält.
- *BstNT* ist ein Nachrichtentyp, der die Bestätigungsdaten zu einer Schreiboperation enthält.
- *DbNT* ist ein Nachrichtentyp für die Übertragung des vollständigen Replikatbestands eines Replikatverwalters inklusive des Schreiblogs.
- *KoordNT* ist ein Nachrichtentyp zur Koordinierung des Abgleichprozesses der Replikatverwalter (beispielsweise Aufbau einer Abgleichsitzung).

¹Die Realisierung eines Semaphors wird in Abschnitt 5.4 beschrieben.

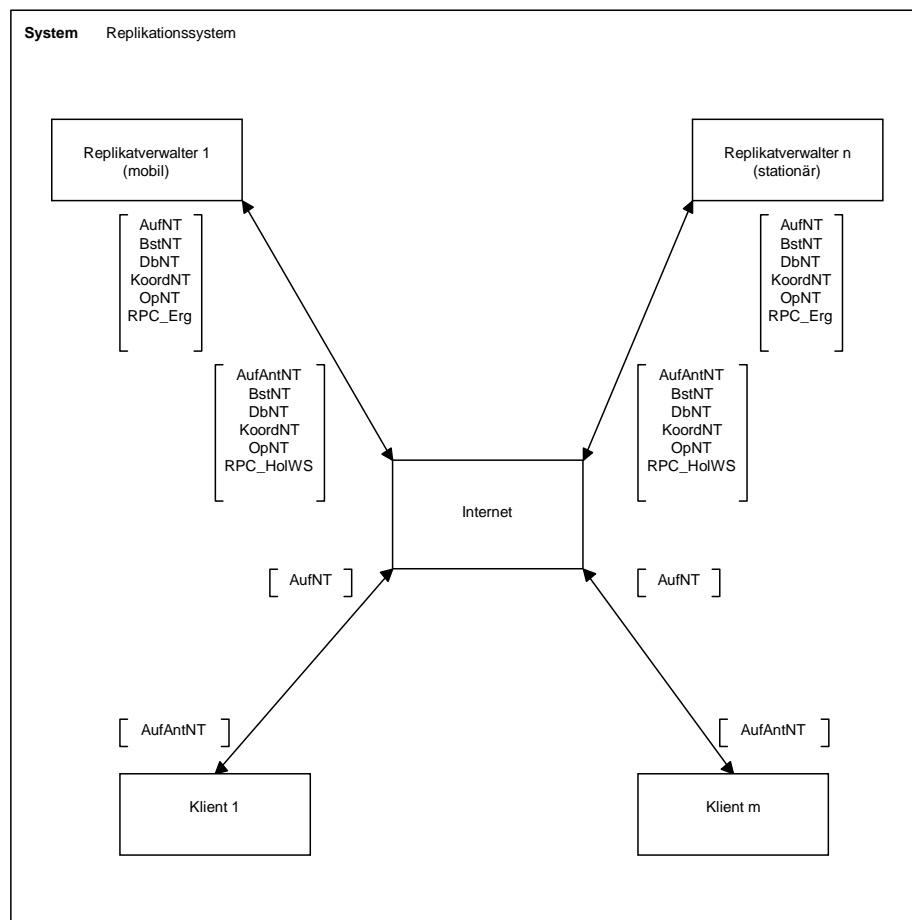


Abbildung 5.4: System in SDL

- OpNT ist ein Nachrichtentyp, der eine Schreiboperation enthält, welche zwischen zwei Replikatorverwaltern in einer Abgleichsitzung ausgetauscht wird.
- RPC_Erg repräsentiert das Ergebnis eines Methodenfernaufrufs (*Remote Procedure Call*, abgekürzt RPC).
- RPC_Ho1WS repräsentiert einen Methodenfernaufruf, um den Wissensstand eines Replikatorverwalters in einer Abgleichsitzung zu erfragen.

Jeder Nachrichtentyp enthält ein Attribut (sndID), welches den Kennzeichner eines Replikatorverwalters oder Klienten enthält, der eine Nachricht eines solchen Typs verschickt hat.

5.3 Replikatverwalter

In diesem Abschnitt wird die Realisierung eines Replikatverwalters beschrieben. Jeder Replikatverwalter hat einen ganzzahligen Zähler, der zusammen mit dem Kennzeichner eines Replikatverwalters für die Bildung der Annahmenummern oder im Fall des primären Replikatverwalters auch für die Bildung der Bestätigungsnummern verwendet wird.

Der Typ einer Annahmenummer wird wie folgt eingeführt:

TYP EINER ANNAHMENUMMER

```
class ANrT
{
int nr;
RVIDT rvID;
}
```

Das Attribut nr enthält den Zählerstand eines Replikatverwalters. Das Attribut rvID enthält den Kennzeichner eines Replikatverwalters, auf welchen sich der Zählerstand bezieht.

Empfängt ein Replikatverwalter einen syntaktisch und semantisch korrekten Schreibauftrag eines Klienten, so wird daraus eine Schreiboperation erzeugt. Diese enthält die Attribute des Schreibauftrags sowie Verwaltungsdaten von dem erzeugenden Replikatverwalter, nämlich eine Annahmenummer (die mit jedem empfangenen korrekten Auftrag um Eins erhöht wird) und bei einer Bestätigung durch den primären Replikatverwalter eine Bestätigungsnummer (die bei jeder Bestätigung um Eins erhöht wird). Der Typ einer Schreiboperation wird wie folgt eingeführt:

TYP EINER SCHREIBOPERATION

```
class OpT
{
int bNr;
ANrT aNr;
Operationsspezifische Attribute;
}
```

Das Attribut bNr enthält den Zählerstand, der die logische Zeit der Bestätigung einer Schreiboperation repräsentiert. Ist eine Schreiboperation unbestätigt, so enthält dieses Attribut den Wert ∞ , der in den SDL-Diagrammen durch $InfTy$ dargestellt wird. Eine bestätigte Schreiboperation enthält einen ganzzahligen Wert, der größer oder gleich Null ist. Das Attribut aNr enthält die Annahmenummer von dem Replikatverwalter, der die Schreiboperation erzeugt hat. Die operationsspezifischen Attribute werden durch den Typ der Schreiboperation und den Typ der Knoten bestimmt, auf welche sich die Schreiboperation bezieht. Diese sind jedoch für die Erläuterung der Realisierung unseres Replikationskonzepts unerheblich.

Die Schreiboperationen sind in Abgleichsitzungen in einer geordneten Reihenfolge dem Abgleichpartner bekannt zu machen. Dies wurde bereits in den Abschnitten 4.2 sowie 4.3 erläutert und drückt sich formal in den Konsistenzbeziehungen 4.7 sowie 4.8 aus. Zur Realisierung einer geordneten Übertragung werden Ordnungsrelationen über den Schreiboperationen eingeführt:

ORDNUNGSRELATIONEN

Seien x, y Instanzen vom Typ Schreiboperation aus der Menge \mathcal{O} aller Schreiboperationen. Die Menge \mathcal{O} ist die Vereinigung aller Teilmengen \mathcal{O}_{R_x} von Schreiboperationen, die von einem Replikatverwalter R_x erzeugt wurden.

Der Operator $x \stackrel{U}{<} y$ bezeichnet die **Ordnung der unbestätigten Schreiboperationen**. Diese Ordnung ist bezüglich der Menge \mathcal{O} partiell, bezüglich der Menge \mathcal{O}_{R_x} total.

Es gilt $x \stackrel{U}{<} y$, gdw. $(x.bNr = \infty) \wedge (y.bNr = \infty) \wedge (x.aNr.rvID = y.aNr.rvID) \wedge (x.aNr.nr < y.aNr.nr)$.

Der Operator $x \stackrel{B}{<} y$ bezeichnet die **Ordnung der bestätigten Schreiboperationen**. Diese Ordnung ist bezüglich aller bestätigten Schreiboperationen der Menge \mathcal{O} total.

Es gilt $x \stackrel{B}{<} y$, gdw. $(x.bNr \neq \infty) \wedge (y.bNr \neq \infty) \wedge (x.bNr < y.bNr)$.

Die Ordnung der unbestätigten Schreiboperationen ist lediglich total für alle unbestätigten Schreiboperationen, die von dem gleichen Replikatverwalter erzeugt wurden. Unbestätigte Schreiboperationen, die von unterschiedlichen Replikatverwaltern erzeugt wurden, stehen in keiner Ordnungsrelation zueinander, weil es keinen kausalen Zusammenhang gibt. Die Ordnung der bestätigten Schreiboperationen

wird von dem primären Replikatverwalter durch die Serialisierung bestimmt und ist systemweit total für alle bestätigten Schreiboperationen, unabhängig von dem Replikatverwalter, der diese erzeugt hat.

Um inkrementelle Abgleichsitzungen zwischen zwei Replikatverwaltern realisieren zu können, ist es notwendig, daß Änderungen am Replikatbestand von dem Replikatbestand selbst getrennt gespeichert werden. Dafür wird ein Schreiblog pro Replikatverwalter verwendet, welches alle bekannten Schreiboperationen enthält. Ein Replikatverwalter kann voraussichtlich nicht mehr benötigte Schreiboperationen aus dem Schreiblog entfernen, wie es in Abschnitt 4.5 beschrieben wurde. Der Typ eines Schreiblogs wird wie folgt eingeführt:

TYP EINES SCHREIBLOGS

```
class SLogT
```

```
{
```

BstOp(int bNr, ANrT aNr): Trägt die Bestätigung der unbestätigten Schreiboperation mit der Annahmenummer aNr und ihrer Bestätigungsnummer bNr ein.

EinfOp(OpT op): Fügt eine Schreiboperation ein.

EntfBOps(int bNr): Entfernt alle bestätigten Schreiboperationen, deren Bestätigungsnummer kleiner oder gleich bNr ist.

OpT **HolBOp**(int bNr): Liefert die Schreiboperation zurück, welche die Bestätigungsnummer bNr hat. undefiniert, wenn eine Schreiboperation mit der Bestätigungsnummer bNr nicht existiert oder bNr den Wert ∞ hat.

OpT **HolUOp**(ANrT aNr): Liefert die Schreiboperation zurück, welche die Annahmenummer aNr hat. undefiniert, wenn eine Schreiboperation mit der Annahmenummer aNr nicht existiert.

