CARL VON OSSIETZKY universität OLDENBURG

DEPARTMENT FÜR INFORMATIK
SYSTEMSOFTWARE UND VERTEILTE SYSTEME

Konzeption und Implementierung eines Rahmenwerkes für adaptiv-dynamische Replikationsstrategien

Diplomarbeit

6. Januar 2006

Christian Storm
Schützenweg 42
26129 Oldenburg

Erstprüfer Prof. Dr.-Ing. Oliver Theel
Zweitprüfer Dipl.-Inf. Philipp Hahn

Erklärung zur Urheberschaft

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe.

Oldenburg, den 6. Januar 2006

Christian Storm

Replication techniques are commonly used to improve the availability of critical data and operations thereon in a way not achievable using central, single-server approaches. Many replication strategies were presented in literature so far, most of them directed towards a specific optimization for a given application scenario. It became more and more obvious that there is no strategy that is generally best suited for any purpose. If the application's need changes over time, a time-consuming, error-prone and cost-intensive manual adaption of the replication strategy's implementation needs to be done to suit the new requirements. Thus, generalizing frameworks which allow an easy adaption to changing applications' needs by means of changing the configuration and not the implementation were developed. However, these frameworks mostly are still limited with respect to the maximum number of replicas they can manage. At design time, it has to be clear what the projected maximal number of replicas will be. The main objective of this diploma thesis is to overcome this limitation regarding the *dynamic General Structured Voting* framework by providing and implementing mechanisms to create or delete replicas at runtime, thereby allowing a free adoption to the numbers of replicas at runtime.
Futhermore, known dynamic replication strategies exhibit homogeneous behavior in their dynamics, i.e. they are strategy-bound for all numbers of replicas they manage by means of always using the same replication strategy. A mechanism to permit the usage of inhomogeneous replication strategies, i.e. using different (static) strategies for different numbers of replicas, will be presented and analyzed by example with respect to the provided operation availability compared to homogeneous strategies.

Replikationstechniken werden eingesetzt, um die Verfügbarkeit kritischer Daten bzw. den Operationen darauf in einer Weise zu erhöhen, die mit zentralen Ansätzen nicht zu erreichen ist. Viele der bisher vorgestellten Replikationsverfahren sind für ein bestimmtes Anwendungsszenario entwickelt und entsprechend spezifisch optimiert worden. Kein Verfahren ist jedoch für alle Anwendungsszenarien gleichermaßen gut geeignet. Bei sich ändernden Anforderungen des Anwendungsszenarios an das Replikationsverfahren muss das Verfahren manuell den neuen Anforderungen angepasst werden, was kostenintensiv, aufwendig und fehlerträchtig ist. Die daraufhin entwickelten generalisierenden Replikationsverfahren ermöglichen die Anpassung der Strategie ohne die Implementation verändern zu müssen. Die meisten generalisierenden Replikationsverfahren sind jedoch in Hinsicht auf die maximale Anzahl zur Laufzeit verwaltbarer Replikate beschränkt. Diese Obergrenze muss im Voraus bekannt sein und kann zur Laufzeit nicht verändert werden. Das Hauptziel dieser Arbeit ist es, Mechanismen zum Hinzufügen und Entfernen von Replikaten zur Laufzeit zu entwickeln und zu implementieren, um eine flexible Anzahl Replikate verwalten zu können, und diese Mechanismen in das *dynamic General Structured Voting*-Verfahren zu integrieren.
Bisherige dynamische Replikationsverfahren weisen ein homogene Dynamik in dem Sinne auf, als dass die gleiche Strategie für alle Replikatanzahlen verwendet wird. Ein Mechanismus, der die Benutzung verschiedener Strategien für verschiedene Knotenanzahlen erlaubt und somit zu einer inhomogenen Strategie führt, wird vorgestellt und in Hinblick auf die Operationsverfügbarkeiten im Vergleich zu homogenen Strategien verglichen.

Inhaltsverzeichnis

1 Einleitung **1**

2 Replikationsverfahren **3**
2.1 Grundlagen . . . 3
2.1.1 Quorum-Strukturen . . . 5
2.1.2 Transaktionen und Commit-Protokolle . . . 7
2.2 Statische Verfahren . . . 11
2.2.1 Unstrukturierte Verfahren . . . 11
2.2.2 Strukturbasierte Verfahren . . . 13
2.2.3 Generalisierung statischer Verfahren . . . 15
2.3 Dynamische Verfahren . . . 18
2.3.1 Unstrukturierte Verfahren . . . 20
2.3.2 Strukturbasierte Verfahren . . . 22
2.3.3 Generalisierung dynamischer Verfahren . . . 23
2.4 Zusammenfassung . . . 26

3 Ein adaptives, dynamisches und generalisiertes Voting-Verfahren **27**
3.1 Generierung von Voting-Strukturen . . . 27
3.2 Durchführung von Operationen . . . 32
3.2.1 Durchführung der Leseoperation . . . 33
3.2.2 Durchführung der Schreiboperation . . . 34
3.2.3 Durchführung der Aktualisierungsoperation . . . 35
3.2.4 Durchführung der Epochenwechseloperation . . . 36
3.3 Hinzufügen und Entfernen von Knoten . . . 38
3.3.1 Hinzufügen eines Knotens . . . 38
3.3.2 Entfernen eines Knotens . . . 38
3.4 Emulation des dynamic General Structured Voting-Verfahrens . . . 39
3.5 Erweiterungen . . . 41
3.5.1 Unterstützung für blockweisen Lese- und Schreibzugriff . . . 41
3.5.2 Unterstützung paralleler Leseoperationen . . . 42
3.5.3 Epochenverwaltung durch Epochenüberwachungsknoten . . . 44
3.5.4 Verzögerung des Epochenwechsels nach dem Hinzufügen oder Entfernen eines Knotens . . . 45
3.5.5 Überlauf der Versions- bzw. Epochennummer . . . 46
3.5.6 Generalisierung von Strukturgeneratoren . . . 46
3.6 Zusammenfassung . . . 48

4 Architektur und Realisierung des Rahmenwerkes **49**
4.1 Technologien . . . 49
4.2 Konzeption, Entwurf und Komponenten . . . 50
4.3 Erweiterungen . . . 56

4.4 Zusammenfassung 56

5 Simulation **57**
5.1 Simulationsverfahren und Voraussetzungen 57
5.2 Messergebnisse vorgestellter Verfahren mit jeweils neun Knoten 58
5.2.1 Dynamic Voting-Verfahren 59
5.2.2 dynamic Grid Protocol-Verfahren 62
5.2.3 Tree Quorum Protocol-Verfahren 65
5.2.4 adaptive dynamic General Structured Voting-Verfahren 68
5.2.5 direkter Vergleich der Verfahren 71
5.3 Beispielszenario und Messergebnisse des Adaptivitätsaspektes 73
5.4 Zusammenfassung 79

6 Zusammenfassung und Ausblick **80**

Literaturverzeichnis **81**

Glossar **84**

Abbildungsverzeichnis

2.1 Zielkonflikt von Replikationsverfahren 4
2.2 Klassifikationsspektrum von Replikationsverfahren 4
2.3 Zwei Subnetzwerke A und B, die nach dem Ausfall des Verbindungsrechners voneinander isoliert sind 5
2.4 Beispiel eines Mehrheitsverfahrens mit fünf Knoten 7
2.5 Statusdiagramm des 2 Phase Commit-Protokolls 9
2.6 Statusdiagramm des 3 Phase Commit-Protokolls 10
2.7 Tree Quorum Protocol-Verfahren mit 13 Knoten 13
2.8 4×3 Grid Protocol-Verfahren mit 12 Knoten 14
2.9 Voting-Struktur des ROWA-Verfahrens mit fünf Knoten 16
2.10 Voting-Strukturen vorgestellter Replikationsverfahren 17
2.11 Voting-Strukturen eines einfachen Mehrheitsverfahrens mit sieben und drei Knoten 18
2.12 Notwendigkeit, einen Epochenwechsel mit einem Schreibquorum der alten und der neuen Epoche durchzuführen 20
2.13 Ableitung der CVS mit vier Knoten aus der MVS für sechs Knoten wegen des Aufalls zweier Knoten 25

3.1 Register und Zuordnung von Strukturgeneratoren zu Registereinträgen . . . 28
3.2 Voting-Struktur des Majority Consensus Voting-Verfahrens mit fünf physikalischen Knoten 32
3.3 Emulation des dGSV-Verfahrens 40
3.4 Voting-Struktur des Majority Consensus Voting-Verfahrens mit fünf physikalischen Knoten 43
3.5 Integration des generischen Strukturgenerators in das adGSV-Verfahren . . . 47