(int aV[], int hbNr) **Rekonstr**(int heNr): Rekonstruiert aus dem Schreiblog mit Hilfe der höchsten Bestätigungsnummer aller entfernten bestätigten Schreiboperationen den Annahmevektor und die höchste bekannte Bestätigungsnummer.

```
}
```


Für die Verwaltung des Replikatbestands wird auf jedem Replikatverwalter ein persistenter Speicher für strukturierte Datenobjekte benötigt. Ob dafür ein aufwendiges Datenbanksystem oder eine selbst entwickelte einfache Java-Klasse eingesetzt wird, hängt im wesentlichen von den Leistungsanforderungen des Diskurssystems ab und wird hier nicht diskutiert. Eine wichtige Funktionalität des persistenten Speichers ist die Möglichkeit der Rücknahme von vorläufig ausgeführten Schreiboperationen und einer anschließenden Wiederausführung. Damit kann die lokale Ausführungsreihenfolge der Schreiboperationen geändert werden, wenn dies durch eine Änderung der systemweiten Ausführungsreihenfolge erforderlich ist. Wird in einer Nachricht des Typs `DbNT` ein Objekt vom Typ einer Datenbank verschickt, so werden ausschließlich die Daten des Objekts versendet, in diesem Fall der Replikatbestand, und nicht der Code zur Verwaltung. Der Typ einer Datenbank wird wie folgt eingeführt:

TYP EINER DATENBANK

```
class DbT
```

```
{
```

ExeOp(OpT op): *Führt die Schreiboperation op im Replikatbestand aus. Treten Konflikte zwischen einer auszuführenden Schreiboperation und bereits ausgeführten Schreiboperationen auf, so werden diese, wie in Abschnitt 4.2 beschrieben, erkannt und aufgelöst.*

KnotenT **HolKobj**(ANrT kID): *Liefert den Knoten, identifiziert durch seinen Kennzeichner kID, der beim Erstellen des Knotens aus der Annahmenummer gebildet wurde, zurück.*

UndoUOps(): *Nimmt die vorläufige Ausführung aller Schreiboperationen im Replikatbestand zurück.*

RedoUOps(): *Führt mit Hilfe der Methode ExeOp alle Schreiboperationen im Replikatbestand aus, deren vorläufige Ausführung durch einen Aufruf der Methode UndoUOps rückgängig gemacht wurde. Ist eine vorläufig ausgeführte Schreiboperation (die damit zum Ausführungszeitpunkt als unbestätigt bekannt war) rückgängig gemacht worden, die danach als bestätigte Schreiboperation durch ExeOp endgültig ausgeführt wurde, so wird diese bei der erneuten Ausführung der rückgängig gemachten Schreiboperationen nicht mehr berücksichtigt.*

```
}
```

Zur Prozeßkommunikation sind Objekte notwendig, die global von allen Prozessen verwendet werden können. Diese sind wie folgt:

GLOBALE OBJEKTE

guarded int **AnzESProz**. Anzahl der nebenläufig aktiven Sende- und Empfangsprozesse.

persistent guarded ANrT **AV**[]): Annahmevektor, der die höchste bekannte Annahmenummer für jeden Replikatverwalter festhält.

persistent DbT **Db**: Datenbank, die den Replikatbestand des Replikatverwalters enthält.

persistent ANrT **ExeANr**[]): Vektor, der die niedrigste Annahmenummer der unbestätigten Schreiboperationen für jeden Replikatverwalter enthält, die bereits bekannt und im Schreiblog vorhanden aber noch nicht ausgeführt sind. Sind keine unbestätigten Schreiboperationen von einem Replikatverwalter auszuführen, so erhält dessen Eintrag den Wert -1.

persistent int **ExeBNr**. Niedrigste Bestätigungsnummer, ab welcher bestätigte Schreiboperationen bereits im Schreiblog vorhanden aber noch nicht ausgeführt sind. Ist keine bestätigte Schreiboperation auszuführen, ist der Wert -1.

guarded bool **ExeL**. Semaphore zur Synchronisierung von Prozessen, die exklusiv auf einem Replikatverwalter laufen müssen.

persistent guarded int **HbBNr**: Höchste Bestätigungsnummer, die dem Replikatverwalter bekannt ist.

persistent int **HeBNr**: Repräsentiert die höchste Bestätigungsnummer aller entfernten Schreiboperationen.

Primär(). Prädikat, welches wahr ist, falls der Replikatverwalter der primäre ist.

persistent RVIDT **RVIDs**[]): Enthält die Kennzeichner aller Replikatverwalter, die einem Replikatverwalter bekannt sind.

persistent SLogT **SLog**: Schreiblog des Replikatverwalters zur Speicherung von Schreiboperationen.

Ein Replikatverwalter wird in unserer Realisierung als SDL-Block modelliert, der in Abbildung 5.5 dargestellt ist. Dort sind die einzelnen Prozesse mit ihren Kanälen für einen Nachrichtenaustausch, die Typen der austauschbaren Nachrichten und die Erzeugungsbeziehungen zwischen den einzelnen Prozessen abgebildet.

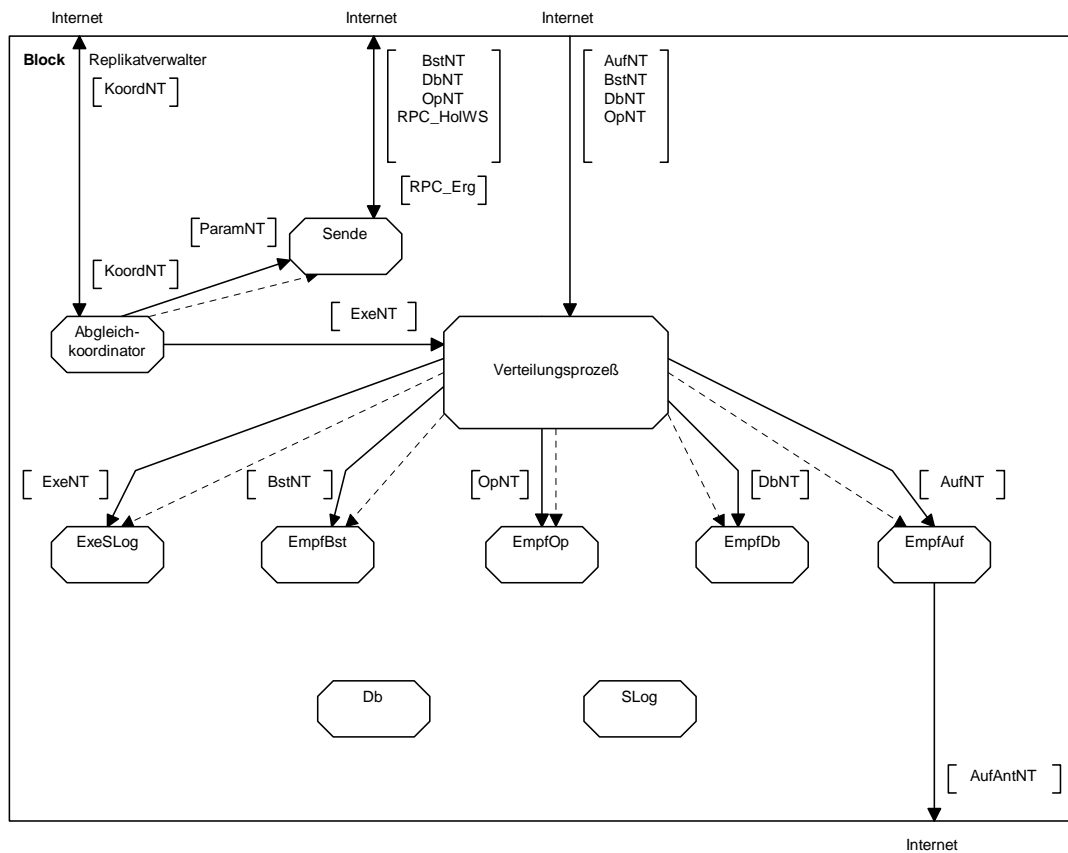


Abbildung 5.5: Replikatorverwalter in SDL

Folgende Aufgaben sind den einzelnen Prozessen zugeordnet:

- **Abgleichskordinator.** Dieser Prozeß koordiniert die Durchführung von Abgleichssitzungen, -runden und -zyklen mit anderen Replikatorverwaltern des Systems sowie die Ausführung von Schreiboperationen im Replikatorbestand, die in Abgleichssitzungen empfangen wurden.
- **Db.** Dieser Prozeß verwaltet den Replikatorbestand in der Datenbank.
- **EmpfAuf.** Für jeden empfangenen Auftrag eines Klienten wird ein solcher Prozeß erzeugt, der den Auftrag bearbeitet und daraus gegebenenfalls eine Schreiboperation erzeugt.
- **EmpfBst.** Für jede in einer Abgleichssitzung empfangene Bestätigungsnachricht wird ein solcher Prozeß erzeugt, welcher die Bestätigungsdaten für die als unbestätigt bekannte Schreiboperation im Schreiblog vermerkt.

- **EmpfDB.** Für eine in einer Abgleichsitzung empfangene Nachricht, die einen kompletten Replikatabestand sowie ein Schreiblog enthält, wird ein Prozeß erzeugt, der diesen Replikatabestand auf dem empfangenden Replikatverwalter einrichtet. Eine solche Nachricht wird empfangen, wenn ein Replikatverwalter neu erzeugt wurde oder so weit in einer Abgleichsitzung außerhalb der „Synchronisation“ liegt, daß ein inkrementeller Abgleich nicht vorgenommen werden kann.
- **EmpfOp.** Für jede in einer Abgleichsitzung empfangene Nachricht mit einer Schreiboperation wird ein solcher Prozeß erzeugt, der die Schreiboperation in das Schreiblog einträgt und gegebenenfalls im Replikatabestand ausführt.
- **ExeSLog.** Die in Abgleichsitzungen empfangenen Schreiboperationen, die noch nicht im Replikatabestand ausgeführt wurden, werden durch diesen Prozeß ausgeführt. Die Ausführung wird durch den Abgleichkoordinator initiiert.
- **Verteilungsprozeß.** Der Verteilungsprozeß erzeugt und startet Prozesse für die Bearbeitung von empfangenen Nachrichten. Zudem kontrolliert er die Nebenläufigkeit der erzeugten Prozesse.
- **Sende.** Für jede Abgleichsitzung wird ein Sendeprozeß erzeugt, der einem Partner während einer Abgleichsitzung jene Schreiboperationen übermittelt, die diesem unbekannt sind.
- **SLog.** Dieser Prozeß verwaltet das Schreiblog.

Jeder Replikatverwalter hat einen eindeutigen Kennzeichner, der vom Typ RVIDT ist. Der Kennzeichner wird bei der Erzeugung eines Replikatverwalters, wie im Modell in Abschnitt 4.5 beschrieben, festgelegt.

5.4 Nebenläufigkeit und der Verteilungsprozeß

Die Nebenläufigkeit der Prozesse eines Replikatverwalters erfordert eine Koordination der Ressourcennutzung, die durch Semaphore erfolgt. Die Realisierung eines Semaphors ist in Abbildung 5.6 dargestellt. Das Semaphor unterstützt den Schutz

eines Betriebsmittels und ist um eine Leseoperation erweitert, in der das geschützte Betriebsmittel ohne Blockierung gelesen werden kann. Unberechtigte Unlock-Operationen an ein Betriebsmittel werden ignoriert. Das Semaphore besitzt zwei Zustände, in denen ein Betriebsmittel entweder offen oder gesperrt ist. Eine Sperre kann nur durch den ursprünglich sperrenden Prozeß aufgehoben werden. Ein Prozeduraufruf zum Sperren ist in Abbildung 5.7 dargestellt. In der Realisierung wird vorausgesetzt, daß ein Objekt von einer Semaphoreklasse erben kann. Dies wird, wie in Abschnitt 5.1 beschrieben, durch das Schlüsselwort `guarded` gekennzeichnet.

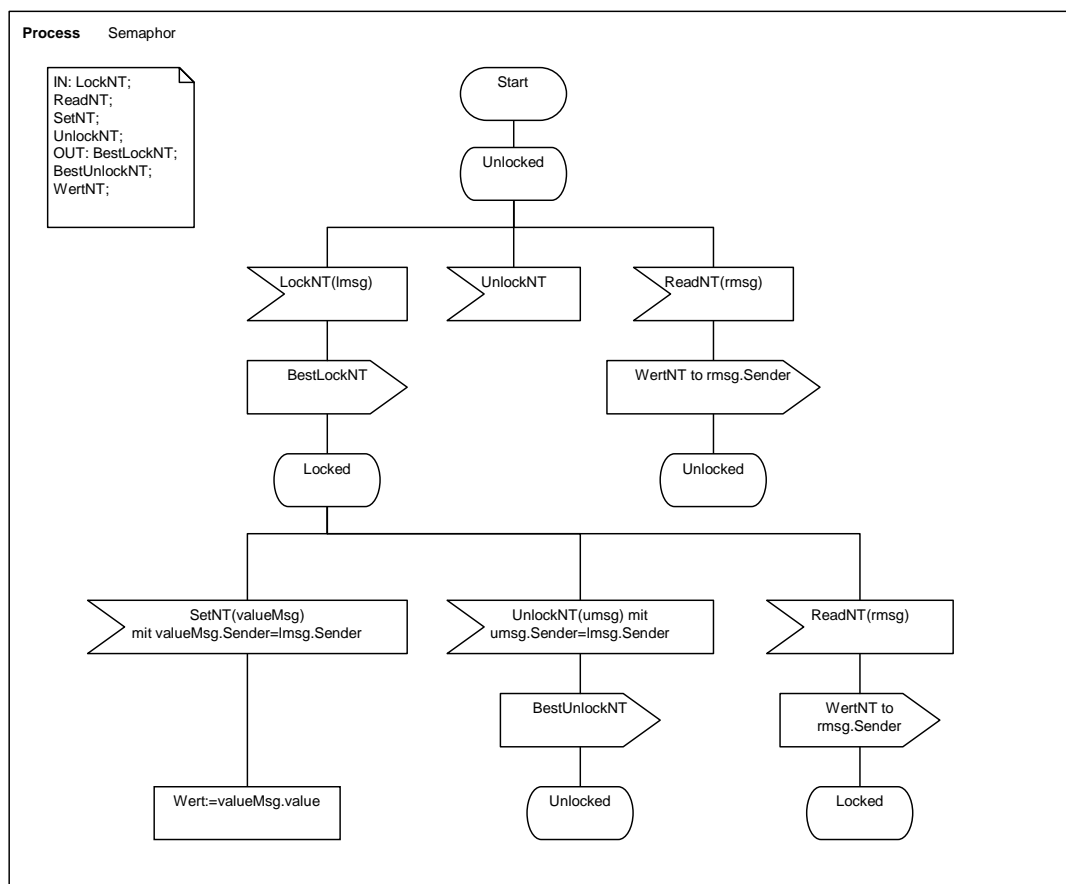


Abbildung 5.6: Semaphore

Die Typen der Aufgaben, die ein Replikatverwalter nebenläufig bearbeiten kann, sind im Modell in Abschnitt 4.6 beschrieben. In diesem Abschnitt wird die Nebenläufigkeit der Prozesse erläutert, die eine nebenläufige Bearbeitung dieser Aufgaben ermöglicht.

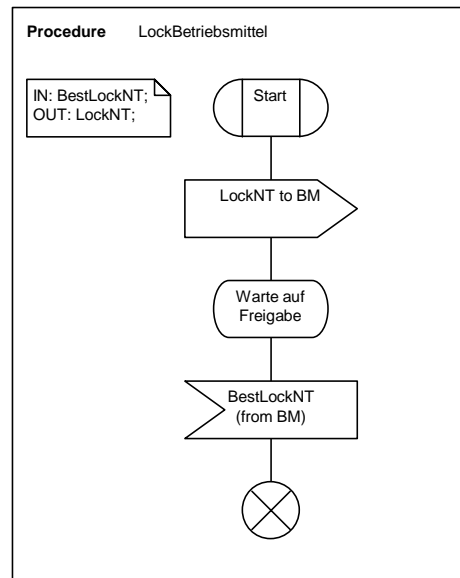


Abbildung 5.7: Reservieren des exklusiven Zugriffs auf ein Betriebsmittel

Um Abgleichsitzungen nebenläufig zur Bearbeitung von Klientenaufträgen durchführen zu können, wurde der Vorgang des Empfangens und Eintragens von Schreiboperationen und Bestätigungen von dem Vorgang der Ausführung im Replikatabstand getrennt. Empfangene Schreiboperationen aus Abgleichsitzungen werden erst nach einem vollständigen Abgleichzyklus von dem Prozeß ExeSLog gruppiert und exklusiv ausgeführt. Die Trennung von Empfang und Ausführung von Schreiboperationen bringt einen signifikanten Performanzgewinn. Die Abgleichsitzungen erfordern im allgemeinen eine Rücknahme der vorläufigen Ausführung von unbestätigten Schreiboperationen (Konsistenzbeziehung 4.11), damit durch eine Abgleichsitzung bekannt gewordene bestätigte Schreiboperationen in dem Replikatabstand ausgeführt werden können. Während der Durchführung von Abgleichsitzungen sollen jedoch auch Klienten bedient werden, deren Aufträge sofort im Replikatabstand auszuführen sind. Diese müssen in dem aktuellen und nicht dem zurückgesetzten Replikatabstand ausgeführt werden, damit die potentielle Kausalität der unbestätigten Schreiboperationen erhalten bleibt. Eine Realisierung, in der sowohl die empfangenen Schreiboperationen aus Abgleichsitzungen als auch die durch Aufträge von Klienten erzeugten Schreiboperationen sofort im Replikatabstand ausgeführt werden, würde ein fortwährendes „Springen“ des Replikatabstands zur Folge haben, was sowohl den Durchsatz als auch die Antwortzeit des Dienstes signifikant verringern würde. Das Springen entsteht durch das Zurücknehmen von Schreiboperationen und

Prozeß	AK	EAuf	EBst	EDb	EOp	ExeSLog	Snd	Vrtl
AK	Nein	Ja	Ja	Ja	Ja	Ja	Ja	Ja
EAuf		Ja	Ja	Nein	Ja	Nein	Ja	Ja
EBst			Nein	Nein	Ja ₁	Nein	Ja	Ja
EDb				Nein	Nein	Nein	Nein	Ja
EOp					Ja ₁	Nein	Ja	Ja
ExeSLog						Nein	Nein	Ja
Snd							Ja	Ja
Vrtl								Nein

Tabelle 5.1: Parallelität der Prozesse

deren erneute Ausführung. Weil der primäre Replikatverwalter die systemweite Ausführungsreihenfolge festlegt und keine Schreiboperationen zurücknehmen braucht, ist für diesen eine Trennung von Empfang und Ausführung der Schreiboperationen nicht notwendig.

Die mögliche Parallelität der einzelnen Prozesse ist in der Tabelle² 5.1 dargestellt. ‚Ja‘ steht für Prozesse, die gleichzeitig ablaufen können. ‚Nein‘ steht für Prozesse, die entweder nicht gleichzeitig ablaufen oder in einem Wartezustand gestartet werden, um auf die Freigabe eines Semaphors zur Prozeßsynchronisation zu warten.

Die Prozesse EmpfDB und ExeSLog müssen bezogen auf die Sende- und Empfangsprozesse exklusiv aktiv sein. Die Notwendigkeit für die exklusive Ausführung der Schreiboperationen des Schreiblogs wurde bereits begründet. Der Prozeß EmpfDB darf nicht nebenläufig zu Sende- und Empfangsprozessen aktiv sein, weil er einen vollständigen Replikatbestand inklusive eines Schreiblogs auf einem Replikatverwalter einrichtet. Ein Replikatverwalter hat nur einen Abgleichkoordinator und einen Verteilungsprozeß, weil deren Koordinierungsaufgaben nicht aufgeteilt und parallelisiert werden können.

Der Verteilungsprozeß, dargestellt in Abbildung 5.8, instantiiert und startet die Prozesse für die Bearbeitung empfangener Nachrichten und überwacht die exklusive Ausführung der Prozesse EmpfDB und ExeSLog, die erst dann instantiiert werden, wenn alle aktiven Sende- und Empfangsprozesse beendet sind. Die Koordination der Anzahl von Empfangsprozessen, die in Tabelle 5.1 mit ‚Ja₁‘ gekennzeichnet sind,

²Die Prozeßnamen wurden abgekürzt, damit die Tabelle ungedreht dargestellt werden kann.

wird in den einzelnen Prozessen selbst realisiert, die über Semaphore in einen Wartezustand gehen. Außerdem darf es nur jeweils einen aktiven Prozeß vom Typ EmpfOP und EmpfBst für jede Schreiboperation oder deren Bestätigung geben, die von demselben Replikatverwalter erzeugt wurde, damit die parallele Bearbeitung nicht zu einer Änderung der Reihenfolge führt.

5.5 Prozesse zur Realisierung von Abgleichsitzungen

Ein besondere Bedeutung für das Replikationskonzept haben jene Prozesse eines Replikatverwalters, welche den Abgleich der Replikatbestände realisieren und in diesem Abschnitt beschrieben werden.

Der Prozeß Abgleichkoordinator koordiniert die Durchführung von Abgleichsitzungen, -runden und -zyklen mit anderen Replikatverwaltern des Systems. Dafür werden Nachrichten des Typs KoordNT für die Koordination des Abgleichprozesses³ ausgetauscht. Die Partnerwahl für Abgleichsitzungen und deren algorithmische Realisierung ist in Abschnitt 4.3 beschrieben und wird hier nicht wiederholt. Nach der Beendigung eines Abgleichzyklus initiiert der Abgleichkoordinator die Ausführung der in den Abgleichsitzungen bekannt gewordenen Schreiboperationen durch eine Nachricht an den Prozeß ExeSLog.