4.1 Klassendiagramm des Rahmenwerkes 51

5.1 Resultierende durchschnittliche Zeit bis zum Ausfall eines Knotens und durchschnittliche Anzahl ausgeführter oder daran teilgenommener Operationen vor dem Ausfall eines Knotens in Abhängigkeit vom Verfügbarkeitsgrad des Knotens 58
5.2 Durchschnittliche Anzahl verfügbarer Knoten und gemessene Knotenverfügbarkeit in Abhängigkeit der vorgegebenen Knotenverfügbarkeit 59
5.3 Messergebnisse des Dynamic Voting-Verfahrens mit neun Knoten 60
5.4 Voting-Strukturen des Dynamic Voting-Verfahrens für einen bis neun Knoten 61
5.5 Zwei mögliche Knotenanordnungen für das GP-Verfahren mit 6 Knoten . . . 62
5.6 Anzahl der generierten Voting-Strukturen für die Knotenanzahlen eins bis neun für die Knotenverfügbarkeiten 0.4 und 0.5 62
5.7 Messergebnisse des dynamic Grid Protocol-Verfahrens mit neun Knoten . . . 63

5.8 Voting-Strukturen des dynamic Grid Protocol-Verfahrens für einen bis neun Knoten . . . 64
5.9 Messergebnisse des Tree Quorum Protocol-Verfahrens mit neun Knoten . . . 66
5.10 Voting-Strukturen des Tree Quorum Protocol-Verfahrens für einen bis neun Knoten . . . 67
5.11 Messergebnisse des adaptive dynamic General Structured Voting-Verfahrens mit neun Knoten . . . 69
5.12 Voting-Strukturen des adaptive dynamic General Structured Voting-Verfahrens für einen bis neun Knoten . . . 70
5.13 Vergleich der Leseoperationsverfügbarkeiten in Abhängigkeit der Knotenverfügbarkeit . . . 71
5.14 Vergleich der Schreiboperationsverfügbarkeiten in Abhängigkeit der Knotenverfügbarkeit . . . 72
5.15 Vergleich der Operationsverfügbarkeit in Abhängigkeit der Knotenverfügbarkeit bei einem Operationsverhältnis von 80% Leseoperationen und 20% Schreiboperationen . . . 72
5.16 Knotenanordnung des dGP-Verfahrens mit fünfzehn Knoten . . . 73
5.17 Graph der Leseoperationsverfügbarkeiten des dGP(15)-Verfahrens, des dGP-(09)-Verfahrens und des adGSV-Verfahrens in Abhängigkeit der Knotenverfügbarkeit . . . 74
5.18 Graph der durchschnittlichen Leseoperationskosten des dGP(15)-Verfahrens, des dGP(09)-Verfahrens und des adGSV-Verfahrens in Abhängigkeit der Knotenverfügbarkeit . . . 75
5.19 Tabelle der Leseoperationsverfügbarkeiten und der durchschnittlichen Leseoperationskosten des dGP(15)-Verfahrens, des dGP(09)-Verfahrens und des adGSV-Verfahrens . . . 75
5.20 Graph der Schreiboperationsverfügbarkeiten des dGP(15)-Verfahrens, des dGP(09)-Verfahrens und des adGSV-Verfahrens in Abhängigkeit der Knotenverfügbarkeit . . . 76
5.21 Graph der durchschnittlichen Schreiboperationskosten des dGP(15)-Verfahrens, des dGP(09)-Verfahrens und des adGSV-Verfahrens in Abhängigkeit der Knotenverfügbarkeit . . . 77
5.22 Tabelle der Schreiboperationsverfügbarkeiten und durchschnittlichen Schreiboperationskosten des dGP(15)-Verfahrens, des dGP(09)-Verfahrens und des adGSV-Verfahrens . . . 77
5.23 Voting-Strukturen des Grid Protocol-Verfahrens für zehn bis fünfzehn Knoten 78

1 Einleitung

Der Einsatz von Replikationstechniken ermöglicht eine höhere Verfügbarkeit kritischer Daten gegenüber zentralen Ansätzen. Bei zentralen Ansätzen ist die Verfügbarkeit der Daten durch den Ausfall des Rechners oder des Kommunikationsweges zu dem Rechner, der die Daten besitzt, nicht mehr gegeben. Wenn die Daten dagegen auf mehreren Rechnern vorhanden sind, dann ist der Zugriff auf eine Kopie der Daten auf einem anderen Rechner in der Regel weiterhin möglich und die Daten bleiben verfügbar. Die Kopien der Daten werden Replikate genannt. Die Aufgabe eines Replikationsverfahrens ist die gegenseitige Konsistenz der Replikate im Sinne eines vorgegebenen Korrektheitskriteriums sicherzustellen und dazu konsistenzerhaltende Strategien zum lesenden und schreibenden Zugriff auf die Replikate bereitzustellen. Das Replikationsverfahren soll möglichst geringe Lese- und Schreiboperationskosten auf der einen, und auf der anderen Seite eine möglichst hohe Verfügbarkeit der kritischen Daten und somit auch eine möglichst hohe Verfügbarkeit der Operationen darauf gewährleisten. Darüberhinaus ermöglichen Replikationsverfahren die Lastverteilung des Zugriffs auf die Daten auf mehrere Rechner. In zentralen Ansätzen existiert nur ein Rechner, der die Daten vorhält und jede Operation ausführen muss. Dieser Rechner kann leicht zum Engpass werden, wenn die Belastung durch zu viele Operationen die Kapazität des Rechners übersteigt und Operationen deshalb nicht mehr ausgeführt werden können oder verzögert werden. Wenn mehrere Rechner Replikate besitzen, dann kann potenziell jeder dieser Rechner Operationen ausführen. Die Belastung durch die Ausführung von Operationen wird auf mehrere Rechner verteilt und der Engpass behoben.
Statische Replikationsverfahren basieren auf einer fixen Anzahl Replikate, die zu Beginn des Verfahrens festgelegt wird und während der Laufzeit nicht anpassbar ist, wodurch auf Ausfälle von Rechnern oder Kommunikationswege nicht reagiert werden kann. Wenn mehr Rechner ausgefallen sind als die vom Replikationsverfahren verwendete Strategie zur Durchführung von Lese- oder Schreiboperationen tolerieren kann, dann ist bei Rechnerausfällen der Zugriff auf die Daten nicht mehr möglich. Dynamische Verfahren überwinden diese Einschränkung durch die Adaptionfähigkeit an die neue Anzahl Replikate, indem die Strategie an die neue Replikatanzahl angepasst wird. Viele dynamische Verfahren weisen ebenso wie die statischen Verfahren eine Obergrenze verwaltbarer Replikate auf, die zu Beginn festgelegt wird und nicht veränderbar ist. Die dynamische Anpassung der Strategie findet nur im Rahmen von einem Replikat bis zur Obergrenze statt. Das Hinzufügen von Replikaten zur Laufzeit über diese Obergrenze hinaus und das Entfernen von Replikaten zur Laufzeit ist entweder nicht möglich oder spezifisch für das jeweilige Verfahren und nicht allgemeingültig.
Im Laufe der Zeit wurden viele Replikationsverfahren im Hinblick auf unterschiedlichste Anwendungsszenarien entwickelt. Jedes Replikationsverfahren bringt spezifische Vor- und Nachteile mit sich, kein Verfahren ist jedoch für alle Anwendungsszenarien gleichermaßen gut geeignet. Bei sich ändernden Anforderungen des Anwendungsszenarios an das Replikationsverfahren muss das Verfahren manuell den neuen Anforderungen angepasst werden, was kostenintensiv, aufwendig und fehlerträchtig ist. Generalisierende Verfahren ermöglichen durch die Trennung der Strategie des Verfahrens von der Implementation das Anpassen der Strategie ohne die Implementation verändern zu müssen.