Prozesse vom Typ Sende (Abbildung 5.9 und 5.10) übermitteln einem Abgleichpartner in einer Abgleichsitzung alle Schreiboperationen und Bestätigungen, die diesem unbekannt sind. Dafür wird zuerst der Wissensstand des Partners erfragt. Anschließend wird festgestellt, ob eine Abgleichsitzung inkrementell durchgeführt werden kann. Ist dies nicht möglich, so wird dem Partner der vollständige Replikatbestand inklusive des Schreiblogs übermittelt. Im Normalfall ist jedoch eine inkrementelle Abgleichsitzung durchführbar und es werden dem Partner alle unbekannt bestätigten Schreiboperationen übermittelt. Ist dem Partner eine inzwischen bestätigte Schreiboperation als unbestätigt bekannt, so werden lediglich deren Bestätigungsdaten und nicht die Schreiboperation selbst übertragen. Die Übertragung erfolgt in der Bestätigungsreihenfolge ($\overset{B}{<}$) der Schreiboperationen (Realisierung der

³Der im Kapitel der Realisierung verwendete Begriff des Prozesses bezieht sich auf Prozesse in SDL. Der Begriff Abgleichprozeß gilt in der Definition von Seite 10 und bezieht sich auf die allgemeine, nicht SDL spezifische Bedeutung des Wortes Prozeß.

Konsistenzbeziehung 4.8). Anschließend erfolgt die Übertragung der unbestätigten Schreiboperationen in der Annahmereihenfolge ($\stackrel{U}{<}$), wie es in der Konsistenzbeziehung 4.7 festgelegt ist. Die dargestellten Diagramme beziehen sich auf Abgleichsitzungen von stationären Replikativverwaltern. Erfolgt eine Abgleichsitzung zwischen einem stationären und einem mobilen Replikativverwalter, so besteht der Unterschied lediglich darin, daß der stationäre Replikativverwalter nur solche Schreiboperationen übermittelt, die der mobile Replikativverwalter mit partieller Replikation benötigt. Des weiteren werden Verschiebe-Operationen in lokale Schreiboperationen für den mobilen Replikativverwalter umgesetzt, weil Unterbäume in dessen Replikativbestand aufgenommen oder entfernt werden (siehe Abschnitt 4.3).

Jede empfangene Nachricht, die eine Schreiboperation aus einer Abgleichsitzung enthält, wird durch einen Prozeß vom Typ `EmpfOp` bearbeitet, der in Abbildung 5.11 dargestellt ist. Liegt eine unbestätigte Schreiboperation vor, so wird zuerst der Annahmvektor für jenen Replikativverwalter gesperrt, welcher die empfangene Schreiboperation erzeugt hat. Damit wird verhindert, daß nebenläufig durchgeführte Abgleichsitzungen mit mobilen und stationären Replikativverwaltern zu Inkonsistenzen durch eine Reihenfolgeänderung der Bearbeitung von Schreiboperationen führen. Anschließend wird überprüft, ob die empfangene Schreiboperation bereits im Schreiblog vorhanden ist, weil diese bereits in einer nebenläufigen Abgleichsitzung empfangen wurde. Liegt eine unbekannte Schreiboperation vor, so wird diese im Fall eines sekundären Replikativverwalters im Schreiblog eingetragen und gegebenenfalls die Annahmnummer für eine Ausführung im Replikativbestand vorgemerkt. Ist der empfangene Replikativverwalter der primäre, so wird die Schreiboperation bestätigt und sowohl im Schreiblog eingetragen als auch im Replikativbestand ausgeführt. Der primäre Replikativverwalter führt jede bekannte Schreiboperation sofort im Replikativbestand aus, weil dieser keine Schreiboperationen zurücknimmt.

Empfängt ein Replikativverwalter eine Nachricht mit Bestätigungsdaten, so kennt er eine Schreiboperation als unbestätigt, welche seinem Abgleichspartner inzwischen als bestätigt bekannt ist. Der Prozeß `EmpfBst`, dargestellt in Abbildung 5.12, trägt die Bestätigungsdaten einer als unbestätigt bekannten Schreiboperation in das Schreiblog ein und merkt gegebenenfalls die Bestätigungsnummer zur Ausführung vor.

Ist ein Replikatverwalter so weit außerhalb der „Synchronisation“ mit seinem Partner, daß eine inkrementelle Abgleichsitzung nicht durchgeführt werden kann, so empfängt er den vollständigen Replikatbestand, das Schreiblog und die Bestätigungsnummer der jüngsten entfernten Schreiboperation des Partners. Der Prozeß *EmpfDb* für die Bearbeitung einer solchen Nachricht ist in Abbildung 5.13 dargestellt. Zuerst muß das eigene Schreiblog gesichert werden, damit die Schreiboperationen, die dem Partner unbekannt sind, nicht verloren gehen. Danach wird der neue Replikatbestand eingerichtet, der Wissensstand aus dem Schreiblog rekonstruiert und das neue Schreiblog um die Schreiboperationen ergänzt, die lediglich im alten und nicht im neuen Schreiblog vorhanden sind. Nach einer Initialisierung der Variablen ist der Replikatverwalter einsatzbereit.

Der vom Abgleichkoordinator initiierte Prozeß *ExeSLog* (Abbildung 5.14) führt die Gruppe jener Schreiboperationen im Replikatbestand aus, die im letzten Abgleichzyklus empfangen wurden und noch nicht ausgeführt sind. Zuerst erfolgt eine Überprüfung, ob bestätigte Schreiboperationen zur Ausführung vorliegen. Ist dies der Fall, so wird die Ausführung aller unbestätigten Schreiboperationen zurückgenommen (Konsistenzbeziehung 4.11). Anschließend werden die noch nicht ausgeführten bestätigten Schreiboperationen in der Bestätigungsreihenfolge ausgeführt (Konsistenzbeziehung 4.12). Es folgt die Ausführung der zurückgenommenen Schreiboperationen an der gegebenenfalls geänderten Position in der Ausführungsreihenfolge. Danach werden die noch nicht ausgeführten unbestätigten Schreiboperationen in der Erzeugungsreihenfolge ausgeführt (Konsistenzbeziehung 4.13).

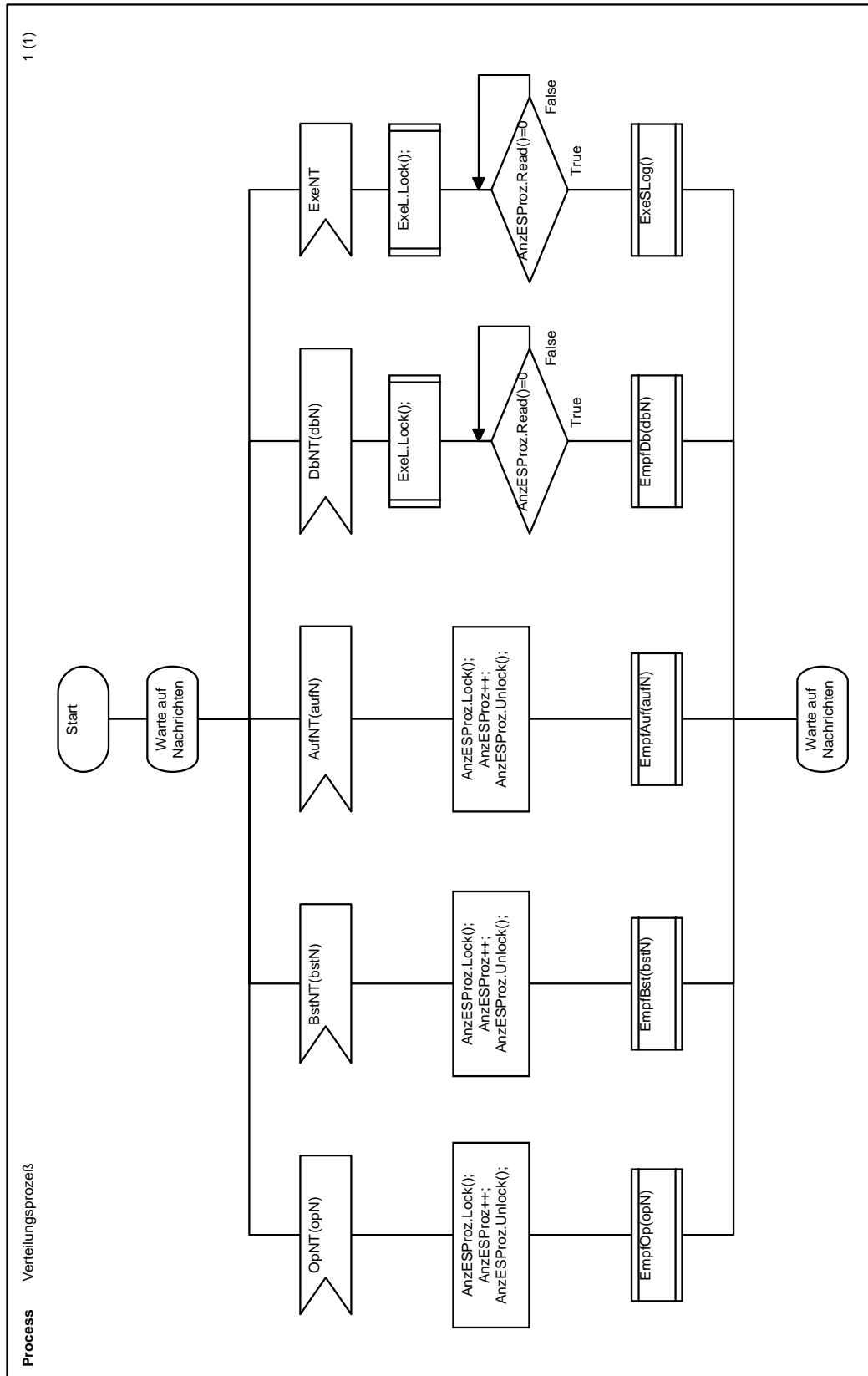


Abbildung 5.8: Verteilungsprozess

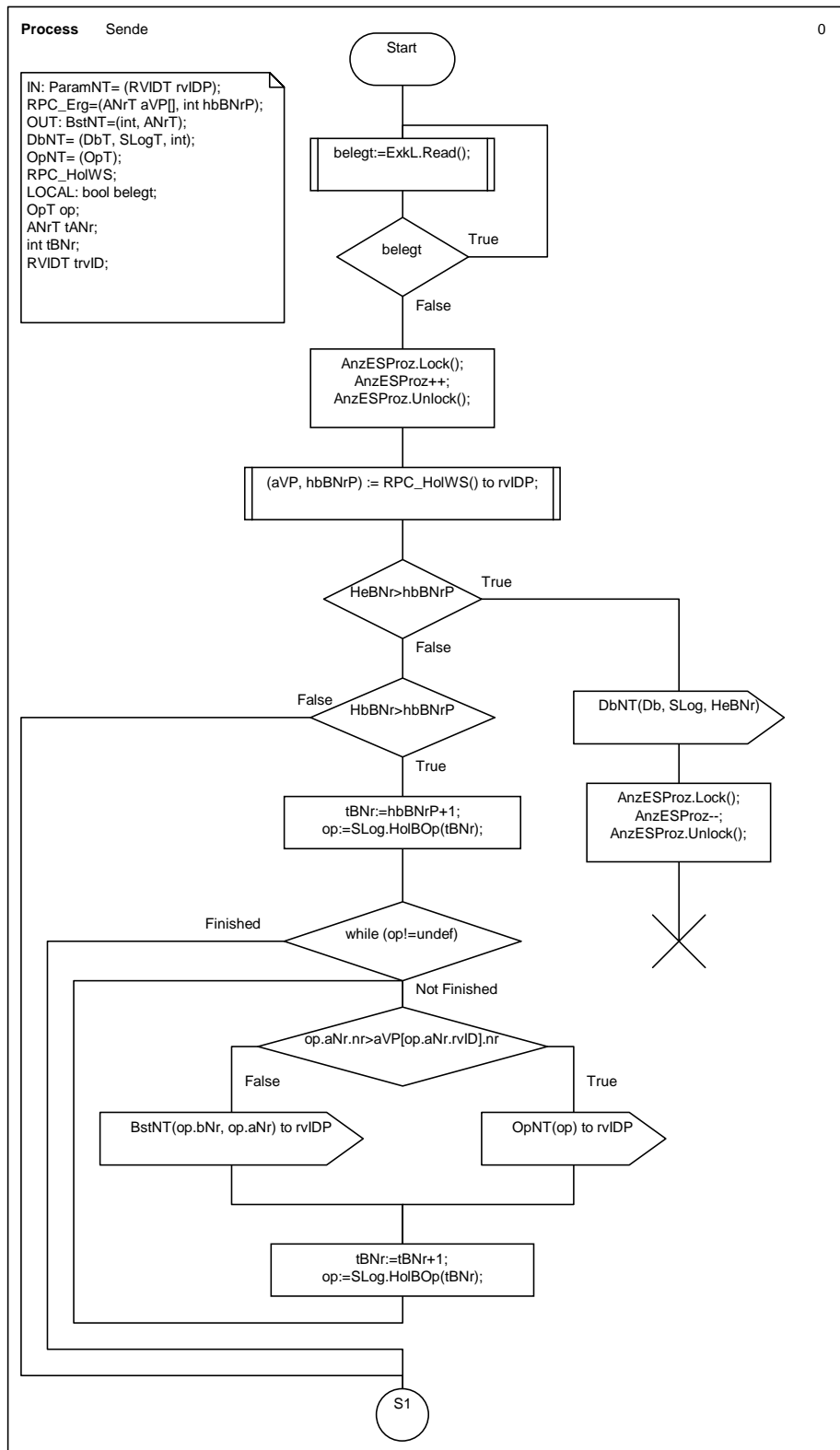


Abbildung 5.9: Sendeprozess Teil 1

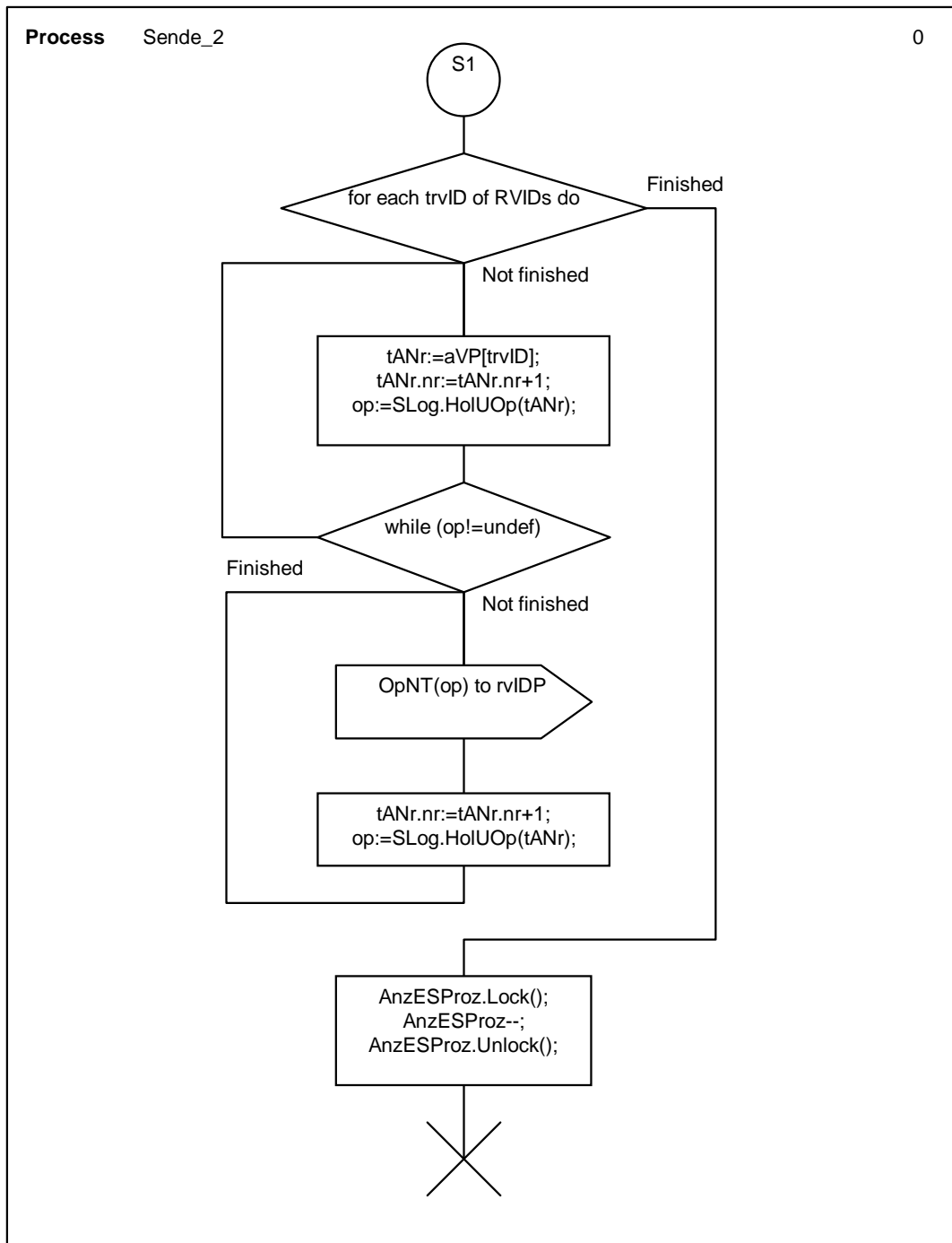


Abbildung 5.10: Sendeprozess Teil 2

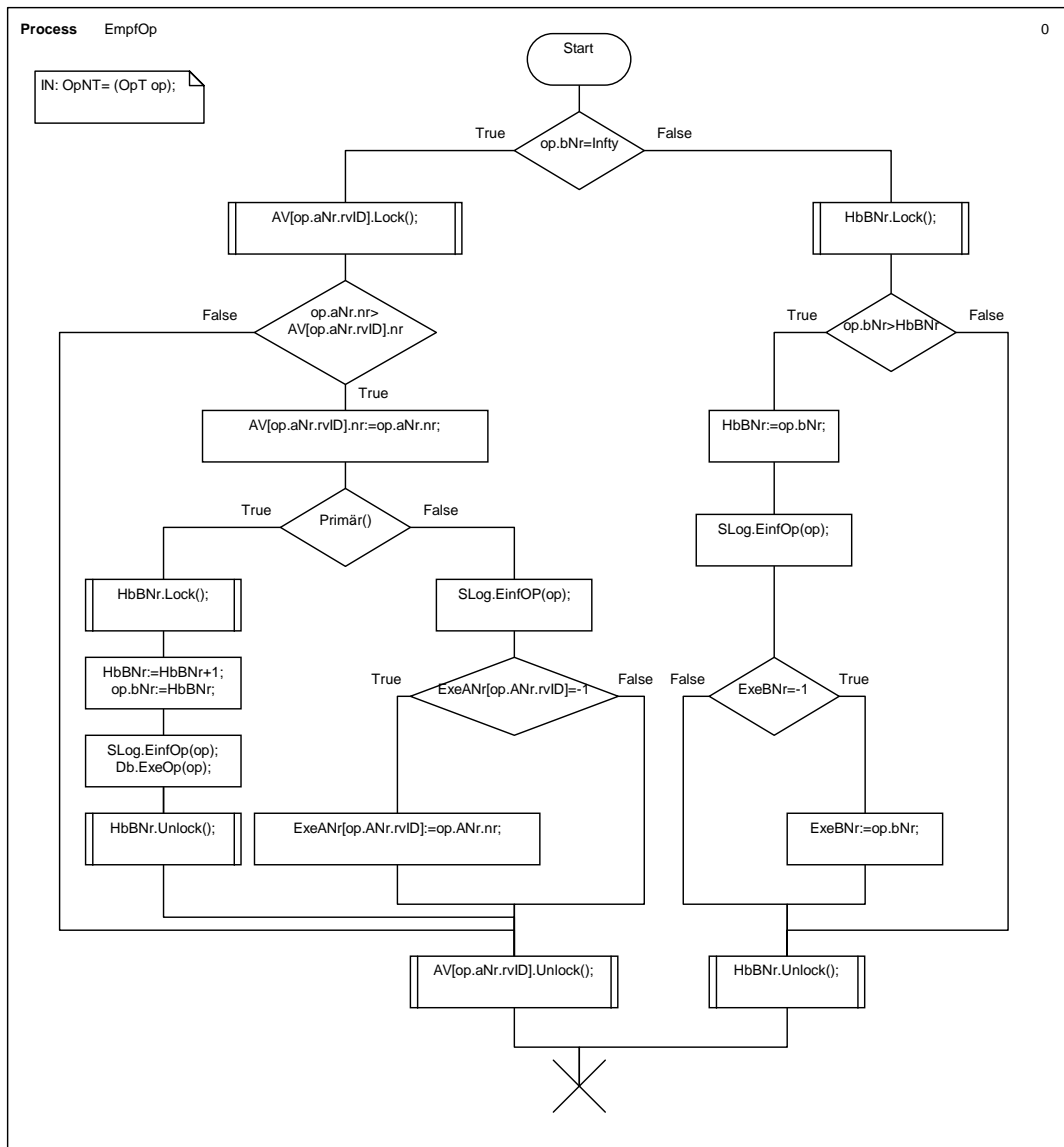


Abbildung 5.11: Empfang einer Schreiboperation in einer Abgleichsitzung

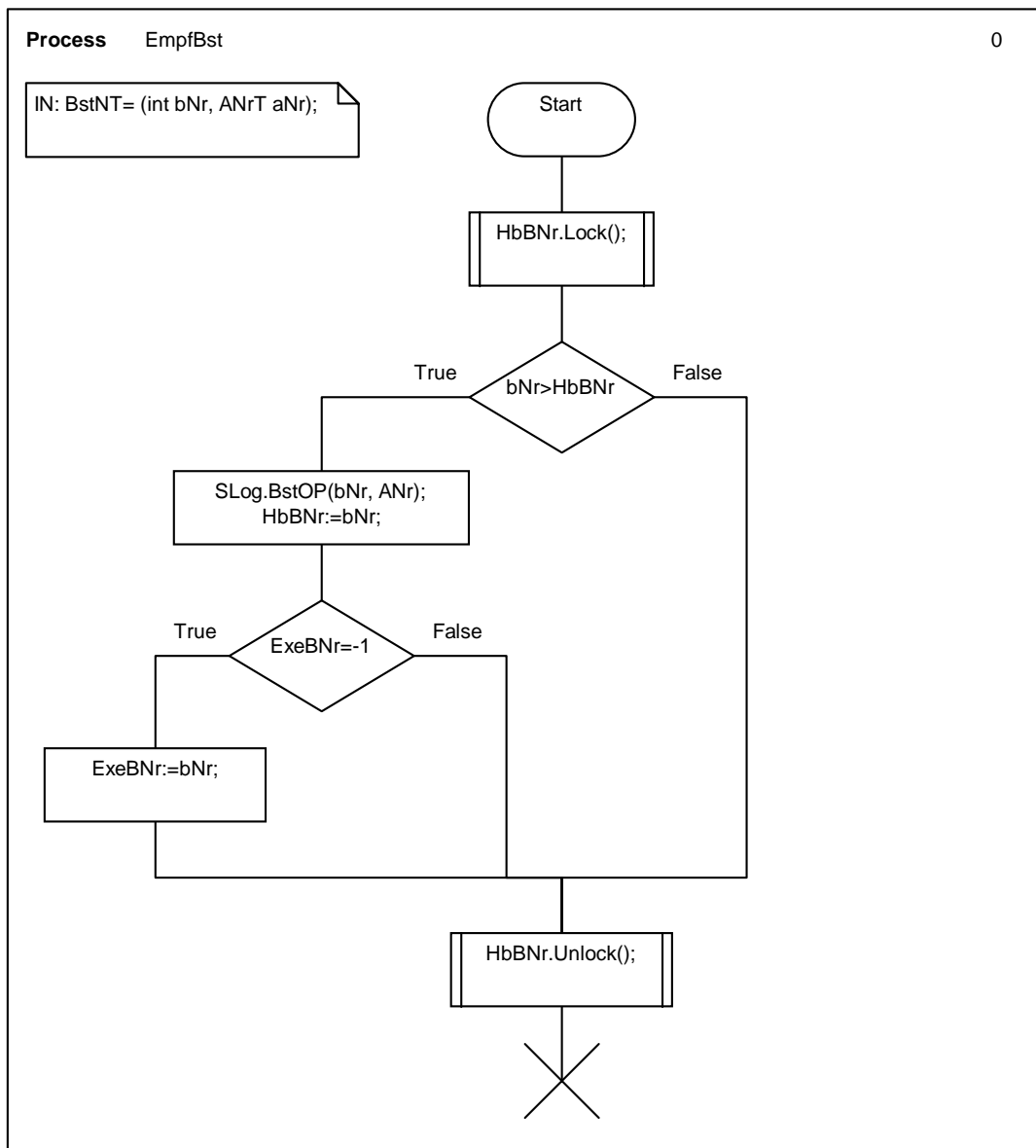


Abbildung 5.12: Empfang der Bestätigung einer Schreiboperation

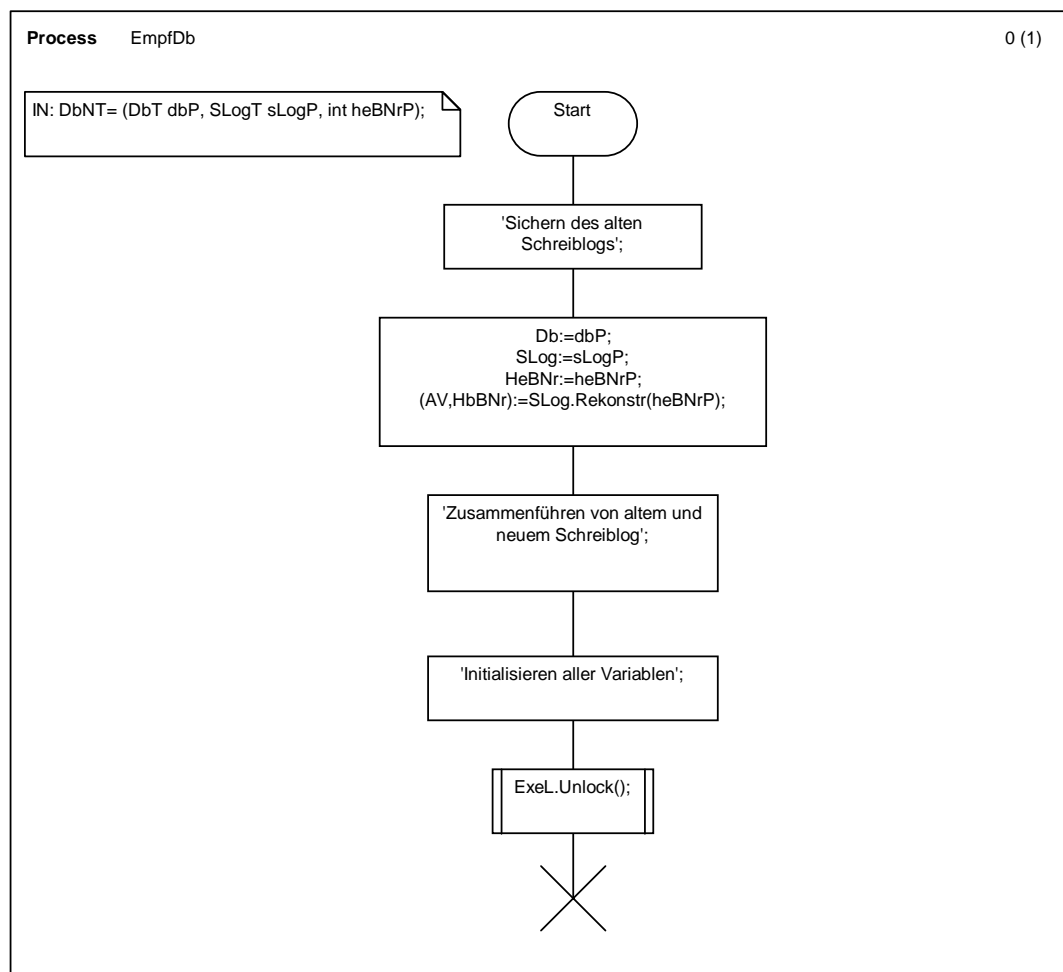


Abbildung 5.13: Empfang eines vollständigen Replikatbestands mit Schreiblog

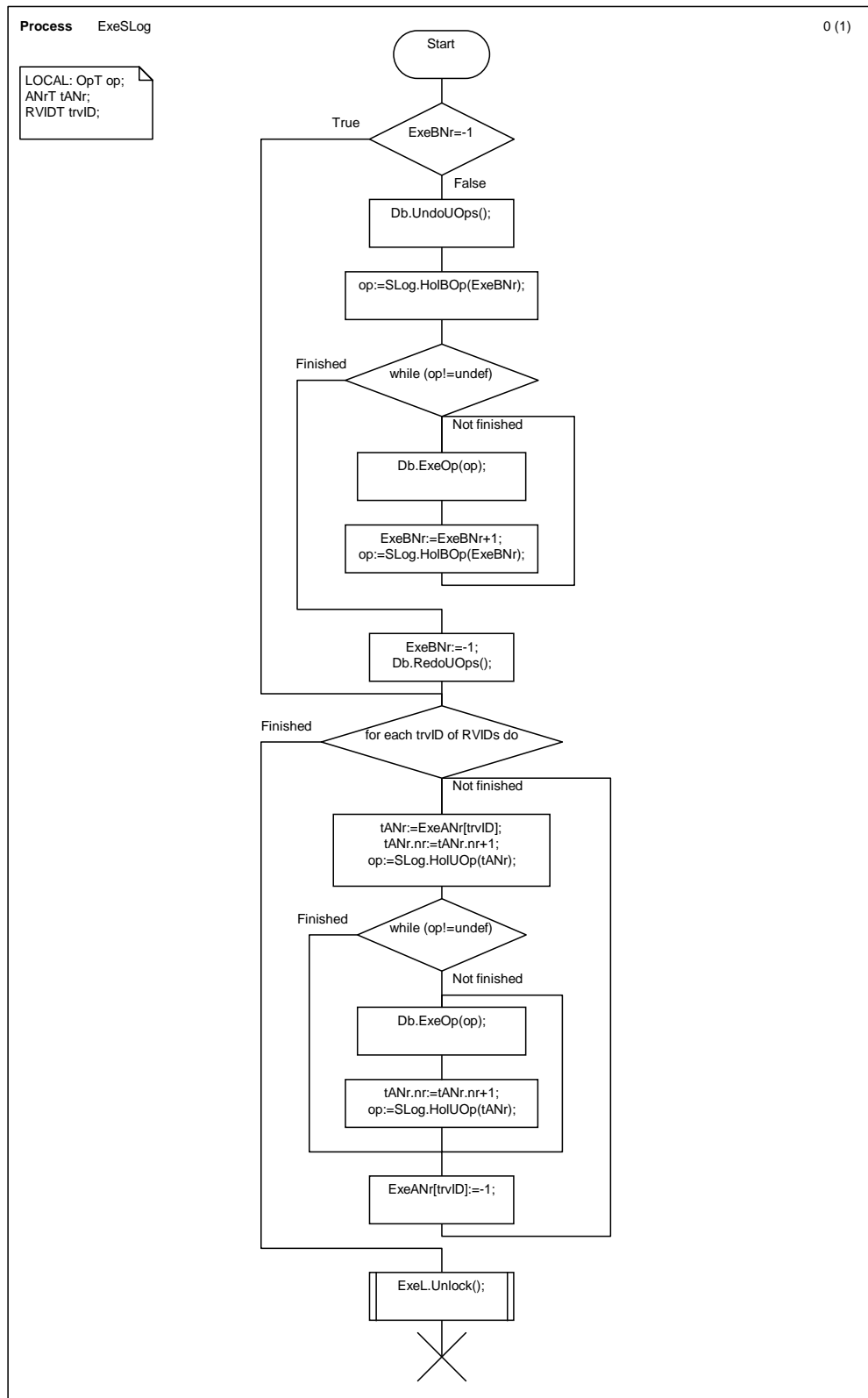


Abbildung 5.14: Ausführung von Schreiboperationen des Schreiblogs

Schlußfolgerungen und offene Fragen

In den untersuchten Publikationen über Replikationskonzepte mit schwacher Konsistenz wird das jeweilige Gesamtkonzept lediglich in Ausschnitten betrachtet. Dabei werden weder die verschiedenen Perspektiven, unter denen solche Replikationskonzepte zu betrachten sind, noch deren Zusammenhänge aufgezeigt. Eine Ansammlung unterschiedlicher Mosaiksteine in einer oftmals nicht konsistenten Begrifflichkeit ist die Folge. Ein umfassendes Gesamtkonzept, welches diese Mosaiksteine zu einem konsistenten Ganzen zusammenfügt, ist nicht erkennbar. Folglich ist es schwierig, Replikationskonzepte mit schwacher Konsistenz in ihrer Gesamtheit zu verstehen. So müssen beispielsweise Entwickler, die ihre Applikation um ein Replikationskonzept erweitern möchten, einen erheblichen Aufwand für das Studium von Publikationen über bereits bekannte, jedoch nicht adäquat dargestellte Einzelkonzepte aufwenden, deren Zusammenführung keineswegs trivial ist. Es ist die Leistung dieser Diplomarbeit, die signifikanten Aspekte eines Gesamtkonzepts und ihre Interdependenzen zu identifizieren und das entworfene Replikationskonzept aus unterschiedlichen Perspektiven in einer konsistenten Begrifflichkeit darzustellen. Aus dieser Betrachtung ergeben sich mehrere einander ergänzende Teilmodelle, auf deren Grundlage ein durchgängiges Verständnis der vielfältigen Aspekte von Replikationskonzepten mit schwacher Konsistenz gewonnen werden kann. Die Teilmodelle mit den Bezeichnungen Systemmodell, Konsistenzmodell, Abgleichmodell, Fehlermodell, Mobilitätsmodell und Nebenläufigkeitsmodell bilden das in dieser Arbeit entwickelte Gesamtmodell.

Des weiteren zeigt diese Arbeit an einem konkreten Konsistenzmodell, wie Konsistenzbeziehungen für ein Replikationskonzept mit schwacher Konsistenz formal spezifiziert werden können. Dazu wird eine first-order language (FOL) auf Basis der Prädikatenlogik 1. Stufe eingeführt, welche die Beziehungen zwischen den verteilten Replikatbeständen, ihren Veränderungen und den nebenläufig erteilten Änderungsaufträgen beschreibt.

In der Diskussion der Konfliktbehandlung wird die Notwendigkeit begründet, daß der Konflikt-Begriff um die Kontextdiskrepanz zu erweitern ist. Zu deren Erkennung wird in Anlehnung an den Einsatz von Hash-Funktionen in digitalen Signaturen die Verwendung von Prüfsummen vorgeschlagen, mit denen Eigenschaften der Bezugsbäume von Aufträgen effizient dargestellt werden können. Damit kann die Nutzersicht bei der Erteilung eines Auftrags für einen ausgedehnten potentiellen Konfliktbereich, der nicht nur einzelne Knoten, sondern vollständige Bäume umfassen kann, festgehalten werden.