Das Hauptziel dieser Arbeit ist es, einen Mechanismus zum Hinzufügen und Entfernen von Replikaten zu entwickeln und zu implementieren, um eine zur Laufzeit flexible Anzahl Replikate verwalten zu können, und diesen Mechanismus in das dynamische generalisierende *Dynamic General Structured Voting*-Verfahren zu integrieren.
Bisherige dynamische Replikationsverfahren weisen ein homogene Dynamik in dem Sinne auf, als dass immer die gleiche Basisstrategie für alle Replikatanzahlen verwendet wird. Wenn z.B. das *Dynamic Voting*-Verfahren benutzt wird, dann wird nach dem Ausfall oder der Wiederherstellung von ausgefallenen Rechnern auch das *Dynamic Voting*-Verfahren für die neue Replikatanzahl benutzt. Das zweite Ziel dieser Arbeit ist es, einen Mechanismus zu entwickeln und ebenfalls in das *Dynamic General Structured Voting*-Verfahren zu integrieren, der die Benutzung verschiedener Strategien für verschiedene Replikatanzahlen erlaubt und somit zu einer inhomogenen Strategie führt.

In Kapitel 2 werden zuerst gundlegende Definitionen, Terminologien und Methoden erläutert. Daran anschließend werden exemplarisch einige statische und dynamische Replikationsverfahren in den Abschnitten 2.2 und 2.3 vorgestellt. Jeder Abschnitt schließt mit mit der Vorstellung eines generalisierenden Verfahrens der jeweiligen Gattung. In Kapitel 3 wird das Verfahren vorgestellt, das eine zur Laufzeit flexible Anzahl Replikate verwalten kann und dafür einen Mechanismus zum Hinzufügen und Entfernen von Replikaten bereitstellt. Auf das Verfahren zur Benutzung einer inhomogenen Strategie wird ebenfalls eingegangen. Die Implementation in Form eines Rahmenwerkes wird konzeptionell in Kapitel 4 beschrieben. Kapitel 5 präsentiert die Simulationsergebnisse des in Kapitel 3 beschriebenen Verfahrens im Vergleich zu anderen vorgestellten Verfahren. Mit einer Zusammenfassung und Darstellung noch offener Arbeiten, zu finden in Kapitel 6, schließt die vorliegende Arbeit.

2 Replikationsverfahren

Dieses Kapitel dient als Einführung in die Thematik der Replikationsverfahren. Im folgenden Abschnitt wird grundlegende Terminologie definiert und auf die Voraussetzungen für Replikationsverfahren, wie Commit-Protokolle und Quorum-Strukturen, eingegangen. In den darauf folgenden beiden Abschnitten 2.2 und 2.3 werden zuerst statische und dann dynamische Replikationsverfahren vorgestellt. Sowohl für statische als auch für dynamische Verfahren werden unstrukturierte und strukturbasierte Verfahren behandelt. Beide Abschnitte werden mit der Vorstellung eines generalisierenden Verfahrens abgeschlossen.

2.1 Grundlagen

Ein *verteiltes System* besteht aus einer Menge von voneinander unabhängigen Rechnern, die über ein Netzwerk miteinander verbunden sind und durch Nachrichtenaustausch miteinander kommunizieren können. Die Rechner eines verteilten Systems werden *Knoten* genannt. Alle Knoten verhalten sich *FailStop* [SS83], d.h. ein Knoten ist entweder ausgefallen und nicht verfügbar oder voll funktionsfähig und verfügbar. Ein *Replikat* ist eine Kopie eines Datums, die auf verschiedenen Knoten eines verteilten Systems abgelegt ist. Alle Knoten, die ein Replikat besitzen, nehmen am Replikationsverfahren für das Replikat teil. Die Aufgabe des *Replikationsverfahrens* ist die gegenseitige Konsistenz der Replikate im Sinne eines vorgegebenen Korrektheitskriteriums sicherzustellen und dazu konsistenzerhaltende Strategien zum lesenden und schreibenden Zugriff auf die Replikate bereitzustellen. Dabei sollen geringe Lese- und Schreiboperationskosten auf der einen, und möglichst hohe Operationsverfügbarkeiten auf der anderen Seite erreicht werden. Eine Lese- oder Schreiboperation ist *verfügbar*, wenn die Ausführung der Operation möglich ist und die gegenseitige Konsistenz der Replikate nach der Ausführung der Operation weiterhin gegeben ist. In einem nicht idealen Netzwerk, in dem die Verwendung von Netzwerkverbindungen Kosten verursacht, errechnen sich die *Operationskosten* aus der Anzahl benötigter Nachrichten und den verwendeten Netzwerkverbindungen für den Nachrichtenaustausch, wobei verschiedene Netzwerkverbindungen unterschiedliche Kosten verursachen können. Die Operationskosten in einem idealen Netzwerk ergeben sich nur aus der Anzahl benötigter Nachrichten.
In Abbildung 2.1 sind die komplementären Anforderungen an ein Replikationsverfahren dargestellt. Für hohe Operationsverfügbarkeit und geringe Leseoperationskosten muss es eine große Anzahl Replikate geben, auf die wahlfrei zugegriffen werden kann. Geringe Schreiboperationskosten lassen sich nur durch eine kleine Anzahl an Replikaten, von denen möglichst wenige synchron aktualisiert werden müssen, erreichen. Dagegen dienen viele Replikate, die synchron zu aktualisieren sind, der Erhaltung der Datenkonsistenz. Replikationsverfahren implementieren einen Kompromiss zwischen diesen nicht zu vereinbarenden Anforderungen. Je nach Anwendungsszenario wird dabei auf einen Aspekt mehr oder weniger Wert gelegt. In Abbildung 2.2 ist das Klassifikationsspektrum für Replikationsverfahren dargestellt. Die erste Dimension erstreckt sich zwischen Verfügbarkeit und Konsistenz. Die jeweiligen Extreme sind optimistische und pessimistische Replikationsverfahren. Die zweite Dimension beinhaltet den Korrektheitsbegriff und reicht von syntaktisch bis semantisch. Optimistische Repli-

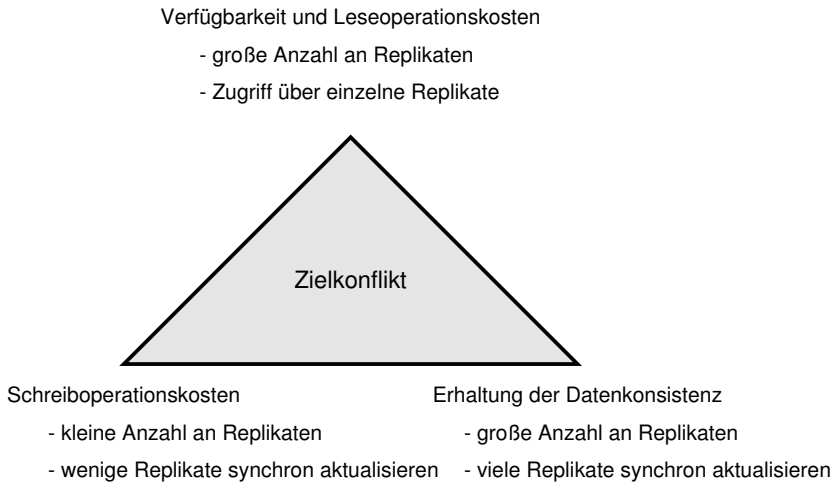


Abbildung 2.1: Zielkonflikt von Replikationsverfahren

kationsverfahren legen mehr Wert auf hohe Verfügbarkeit und geringe Operationskosten zu Lasten der Datenkonsistenz. Dateninkonsistenzen werden nicht verhindert sondern bei Eintritt erkannt und aufgelöst, um wieder einen global konsistenten Zustand herzustellen, was im Allgemeinen durch kompensierende Transaktionen geschieht. Der optimistische Charakter äußert sich in der Annahme dass Dateninkonsistenzen eher selten auftreten und dann aufgelöst werden können. In [Her90] und [SS03] wird ein Überblick über optimistische Replikationsverfahren gegeben. Pessimistische Replikationsverfahren verhindern Dateninkonsistenzen zu Lasten der Verfügbarkeit und Operationskosten. Der pessimistische Charakter äußert sich in der Annahme dass Dateninkonsistenzen auftreten werden, wenn sie auftreten können. Syntaktische Ansätze benutzen *1-Kopien-Serialisierbarkeit (1SR)* [BHG87] als

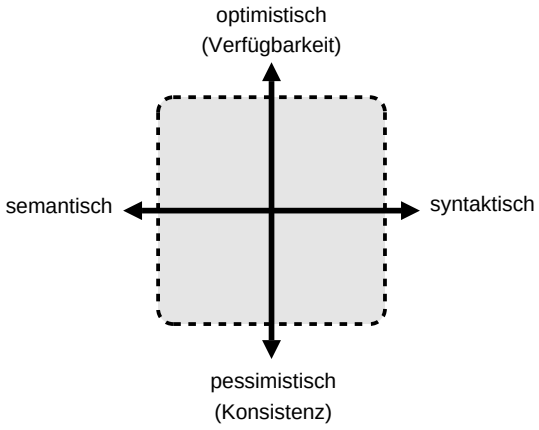


Abbildung 2.2: Klassifikationsspektrum von Replikationsverfahren

Korrektheitskriterium, d.h. die Ausführung einer Operation auf einem replizierten Datum gleicht der Ausführung derselben Operation auf einem nicht replizierten Datum. Insbesondere gibt eine Leseoperation den zuletzt geschriebenen Wert des Datums zurück. Semantische Ansätze benutzen die Semantik der replizierten Daten als Korrektheitskriterium, d.h. bestimmte Eigenschaften des replizierten Datums werden zur Definition des Korrekt-

heitsbegriffs ausgenutzt. Im Rahmen dieser Arbeit werden nur pessimistisch-syntaktische Replikationsverfahren betrachtet.