Der in dieser Arbeit beschriebene Abgleichprozeß beruht auf paarweisen Abgleichsitzungen von Replikatverwaltern. Für die Auswahl der Abgleichpartner wird ein Algorithmus vorgestellt, der optimal für die vorliegenden Anforderungen geeignet ist und durch eine gute Verteilung der Last im System einen Abgleich der Replikatbestände mit geringem Zeitaufwand unterstützt. An Hand der Anforderungen werden die Vorzüge dieses Algorithmus zu alternativen Ansätzen herausgearbeitet.

Zur Unterstützung mobiler Nutzer sind mobile Replikatverwalter im Replikationskonzept vorgesehen. Um deren limitierte Ressourcen (geringer Massenspeicherbedarf, über längere Zeiträume unsichere Kommunikationsverbindung mit geringer Bandbreite) optimal zu nutzen und damit den Einsatz von Replikation für diese Nutzergruppe in der Praxis zu ermöglichen, wird die vollständige um die partielle Replikation erweitert. Diese verwendet die Struktur von Bäumen zur zusätzlichen Verringerung des Kommunikationsaufwands in Abgleichsitzungen.

Die bis hierher beschriebenen originären Leistungen und Schlußfolgerungen dieser Diplomarbeit lassen sich nahtlos mit den Konzepten verbinden, die den Stand der Kunst darstellen. In dieser Verbindung gewinnen die Vorzüge des entwickelten Replikationskonzepts eine allgemeinere Bedeutung, wie im folgenden kurz gezeigt werden soll.

Der replizierte Dienst bietet eine hohe Verfügbarkeit, Performanz und Skalierbarkeit. Insbesondere bei der Verwendung eines Kommunikationsmediums mit größeren unvorhersehbaren Schwankungen in Bandbreite und Latenzzeit wie dem Internet sind solche qualitativen Eigenschaften für eine hohe Anzahl von Nutzern nur über Replikationskonzepte mit schwacher Konsistenz realisierbar. Durch eine autonome Bearbeitung von Klientenaufträgen auf den einzelnen Replikatverwaltern kann der Bedarf an Kommunikation zu Abgleichzwecken reduziert werden, die dann gebündelt und zu flexibel festlegbaren Zeitpunkten erfolgen kann. Die Vielzahl unterschiedlicher Aufgaben, die ein Replikatverwalter nebenläufig bearbeiten kann, und die Verwendung inkrementeller Abgleichsitzungen führen zu einer weiteren Performanzsteigerung.

Der Abgleichprozeß wird in paarweisen Abgleichsitzungen durchgeführt, die zu Abgleichrunden und Abgleichzyklen gruppiert sind, deren Initiierungszeitpunkte frei wählbar sind. Damit ist ein hohes Maß an Flexibilität gegeben, die in einer Abgleichpolitik einen Ausgleich zwischen der durchschnittlichen Abweichung der Replikatbestände und dem notwendigen Kommunikationsaufwand für deren Abgleiche zu finden gestattet.

In Fehlersituationen verliert ein auf dem entwickelten Replikationskonzept basierendes System bei zunehmender Fehleranzahl nur allmählich an Funktionalität (hauptsächlich Performanz), ohne abrupt auszufallen (graceful degradation). Nach Fail-Stop-Fehlern von Netzknoten und Kommunikationsverbindungen (inklusive einer Netzpartitionierung) sind keine Recovery-Maßnahmen des Systems notwendig.

Zur Unterstützung mobiler Nutzer werden mobile Replikatverwalter mit partieller Replikation zur Verfügung gestellt, die leichtgewichtig erzeugt werden können. Garantien für Klientensitzungen erhöhen die Konsistenz der Sicht von Klienten.

Die Breite der in dieser Diplomarbeit gewählten Herangehensweise bringt es mit sich, daß an mehreren Stellen Fragen offen bleiben, die sich aus der gedanklichen Fortführung des Erreichten ergeben. So wurde zwar ein Algorithmus für die Auswahl von Abgleichpartnern entwickelt, jedoch dessen Korrektheit, ebenso wie seine Komplexität, lediglich durch eine Simulation überprüft. Dafür wurde eine Anzahl von bis zu 1000 Replikatverwaltern betrachtet. Ein allgemeiner mathematischer Beweis steht aus.