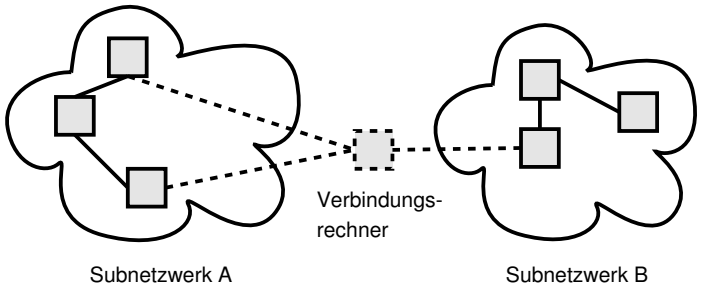


Abbildung 2.3: Zwei Subnetzwerke A und B, die nach dem Ausfall des Verbindungsrechners voneinander isoliert sind

Wenn ein Knoten ausfällt, der als Verbindungsrechner (*Gateway*) zwischen zwei Subnetzwerken fungiert, dann teilt sich das Netzwerk in zwei voneinander isolierte Subnetzwerke (*Partitionen*) [DGMS85, JM90]. Zwischen den Knoten zweier Partitionen ist keine Kommunikation möglich, innerhalb einer Partition ist die Kommunikation der Knoten untereinander jedoch nicht beeinträchtigt. Insbesondere für die Erhaltung der Datenkonsistenz ist Partitionierung problematisch. Eine replizierte Datenbank enthält z.B. Kontostände der Kunden einer Bank. Das Netzwerk wird, wie in Abbildung 2.3 dargestellt, durch den Ausfall des Verbindungsrechners in zwei Partitionen A und B geteilt. Während der Teilung werden zwei Buchungen auf einem Konto mit 900€ Guthaben ausgeführt. Die erste Buchung wird von einem Knoten in der Partition A bearbeitet und schreibt dem Konto 100€ gut. Die zweite Buchung, die von einem Knoten in der Partition B bearbeitet wird, bucht 100€ ab. Für die erste Partition ergibt sich ein Saldo von 1000€, während in der zweiten Partition der Kontostand 800€ beträgt. Der global korrekte Kontostand ist jedoch 900€. Replikationsverfahren müssen solche Inkonsistenzen verhindern, indem entweder zu Lasten der Verfügbarkeit in keiner Partition eine Buchung zugelassen wird oder Buchungen finden exklusiv in einer der beiden Partitionen statt.

Im folgenden Abschnitt wird das Konzept der Quoren bzw. Coterien vorgestellt, das von allen in dieser Arbeit vorgestellten Replikationsverfahren direkt oder indirekt benutzt wird.

2.1.1 Quorum-Strukturen

Von zentraler Bedeutung für Replikationsverfahren ist die Konsistenzerhaltung der Replikate. Schreiboperationen müssen daher unter gegenseitigem Ausschluss und unter Ausschluss von Leseoperationen stattfinden. Leseoperationen dürfen parallel bzw. zeitlich überlappend ausgeführt werden, da Leseoperationen nicht modifizierend und somit nicht konsistenzgefährdend sind. Zur Erfüllung dieser Bedingungen werden Quoren [AE92] bzw. Coterien [GMB85, MN92] benutzt.

Quorum Sei U die nicht leere Menge aller Knoten $R_1, \ldots, R_n$. Eine Menge von Mengen Q ist eine *Quorenmenge* unter U, wenn gilt:

$$\forall\, G \in Q : G \neq \emptyset \wedge G \subseteq U \qquad (Q1)$$

$$\forall\, G, H \in Q : G \not\subset H \qquad (Q2)$$

Jede Teilmenge $G \in Q$ ist ein *Quorum*.

Ein gültiges Quorum darf gemäß der Bedingung $(Q1)$ nicht leer sein und muss aus am Replikationsverfahren teilnehmenden Knoten bestehen. Ausserdem darf ein gültiges Quorum gemäß der Bedingung $(Q2)$ keine echte Teilmenge eines anderen Quorums sein.

Coterie Eine Quorenmenge Q ist eine *Coterie* unter U, wenn die Überschneidungseigenschaft

$$\forall\, G, H \in Q : G \cap H \neq \emptyset \qquad (C1)$$

gegeben ist.

Je zwei Quoren aus der Quorenmenge Q, die die Überschneidungseigenschaft $(C1)$ erfüllen, enthalten mindestens einen gemeinsamen Knoten. Wenn alle Quorenpaare aus der Quorenmenge Q die Überschneidungseigenschaft $(C1)$ erfüllen, dann ist die Quorenmenge Q eine Coterie.

Jedem Knoten $R_x \in U$ wird eine positive Ganzzahl als *Stimme* zugeordnet, die für die Durchführung einer Operation abgegeben werden kann. Wenn ein Knoten die Stimme abgegeben hat, dann darf die Stimme bis zum Abschluss der Operation nicht nochmals vergeben werden. Danach darf der Knoten die Stimme wieder zur Durchführung einer Operation abgeben. Indem die Stimme nicht abgegeben wird verweigert ein Knoten die Zustimmung zur Durchführung der Operation. Jedes Replikationsverfahren impliziert durch die verwendete Strategie, der Anzahl am Replikationsverfahren teilnehmenden Knoten und die den Knoten zugewiesenen Stimmen eine Lesequorenmenge $Q_r = \{Q_{r_1}, \ldots, Q_{r_n}\}$ und eine Schreibquorenmenge $Q_w = \{Q_{r_1}, \ldots, Q_{r_n}\}$. Für die Durchführung einer Leseoperation muss der die Operation initiierende Knoten die Stimmen aller Knoten eines *Lesequorums* aus der Lesequorenmenge erhalten. Entsprechend muss der eine Schreiboperation initiierende Knoten für die Durchführung der Operation die Stimmen aller Knoten eines *Schreibquorums* aus der Schreibquorenmenge erhalten. Die Schreibquorenmenge muss eine Coterie sein, damit zu jedem Zeitpunkt nur ein Schreibquorum existieren kann, in dem alle darin enthaltenen Knoten der Durchführung der Schreiboperation zugestimmt haben. Durch die Überschneidungseigenschaft $(C1)$ besitzen je zwei Schreibquoren mindestens einen gemeinsamen Knoten, der zu einem Zeitpunkt die Stimme nur zur Durchführung einer der beiden Schreiboperationen abgeben kann. Der gegenseitige Ausschluss von Schreiboperationen wird auf diese Weise gewährleistet. Jedes Quorum aus der Lesequorenmenge muss mit jedem Quorum aus der Schreibquorenmenge eine Coterie darstellen, um zum einen den gegenseitigen Ausschluss von Schreib- und Leseoperationen zu gewährleisten und zum anderen zu garantieren dass in jedem Lesequorum mindestens ein Knoten enthalten ist, der an der letzten Schreiboperation beteiligt war und somit das Lesequorum mindestens einen Knoten mit einem aktuellen Replikat enthält.

In Abbildung 2.4 sind fünf Knoten dargestellt. Jedem Knoten ist die Stimme eins zugeordnet. Das Replikationsverfahren benutzt eine einfache Mehrheitsstrategie, bei der zur Durchführung einer Lese- oder Schreiboperation die Zustimmung einer Menge von Knoten nötig ist, deren Stimmensumme mindestens der einfachen Mehrheit der Summe aller am Re-

plikationsverfahren beteiligten Knoten erreicht. Die minimale Stimmensumme der Knoten eines Lese- oder Schreibquorums wird *Mindeststimmenanzahl* genannt. Die Mindeststim-

Abbildung 2.4: Beispiel eines Mehrheitsverfahrens mit fünf Knoten

menanzahl der Knoten jedes Lese- und Schreibquorums für die einfache Mehrheitsstrategie mit fünf Knoten entspricht drei Stimmen. Daraus ergibt sich die Lese- bzw. Schreibquorenmenge $Q_{r,w} = \{\{R1, R2, R3\}, \{R1, R2, R4\}, \{R1, R2, R5\}, \{R1, R3, R4\}, \{R1, R3, R5\}, \{R1, R4, R5\}, \{R2, R3, R4\}, \{R2, R3, R5\}, \{R2, R4, R5\}, \{R3, R4, R5\}\}$. Für je zwei Quoren $G, H \in Q_{r,w}$ gilt:

- beide Quoren sind nicht leer, bestehen aus am Replikationsverfahren teilnehmenden Knoten und erfüllen somit die Bedingung $(Q1)$,
- beide Quoren sind sind keine echte Teilmenge des jeweils anderen Quorums und erfüllen somit die Bedingung $(Q2)$ und
- beide Quoren enthalten gemäß der Bedingung $(C1)$ mindestens einen gemeinsamen Knoten.

Schreiboperationen finden unter gegenseitigem Ausschluss und unter Ausschluss von Leseoperationen statt. Allerdings schließen sich auch je zwei Leseoperationen gegenseitig aus. Die Konsistenz der Daten ist unter Verwendung der Lese- und Schreibquorenmenge $Q_{r,w}$ gewahrt. Wenn z.B. ein neuer Wert auf die Replikate der Knoten des Schreibquorums $\{R3, R4, R5\}$ geschrieben wird, dann kann dieser neue Wert unter Verwendung eines beliebigen Lesequorums aus der Lesequorenmenge von mindestens einem Knoten gelesen werden.

Der folgende Abschnitt behandelt das Konzept der Transaktionen und Commit-Protokolle, das insbesondere für die Ausführung konsistenzerhaltender Schreiboperationen unentbehrlich ist.

2.1.2 Transaktionen und Commit-Protokolle

Für die Durchführung konsistenzerhaltender Operationen ist die atomare Ausführung der Operation auf den Knoten des entsprechenden Quorums unentbehrlich. Insbesondere nach Abschluss einer Schreiboperation müssen die Replikate der im Schreibquorum enthaltenen Knoten alle den gleichen Wert besitzen. Dazu wird das Konzept der Transaktionen und Commit-Protokolle benutzt.

Transaktion Eine Transaktion ist eine endliche Folge von Leseoperationen `x = r(A)` und Schreiboperationen `w(A,x)` auf einem Objekt A, die das Objekt von einem konsistenten Zustand M in einen konsistenten Folgezustand M' unter Berücksichtigung des *ACID* Prinzips überführt. Die letzte Operation einer Transaktion muss entweder die Anwendung der Operationen auf dem Objekt anweisen (*commit*) oder alle Operationen verwerfen und die Transaktion abbrechen (*abort*). Formal:

$$T = p_1, \ldots, p_n, t \text{ mit } p_i \in \{\texttt{x=r(A)}, \texttt{w(A,x)}\} \wedge t \in \{\texttt{commit}, \texttt{abort}\}$$

Das ACID Prinzip besteht aus den vier Bedingungen Atomizität, Konsistenz, Isolation und Dauerhaftigkeit:

Atomicity (Atomizität) Eine Transaktion wird entweder ganz oder gar nicht ausgeführt. Tritt während der Ausführung einer Transaktion ein Fehler auf, der die ordnungsgemäße Fortführung verhindert, dann werden sämtliche evtl. bereits erfolgten Änderungen der Transaktion zurückgesetzt (*Rollback / kompensierende Transaktion*).

Consistency (Konsistenz / Integritätserhaltung) Vor Beginn und nach Beendigung einer Transaktion befindet sich das Objekt in einem nicht notwendigerweise unterschiedlichen aber konsistenten Zustand. Jede Transaktion einzeln betrachtet erhält die Konsistenz des Objekts. Die serielle Hintereinanderausführung mehrerer Transaktionen erhält somit ebenfalls die Konsistenz. Die Abfolge der Operationen (*Schedule*) einer Transaktionen muss äquivalent zu einer Abfolge sein, die sich bei serieller Hintereinanderausführung der Operationen der Transaktionen ergäbe. Je nach Definition der Äquivalenz resultiert daraus eine unterschiedliche Semantik der Serialisierbarkeit. Wenn jede Transaktion serialisierbar ist, dann ist die Konsistenz gesichert.

Isolation Zwei um dieselben Ressourcen konkurrierende parallele Transaktionen dürfen nicht überlappen und insbesondere nicht das gleiche Datum parallel ändern. Jede seriell ausgeführte Transaktion darf somit nur Endergebnisse einer vorherigen Transaktion lesen. Diese Einschränkung bedeutet einen Zielkonflikt zwischen Performanz und Isolation.

Durability (Dauerhaftigkeit / Persistenz) Die Wirkung einer erfolgreich abgeschlossenen Transaktion bleibt dauerhaft erhalten. Nur eine weitere Transaktion kann das Datum ändern.

Ebenso wie auf rechnerlokaler Ebene muss eine Transaktion in einem verteilten System dem ACID-Prinzip Folge leisten. Dazu ist die Synchronisation und Koordination der beteiligten Knoten durch *Commit-Protokolle* nötig. Ein Knoten, der den Ablauf einer verteilten Transaktion kontrolliert, wird *Koordinator* genannt. Alle anderen Knoten werden *Teilnehmer* genannt. Ein Koordinator kann auch gleichzeitig Teilnehmer sein.

Das 2 Phase Commit-Protokoll Das *2 Phase Commit (2PC)*-Protokoll [Ske81] besteht aus zwei Phasen, einer Wahl- und einer Entscheidungsphase. In Abbildung 2.5 ist das Statusdiagramm des 2PC-Protokolls dargestellt. In Anlehnung an die graphische Notation von endlichen Automaten bezeichnen Kreise die möglichen Zustände des Koordinators und der Teilnehmer. Die Endzutände sind durch eine gestrichelte Umrandung gekennzeichnet. Gerichtete Kanten definieren gültige Zustandsübergänge. Die Kantenattribute stellen empfangene Nachrichten (oben) und gesendete Nachrichten (unten) dar.
In der Wahlphase fragt der Koordinator die Teilnehmer nach deren Zustimmung oder Ablehnung zur Durchführung der Transaktion, indem eine PREPARE-Nachricht an die Teilnehmer verschickt wird, und wartet im Zustand *WAIT* auf die Antworten der Teilnehmer. Wenn ein Teilnehmer die Durchführung ablehnt, dann wechselt er in den Endzustand *ABORT* und sendet die Antwort VOTE: ABORT an den Koordinator zurück. Sonst wechselt er in den Zustand *READY* und sendet die Antwort VOTE: COMMIT an den Koordinator. Nachdem alle Teilnehmer ihre Antwort an den Koordinator gesendet haben ist die Wahlphase beendet. In der anschließenden Entscheidungsphase prüft der Koordinator die Antworten. Wenn

alle Teilnehmer mit VOTE: COMMIT geantwortet haben, dann wird den Teilnehmern die Nachricht GLOBAL COMMIT zum Wechsel in den Endzustand *COMMIT* gesendet und der Koordinator wechselt selbst in den Endzustand *COMMIT*. Die Transaktion ist erfolgreich abgeschlossen. Wenn mindestens ein Teilnehmer mit VOTE: ABORT geantwortet hat, dann muss die Transaktion abgebrochen werden. Der Koordinator sendet jedem Teilnehmer die Nachricht GLOBAL ABORT zum Wechsel in den Endzustand *ABORT* und wechselt selbst in den Endzustand *ABORT*.