Der im Replikationskonzept realisierte Abgleichprozeß hat einen hohen Grad an Flexibilität, der durch geeignete Abgleichpolitiken ausgenutzt werden kann. Zuvor wäre jedoch die Frage zu klären, welche Elemente (Ereignisse, Bedingungen) für eine ausdrucksmächtige Politikspezifikation erforderlich sind. In einem weiteren Schritt wäre die Integration der Abgleichpolitik in das Konsistenzmodell anzustreben. Eine bidirektionale Abbildung zwischen der Abgleichpolitik und dem Konsistenzmodell würde die Möglichkeit eröffnen, aus der Spezifikation einer Abgleichpolitik detailliertere Aussagen zur Konsistenz als im vorgestellten Modell zu gewinnen und umgekehrt aus einer Festlegung der Parameter der Konsistenzbeziehungen eine Abgleichpolitik abzuleiten.

In dieser Arbeit wird die partielle Replikation ausschließlich für mobile Replikatverwalter betrachtet. Dies ist für das Einsatzgebiet vollkommen ausreichend. In anderen Anwendungsgebieten sind jedoch Fälle denkbar, in denen die partielle Replikation auch für stationäre Replikatverwalter sinnvoll ist. Dies wäre beispielsweise in einem Umfeld gegeben, in welchem auch stationäre Replikatverwalter lediglich über schmalbandige Kommunikationsverbindungen verfügen und die zu replizierenden Daten einen starken lokalen Bezug haben. Die Ausdehnung der partiellen Replikation auf stationäre Replikatverwalter hätte große Auswirkungen auf den Systementwurf und insbesondere auf die Abgleichalgorithmen. Es bleibt daher die Frage zu klären, wie eine solche partielle Replikation in das vorliegende Replikationskonzept zu integrieren wäre.

Abbildungsverzeichnis

2.1	Beispiel einer Netzpartitionierung	12
2.2	Relative Häufigkeit von Lese- zu Schreibaufträgen während einer Woche	15
2.3	Absolute Häufigkeit von Schreibaufträgen während einer Woche	16
3.1	Available-Copies-Replikationskonzept bei Netzpartitionierung	21
3.2	Primary-Copy-ROWA	22
3.3	Primary-Copy-ROWA bei Netzpartitionierung	23
3.4	Gewichteter Quorumalgorithmus	25
4.1	Systemmodell	40
4.2	Durchführung von Abgleichsitzungen bei nebenläufiger Bearbeitung von Klientenaufträgen	41
4.3	Bearbeitung von Aufträgen	42
4.4	Ausschnitt einer Abgleichsitzung	42
4.5	Serialisierung von Schreiboperationen	43
4.6	Sicht eines Nutzers beim Löschen eines Unterbaumes	56
4.7	Partnerwahl in einem Zyklus	71

4.8	Permutationsprinzip der Partnerwahl	72
4.9	Verschiebe-Operationen bei uneingeschränkter, partieller Replikation	78
4.10	Eingeschränkte Auswahl an Partnern von mobilen Replikativwaltern für Sitzungen	79
4.11	Verschiebe-Operationen aus Sicht eines mobilen Replikativwalters mit partieller Replikation	81
4.12	Nebenläufige Bearbeitung von Aufgaben	93
5.1	Übersicht einiger Symbole in SDL	97
5.2	Übersicht einiger Symbole in einem SDL-Block	97
5.3	Übersicht einiger Symbole in einem SDL-System	97
5.4	System in SDL	99
5.5	Replikativwalter in SDL	105
5.6	Semaphor	107
5.7	Reservieren des exklusiven Zugriffs auf ein Betriebsmittel	108
5.8	Verteilungsprozeß	113
5.9	Sendeprozeß Teil 1	114
5.10	Sendeprozeß Teil 2	115
5.11	Empfang einer Schreiboperation in einer Abgleichsitzung	116
5.12	Empfang der Bestätigung einer Schreiboperation	117
5.13	Empfang eines vollständigen Replikativbestands mit Schreiblog	118
5.14	Ausführung von Schreiboperationen des Schreiblogs	119