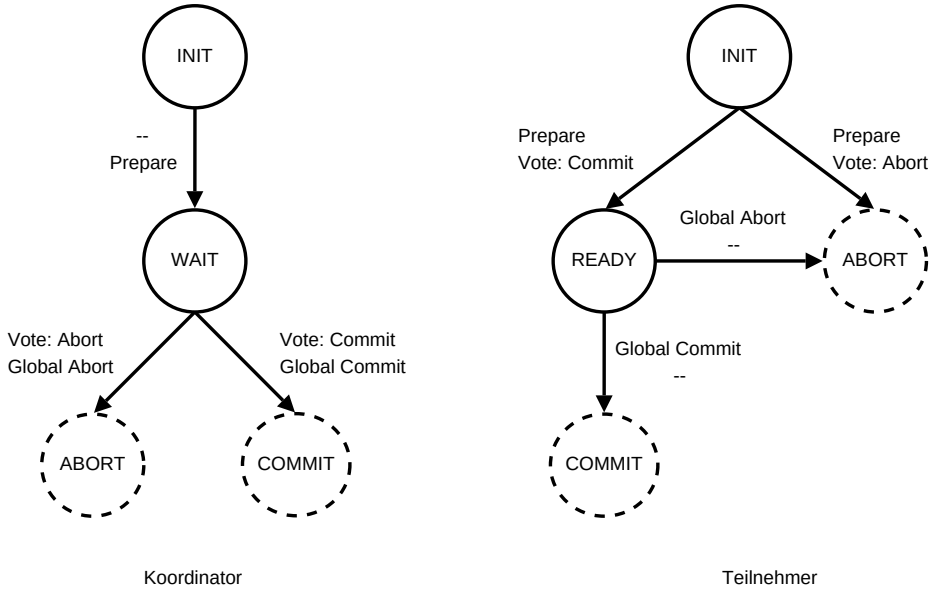


Abbildung 2.5: Statusdiagramm des 2 Phase Commit-Protokolls

Wenn der Koordinator ausfällt während sich die Teilnehmer im Zustand *READY* befinden und auf die Entscheidungsnachricht des Koordinators warten, dann sind alle Teilnehmer solange blockiert bis der Koordinator repariert ist und die Entscheidungsnachricht sendet. Fällt ein Teilnehmer in der Wahlphase aus bevor die Zustimmung oder Ablehnung dem Koordinator mitgeteilt werden kann, dann müssen alle anderen Teilnehmer und der Koordinator auf die Reparatur des ausgefallenen Teilnehmers warten, da nicht eindeutig bestimmt werden kann wie der ausgefallene Teilnehmer entschieden hat.
Das 2PC-Protokoll ist ein *blockierendes* Protokoll, d.h. funktionsfähige Teilnehmer müssen im Fehlerfall auf die Reparatur ausgefallener Teilnehmer warten und sind solange blockiert.

Das 3 Phase Commit-Protokoll Das *3 Phase Commit (3PC)*-Protokoll [Ske81] ist eine Weiterentwicklung des 2PC-Protokolls. Nach der Entscheidungsphase wird eine neue Zwischenphase eingeführt. Der Ablauf des 3PC-Protokolls ist bis zu dieser neuen Phase mit dem 2PC-Protokoll identisch. Nach dem Empfang der Nachrichten VOTE: COMMIT von allen Teilnehmern sendet der Koordinator die Nachricht PREPARE TO COMMIT an alle Teilnehmer. Die Teilnehmer wechseln in den Zustand *PRE COMMIT* und senden READY TO COMMIT an den Koordinator. Sobald der Koordinator alle Antworten der Teilnehmer bekommen hat, sendet er die Nachricht GLOBAL COMMIT zum Wechsel in den Endzustand *COMMIT* an die Teilnehmer und wechselt selbst in den Endzustand *COMMIT*. Die Transaktion ist erfolgreich abgeschlossen.

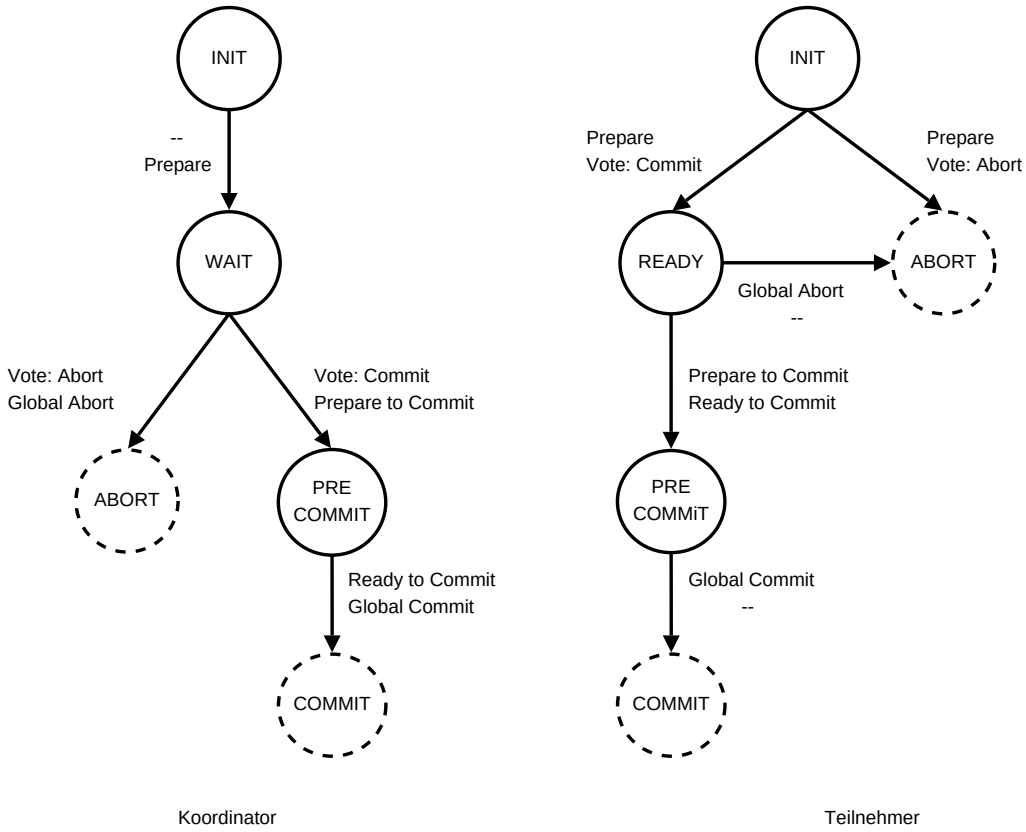


Abbildung 2.6: Statusdiagramm des 3 Phase Commit-Protokolls

In Abbildung 2.6 ist das Statusdiagramm des 3PC-Protokolls dargestellt. Das 3PC-Protokoll ist ein *nicht blockierendes* Protokoll. Durch ein *Termination Protocol* und der *Recovery Procedure* kann die Transaktion auch bei Ausfall einer bestimmten Anzahl Teilnehmer konsistent beendet werden ohne auf die Reparatur ausgefallener Teilnehmer warten zu müssen. Das Termination Protocol führt die Transaktion mit den nicht ausgefallenen Teilnehmern zu Ende, während die Recovery Procedure von ausgefallenen Teilnehmer nach der Reparatur ausgeführt wird. Wenn alle nicht ausgefallenen Teilnehmer sich im Zustand *READY* befinden, dann können sich ausgefallene Teilnehmer nur in den Zuständen *READY*, *ABORT* oder *PRE COMMIT* befinden. Fällt der Koordinator aus und ein Teilnehmer ist im Zustand *PRE COMMIT* oder *COMMIT*, dann wird das Termination Protocol gestartet, der Teilnehmer wird neuer Koordinator und beendet die Transaktion. Zu diesem Zeitpunkt ist bereits die Entscheidung zur Durchführung der Transaktion getroffen.

Im folgenden Abschnitt werden zuerst unstrukturierte statische Replikationsverfahren und danach strukturbasierte statische Replikationsverfahren vorgestellt. Daran anschließend wird ein generalisierendes statisches Verfahren vorgestellt.

2.2 Statische Verfahren

Statische Replikationsverfahren basieren auf einer fixen Lese- und Schreibquorenmenge, die zu Beginn durch die verwendete Strategie und die Anzahl teilnehmender Knoten festgelegt wird. Zur Laufzeit ist weder die Strategie noch die Lese- oder Schreibquorenmenge veränderbar.
In den folgenden beiden Abschnitten werden exemplarisch einige unstrukturierte und daran anschließend einige strukturbasierte statische Verfahren vorgestellt. Abschließend wird ein generalisiertes statisches Verfahren vorgestellt, das alle vorgestellten Replikationsverfahren emulieren kann.
Die Analyse der Kosten und Verfügbarkeiten der folgenden statischen Verfahren ist in [Koc94] zu finden.

2.2.1 Unstrukturierte Verfahren

Ein Lese- oder Schreibquorum unstrukturierter Replikationsverfahren besteht aus einer Submenge der am Replikationsverfahren teilnehmenden Knoten, deren Stimmensumme die Mindeststimmenanzahl für die Durchführung der Lese- oder Schreiboperation erfüllt.

Read One, Write All-Verfahren

Im einfachsten Fall besteht die Strategie des Replikationsverfahrens darin, Schreiboperationen auf allen Replikaten und Leseoperationen auf einem beliebigen Replikat auszuführen. Das einzige Schreibquorum enthält alle am Replikationsverfahren teilnehmenden Knoten, wohingegen jedes Lesequorum aus genau einem Knoten besteht. Konzeptionell besitzt jeder Knoten die Stimmenanzahl eins und die Mindeststimmenanzahl für ein Schreibquorum entspricht der Anzahl der am Replikationsverfahren teilnehmenden Knoten, während die Mindeststimmenanzahl für ein Lesequorum eins ist. Dieses Verfahren wird *Read One, Write All (ROWA)*-Verfahren [AAB96] genannt. Die Leseoperationskosten sind extrem günstig, da nur ein Knoten kontaktiert werden muss. Die Verfügbarkeit von Leseoperationen wird solange nicht beeinträchtigt wie noch mindestens ein Knoten verfügbar ist. Für Schreiboperationen führt bereits der Ausfall eines Knotens zur Verzögerung aller anstehenden Schreiboperationen bis alle Knoten wieder verfügbar sind.
Das *Read One, Write All Available (ROWAA)*-Verfahren [BHG87] erlaubt auch dann Schreiboperationen, wenn Knoten ausgefallen sind, indem Schreiboperationen nur auf allen verfügbaren Knoten ausgeführt werden müssen. Um zu verhindern, dass nach der Reparatur eines ausgefallenen Knotens ein veraltetes Datum von diesem Knoten gelesen wird, müssen ausgefallene Knoten nach der Reparatur ein spezielles Protokoll zur Aktualisierung durchführen. Das ROWAA-Verfahren toleriert Kommunikationsfehler aber keine Partitionierung.
Das *Primary Copy ROWA (PCROWA)*-Verfahren [AAB96] verwendet einen ausgezeichneten Knoten als zentralen Koordinator für alle Operationen. Dieser Knoten wird *Primary Copy-Knoten* genannt. Alle anderen Knoten werden *Backup-Knoten* genannt. Leseoperationen finden nur auf dem Primary Copy-Knoten statt. Schreiboperationen werden auf dem Primary Copy-Knoten und allen erreichbaren Backup-Knoten ausgeführt. Wenn der Primary Copy-Knoten ausfällt, dann wird mittels eines Wahlverfahrens[1] ein neuer Primary Copy-Knoten aus der Menge der Backup-Knoten bestimmt. Die Wahl eines neuen Primary

[1] wie z.B. dem Bully-Algorithmus [Tan92, MMM04]

Copy-Knotens darf nur dann stattfinden, wenn keine Partitionierung vorliegt oder es darf in maximal einer Partition ein neuer Primary Copy-Knoten bestimmt werden. Jeder ausgefallene Knoten muss nach der Reparatur den aktuellen Wert des Replikats vom Primary Copy-Knoten lesen und sich als Backup-Knoten am Replikationsverfahren zurückmelden.

Quorum Consensus-Verfahren

Das ROWA-Verfahren bietet günstige Lese- aber teure Schreiboperationskosten. *Quorum Consensus (QC)*-Verfahren [AAL94] benutzen das in Abschnitt 2.1.1 vorgestellte Konzept der Quoren und ermöglichen dadurch die Balance von Lese- gegenüber Schreiboperationskosten flexibel zu gestalten, indem das Verhältnis der für ein Lese- oder Schreibquorum nötigen Knoten bzw. Stimmenanzahlen variiert wird. Wenn für ein Lesequorum nur ein Knoten und für ein Schreibquorum alle Knoten benötigt werden, dann verhält sich das QC-Verfahren wie das ROWA-Verfahren. Der umgekehrte Fall, in dem ein Lesequorum aus allen Knoten und ein Schreibquorum nur aus einem Knoten besteht, wird *Read All, Write One (RAWO)*-Verfahren genannt. Schreiboperationen in QC-Verfahren schreiben den neuen Wert des Replikats nicht auf alle Knoten, sondern nur auf die Knoten des Schreibquorums, und auch nur diese Knoten haben nach Abschluss der Schreiboperation ein aktuelles Replikat. Leseoperationen lesen den Wert des Replikats von einem Knoten des Lesequorums, der ein aktuelles Replikat besitzt. Es muss ein Indikator für die Aktualität des Replikats eines Knotens eingeführt werden, um bei einer der Schreiboperation nachfolgenden Leseoperation nicht ein veraltetes Datum von einem Knoten des Lesequorums zu lesen, der nicht an der letzten Schreiboperation teilgenommen hat. Das Replikat jedes Knotens wird mit einer Ganzzahl *vn* attributiert, die die *Versionsnummer* des Replikats darstellt. Die Versionsnummer ist initial null und wird bei jeder Schreiboperation, an der der Knoten teilgenommen hat, um eins erhöht. Die aktuelle Versionsnummer vn_{act} entspricht dem Maximum der Versionsnummern aller Replikate. Das Replikat eines Knotens ist genau dann aktuell, wenn dessen Versionsnummer der aktuellen Versionsnummer entspricht. Anhand der Versionsnummer können die Knoten eines Quorums mit aktuellen Replikaten identifiziert werden. Aufgrund der Überschneidungseigenschaft von Lese- und Schreibquoren befindet sich in jedem Lesequorum mindestens ein Knoten, der an der letzten Schreiboperation teilgenommen hat und ein aktuelles Replikat besitzt.

Die Ausprägung des QC-Verfahrens, in der jeder Knoten mit einer Stimme von eins versehen wird und zur Durchführung einer Lese- oder Schreiboperation die einfache Mehrheit aller Knoten bzw. Stimmen benötigt wird, wird *Majority Consensus Voting (MCV)*-Verfahren [AAL94] genannt. Im *Weighted Voting (WV)*-Verfahren [Gif79] werden die Knoten durch eine individuelle Stimmenzuordnung gewichtet. Zur Durchführung einer Lese- oder Schreiboperation wird die einfache Mehrheit aller Stimmen benötigt. Besonders zuverlässige Knoten können bevorzugt und unzuverlässige Knoten entsprechend durch eine geringere Stimmenanzahl benachteiligt werden. Eine günstige Stimmenzuordnung ist schwierig und hängt von verschiedenen Faktoren wie z.B. der Verfügbarkeit des Knotens oder den Kommunikationslatenzen zwischen den einzelnen Knoten ab [GMB85, MN92].

Quorum Consensus-Verfahren sind bezüglich der Schreiboperationsverfügbarkeit nicht so stark von Knotenausfällen betroffen wie ROWA-Verfahren, da die Schreibquorenmenge im Allgemeinen größer ist. Dafür ist die Leseoperationsverfügbarkeit durch größere Lesequoren im Allgemeinen geringer.

2.2.2 Strukturbasierte Verfahren

Strukturbasierte Replikationsverfahren ordnen die daran teilnehmenden Knoten in einer logischen Struktur an. Ein Lese- oder Schreibquorum besteht aus einer Submenge der am Replikationsverfahren teilnehmenden Knoten, die sich in bestimmten Positionen in der Struktur befinden müssen. Nur die Position eines Knotens in der Struktur und nicht dessen Stimmenanzahl wird für die Bildung der Lese- und Schreibquorenmenge berücksichtigt. Die Stimmenanzahl jedes Knotens wird entweder mit eins angenommen oder z.B. zur Positionierung der Knoten in der Struktur benutzt, falls den Knoten individuelle Stimmen zugeordnet wurden.