Tabellenverzeichnis

4.1	Operationen auf Bäumen in Replikatbeständen	40
4.2	Konflikte zwischen verschiedenen Schreiboperationen	58
4.3	Prüfsummenbildung für Schreiboperationen	60
5.1	Parallelität der Prozesse	109

Literaturverzeichnis

- [AD76] ALSBERG, PETER und J. D. DAY: *A principle for resilient sharing of distributed resources*. Proceedings of the 2nd International Conference on Software Engineering, Seiten 627–644, 1976.
- [AU96] AHO, ALFREDO V. und JEFFREY D. ULLMAN: *Informatik – Datenstrukturen und Konzepte der Abstraktion*. Informatik Lehrbuch-Reihe. International Thomson Publishing, 1996.
- [BG84] BERNSTEIN, PHILIP A. und NATHAN GOODMAN: *The Serializability of Concurrent Database Updates*. ACM Transactions on Database Systems, 9(4):596–615, 1984.
- [BHG87] BERNSTEIN, P., V. HADZILACOS und N. GOODMAN: *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.
- [BJ87] BIRMAN, KENNETH P. und THOMAS A. JOSEPH: *Reliable Communication in the Presence of Failures*. ACM Transactions on Computer Systems, 5(1):47–76, 1987.
- [Car99] CARDELLI, LUCA: *Abstractions for Mobile Computation*. In: *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, Band 1603 der Reihe *Lecture Notes in Computer Science*, Seiten 51–94. Springer, 1999.
- [CCI88] CCITT: *CCITT Recommendation Z.100: Specification and Description Language SDL*, Band VI.20 - VI.24 der Reihe *Blue Book*. ITU, 1988.

- [CDK94] COULOURIS, GEORGE, JEAN DOLLIMORE und TIM KINDBERG: *Distributed Systems – Concepts and Design*. Addison-Wesley, zweite Auflage, 1994.
- [CGMW96] CHAWATHE, S., H. GARCIA-MOLINA und J. WIDOM: *A toolkit for constraint management in heterogeneous information systems*. Proceedings of the Twelfth International Conference on Data Engineering, Seiten 56–65, 1996.
- [Chr91] CHRISTIAN, F.: *Reaching Agreement on Processor-group Membership in Synchronous Distributed Systems*. Distributed Computing, 4, 1991.
- [CS95] CRISTIAN, FLAVIU und FRANK SCHMUCK: *Agreeing on Processor Group Membership in Timed Asynchronous Distributed Systems*. Technischer Bericht CSE95-428, University of California, Los Angeles, 1995.
- [DGH⁺87] DEMERS, A., D. GREENE, C. HAUSER, W. IRISH und J. LARSON: *Epidemic algorithms for replicated database maintenance*. Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, Seiten 1–12, 1987.
- [EMP⁺97] EDWARDS, W. K., E. D. MYNATT, K. PETERSEN, M. J. SPREITZER, D. B. TERRY und M. M. THEIMER: *Designing and Implementing Asynchronous Collaborative Applications with Bayou*. Proceedings of the Tenth ACM Symposium on User Interface Software and Technology, Seiten 119–128, 1997.
- [FLP85] FISCHER, M., N. LYNCH, und M. PATERSON: *Impossibility of distributed consensus with one faulty process*. Journal of the ACM, 32(2):374–382, 1985.
- [GHM⁺90] GUY, RICHARD G., JOHN S. HEIDEMANN, WAI MAK, THOMAS W. PAGE, JR., GERALD J. POPEK und DIETER ROTHMEIR: *Implementation of the Ficus Replicated File System*. In: *Proceedings of the Summer 1990 USENIX Conference, Anaheim*, Seiten 63–72, 1990.

- [Gif79] GIFFORD, DAVID K.: *Weighted Voting for Replicated Data*. Proceedings of the Seventh Symposium on Operating System Principles, Seiten 150–162, 1979.
- [Git00] GIT, OLD: *Forum Statistics*, 2000. <http://pub5.ezboard.com/fyourdontipsandguidelines.showMessage?topicID=100.topic>.
- [GMB⁺81] GRAY, J. N., P. MCJONES, M. BLASGEN, B. LINDSAY, L. LORIE, T. PRICE, F. PUTZOLU und I. TRAIGER: *The Recovery Manager of the System R Database Manager*. ACM Computing Surveys, 13(2):223–242, 1981.
- [Gra78] GRAY, N.: *Notes on data base operating systems*. In: *Operating Systems – An advanced course*, Band 60 der Reihe *Lecture Notes in Computer Science*, Seiten 393–481. Springer, 1978.
- [Hei77] HEIN, OLAF: *Graphentheorie für Anwender*. Bibliographisches Institut, 1977.
- [HHB96] HELAL, ABDELSALAM, ABDELSALAM HEDDAYA und BHARAT BHARGAVA: *Replication Techniques in Distributed Systems*. The Kluwer International Series on Advances in Database Systems. Kluwer Academic Publishers, 1996.
- [Hor99] HORSTMANN, THILO: *Unterstützung virtueller Arbeitsumgebungen durch replizierte Kooperationsanwendungen: Spezifikation eines vektorzeitbasierten Replikationsverfahrens für den Einsatz im Internet*. Nummer 14/1999 in *GMD Research Series*. GMD – Forschungszentrum Informationstechnik GmbH, 1999.
- [HR83] HÄRDER, THEO und ANDREAS REUTER: *Principles of Transaction-Oriented Database Recovery*. Computing Surveys, 15(4):287–317, 1983.
- [Jun94] JUNGnickel, DIETER: *Graphen, Netzwerke und Algorithmen*. BI-Wissenschaftsverlag, dritte Auflage, 1994.
- [KS93] KUMAR, PUNEET und MAHADEV SATYANARAYANAN: *Log-Based Directory Resolution in the Coda File System*. In: *Proc. Second Int.*

- Conf. on Parallel and Distributed Information Systems*, Seiten 202–213, San Diego, CA (USA), 1993.
- [Lam78] LAMPORT, L.: *Time, clocks and ordering of events in a distributed system*. Communications of the ACM, 21(7):558–565, 1978.
- [Lam81] LAMPSON, B.: *Atomic transactions*. In: *Distributed Systems – Architecture and Implementation*, Band 105 der Reihe *Lecture Notes in Computer Science*, Seiten 246–265, 1981.
- [LSP82] LAMPORT, LESLIE, ROBERT SHOSTAK und MARSHALL PEASE: *The Byzantine generals problem*. ACM Transactions on Programming Languages and Systems, 4(3):382–401, 1982.
- [Lyn96] LYNCH, NANCY A.: *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.
- [MSC⁺86] MORRIS, J.H., M. SATYANARAYANAN, M. H. CONNER, J. H. HOWARD, D. S. ROSENTHAL und F. D. SMITH: *Andrew: A distributed personal computing environment*. Communications of the ACM, 29(3), 1986.
- [Mul89] MULLENDER, SAPE: *Distributed Systems*. Frontier Series. ACM Press, 1989.
- [PGH⁺91] PAGE, JR., THOMAS W., RICHARD G. GUY, JOHN S. HEIDEMANN, GERALD J. POPEK, WAI MAK und DIETER ROTHMEIER: *Management of Replicated Volume Location Data in the Ficus Replicated File System*. In: *Proceedings of the Summer 1991 USENIX Conference, Nashville*, Seiten 17–30, 1991.
- [PGH⁺98] PAGE, T., J. GUY, J. HEIDEMANN, D. RATNER, P. REIHER, A. GOEL, G. KUENNING und G. POPEK: *Perspectives on optimistically replicated peer-to-peer filing*. Software – Practice and Experience, 28(2):155–180, 1998.
- [PPR⁺83] PARKER JR., D. STOTT, GERALD J. POPEK, GERARD RUDISIN, ALLEN STOUGHTON, BRUCE J. WALKER, EVELYN WALTON, JOHANNA M. CHOW, DAVID A. EDWARDS, STEPHEN KISER und

- CHARLES S. KLINE: *Detection of Mutual Inconsistency in Distributed Systems*. IEEE Transactions on Software Engineering, 9(3):240–247, 1983.
- [PS83] PETERSON, R. J. und J. P. STRICKLAND: *Log Write-Ahead Protocols and IMS/VS Logging*. In: *Proceedings of the Second ACM SIGACT News-SIGMOD Symposium on Principles of Database Systems*, ACM Computing Surveys, Seiten 216–243, 1983.
- [PST⁺97] PETERSEN, K., M. J. SPREITZER, D. B. TERRY, M. M. THEIMER und A. J. DEMERS: *Flexible Update Propagation for Weakly Consistent Replication*. Proceedings of the 16th ACM Symposium on Operating Systems Principles, Seiten 288–301, 1997.
- [PSWL95] PARRINGTON, G. D., S. K. SHRIVASTAVA, S. M. WHEATER und M. C. LITTLE: *The Design and Implementation of Arjuna*. USENIX Computing Systems Journal, 8(3), 1995.
- [RPR96] RATNER, DAVID, GERALD J. POPEK und PETER REIHER: *Peer Replication with Selective Control*. Technischer Bericht CSD-960031, University of California, Los Angeles, 1996.
- [Sch96] SCHNEIDER, H.-J.: *Lexikon der Informatik und Datenverarbeitung*. Oldenbourg, vierte Auflage, 1996.
- [SKK⁺90] SATYANARAYANAN, M., J. J. KISTLER, P. KUMAR, M. E. OKASAKI, E. SIEGEL und D. STEERE: *Coda: A highly available file system for a distributed workstation environment*. IEEE Transactions on Computers, 39(4), 1990.
- [Sto79] STONEBRAKER, M.: *Concurrency control and consistency of multiple copies of data in distributed INGRES*. IEEE Transactions on Software Engineering, 5(3):188–194, 1979.
- [TDP⁺94] TERRY, D. B., A. J. DEMERS, K. PETERSEN, M. J. SPREITZER, M. M. THEIMER und B. B. WELCH: *Session Guarantees for Weakly Consistent Replicated Data*. Proceedings International Conference on Parallel and Distributed Information Systems, Seiten 140–149, 1994.

-
- [YV00a] YU, HAIFENG und AMIN VAHDAT: *Building Replicated Internet Services Using TACT: A Toolkit for Tunable Availability and Consistency Tradeoffs*. In: *Proceedings of the 8th ACM SIGOPS European Workshop on Support for Composing Distributed Applications*, Seiten 75–84, 2000.
- [YV00b] YU, HAIFENG und AMIN VAHDAT: *Design and Evaluation of a Continuous Consistency Model for Replicated Services*. In: *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation*, 2000.