Tree Quorum Protocol-Verfahren

Das *Tree Quorum Protocol (TQP)*-Verfahren [AA90, AE92] arrangiert die am Replikationsverfahren teilnehmenden Knoten in einer logischen Baumstruktur, die einen ausgewiesenen Wurzelknoten besitzt. Je zwei Knoten sind durch eine gerichtete Kante verbunden. Der Quellknoten der gerichteten Kante wird Vaterknoten und der Zielknoten wird Kindknoten genannt. Knoten ohne Kindknoten werden Blattknoten genannt. Innere Knoten haben einen oder mehrere Kindknoten. Bis auf den Wurzelknoten, der keinen Vaterknoten besitzt, hat jeder Knoten genau einen Vaterknoten. Der Grad d eines Baumes gibt die Anzahl der Kindknoten für jeden inneren Knoten an. In einem kompletten Baum hat jeder innere Knoten exakt d Kindknoten. Andernfalls ist der Baum nicht komplett. Alle mit dem Wurzelknoten durch eine Kante verbundenen Knoten bilden die erste Ebene. Die mit den Knoten ersten Ebene durch eine Kante verbundenen Knoten bilden die zweite Ebene und so fort. Die Höhe h eines Baumes entspricht der Anzahl der Ebenen. Zur Bildung eines Quorums wird innerhalb einer Ebene des Baumes das MCV-Verfahren benutzt und über die Ebenen wird das ROWA-Verfahren benutzt. Lesequoren bestehen aus der Mehrheit von Knoten einer Ebene des Baumes. Schreibquoren bestehen aus der Mehrheit von Knoten aus jeder Ebene des Baumes und überschneiden sich somit mit jedem Lesequorum. In Abbildung 2.7 ist ein Lesequorum z.B. der Wurzelknoten $R1$ oder eine

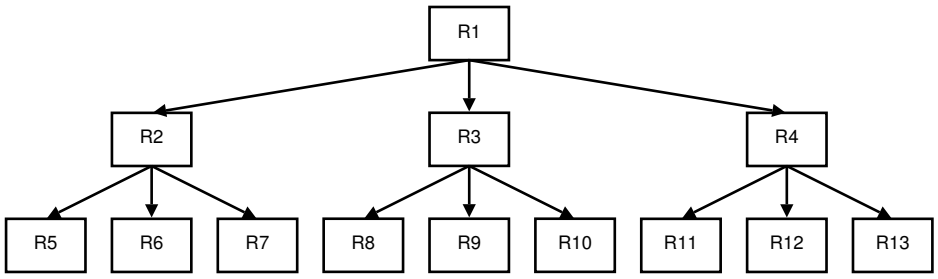


Abbildung 2.7: Tree Quorum Protocol-Verfahren mit 13 Knoten der Höhe $h = 2$ und dem Grad $d = 3$ [AE92]

Mehrheit aus $\{R2, R3, R4\}$ oder $\{R5, R6, R7, R8, R9, R10, R11, R12, R13\}$. Ein Schreibquorum besteht z.B. aus dem Wurzelknoten $R1$ und einer Mehrheit aus $\{R2, R3, R4\}$ und $\{R5, R6, R7, R8, R9, R10, R11, R12, R13\}$.
Eine andere Möglichkeit zur Konstruktion von Quoren ist, ausgehend vom Wurzelknoten, der Struktur des Baumes zu folgen. Schreibquoren bestehen aus dem Wurzelknoten, der Mehrheit seiner Kindknoten, der Mehrheit deren Kindknoten und so weiter. Ein

Lesequorum besteht aus dem Wurzelknoten oder bei Ausfall des Wurzelknotens aus der Mehrheit seiner Kindknoten. Sind auch Kindknoten ausgefallen, dann wird wiederum die Mehrheit deren Kindknoten benötigt und so fort. In Abbildung 2.7 ist z.B. die Menge $\{R1, R2, R3, R5, R6, R8, R9\}$ ein Schreibquorum. Ein Lesequorum besteht z.B. aus dem Wurzelknoten $R1$ oder im Falle des Ausfalls von $R1$ aus der Knotenmenge $\{R2, R3\}$. Sind die Knoten $R1$, $R3$ und $R4$ ausgefallen, dann ist z.B. $\{R2, R8, R9\}$ ein gültiges Lesequorum. Durch Ausnutzen der Struktur verringert sich die Anzahl der der für ein Quorum benötigten Knoten insbesondere für Schreibquoren.
Beide Varianten der Quorenbildung sind insbesondere für Schreibquoren stark von der Verfügbarkeit des Wurzelknotens abhängig, der deshalb ein besonders zuverlässiger Knoten sein sollte. Während Leseoperationen den Ausfall einer bestimmten Anzahl Knoten tolerieren, schlagen Schreiboperationen nach dem Ausfall des Wurzelknotens fehl. Beide Varianten gehen von einem komplett besetztem Baum aus, was die Anwendung des TQP-Verfahrens auf die Knotenanzahlen $1 + \sum_{i=1}^{\infty} d^i$ einschränkt. Indem ein nicht kompletter Baum als kompletter Baum mit fehlenden Subbäumen angesehen wird, lässt das TQP-Verfahren ohne grundsätzliche Veränderungen auch nicht komplette Bäume zu.

Grid Protocol-Verfahren

Das *Grid Protocol (GP)*-Verfahren [AAC90] ordnet die am Replikationsverfahren teilnehmenden Knoten logisch in einem $M \times N$ Gitter an, das aus M Spalten und N Zeilen besteht. Das Problem eines prominenten Knotens, wie es beim TQP-Verfahren der Wurzelknoten ist, wird durch die Gleichberechtigung aller Knoten vermieden. Die Anwendung des GP-Verfahrens ist auf eine Knotenanzahl beschränkt, die sich durch $M * N$ darstellen lässt, da alle Stellen des Gitters mit Knoten besetzt sein müssen. In Abbildung 2.8 ist das GP-Verfahren für zwölf Knoten in einem 4×3 Gitter dargestellt. Bis auf die äußeren Knoten ist

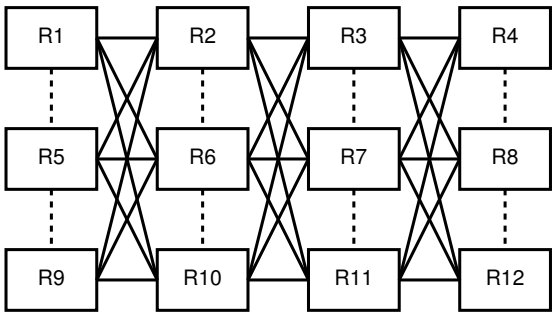


Abbildung 2.8: 4×3 Grid Protocol-Verfahren mit 12 Knoten [Koc94]

jeder Knoten mit allen Knoten seiner beiden Nachbarspalten verbunden. Die Knoten der linken äußeren Spalte sind mit allen Knoten ihrer rechten Nachbarspalte verbunden und die Knoten der rechten äußeren Spalte sind mit allen Knoten ihrer linken Nachbarspalte verbunden. Innerhalb einer Spalte ist bis auf die äußeren Knoten jeder Knoten mit seinen unmittelbaren Nachbarknoten verbunden. Die oberen äußeren Knoten sind nur mit ihren unteren Nachbarknoten verbunden und die unteren äußeren Knoten sind nur mit ihren oberen Nachbarknoten verbunden. Ein Lesequorum besteht aus einem Knoten aus jeder Spalte des Gitters (*C-Cover*). Das entspricht einem horizontalen Weg von links nach rechts durch das in Abbildung 2.8 dargestellte Gitter, der den durchgezogenen Kanten folgt. Die Knotenmenge $\{R1, R6, R7, R12\}$ ist z.B. ein C-Cover. Alternativ kann ein Lesequorum auch aus

den Knoten einer kompletten Spalte bestehen (*CC-Cover*), was eine Erweiterung des ursprünglichen GP-Verfahrens ist. Das entspricht einem Weg von oben nach unten durch das Gitter, der nur den gestichelten vertikalen Kanten folgt. Die Knotenmenge $\{R1, R5, R9\}$ ist z.B. ein CC-Cover. Ein Schreibquorum besteht aus einem C-Cover und einem CC-Cover, wodurch die Überschneidung von Schreib- und Lesequoren in mindestens einem Knoten sichergestellt ist.
Für eine $M \times 1$ Gitterstruktur entsprechen die Operationskosten dem ROWA-Verfahren, wenn ausschließlich CC-Cover, die in diesem Fall aus einem Knoten bestehen, als Lesequoren verwendet werden.

2.2.3 Generalisierung statischer Verfahren

Jedes der bisher vorgestellten statischen Replikationsverfahren hat für bestimmte Anwendungsszenarien Vorteile gegenüber anderen Verfahren, aber keines ist für alle Anwendungsszenarien gleichermaßen gut geeignet [KA01]. Für jedes Anwendungsszenario muss das jeweils beste Verfahren ausgewählt und implementiert werden. Wenn sich die Anforderungen des Anwendungsszenarios an das Replikationsverfahren ändern, muss das Verfahren manuell den neuen Anforderungen angepasst werden, was kostenintensiv, aufwendig und fehlerträchtig ist. Die Trennung der Strategie des Verfahrens von der Implementation ermöglicht das Anpassen der Strategie ohne die Implementation verändern zu müssen.
Das *General Structured Voting (GSV)*-Verfahren [The93a, The93b] stellt ein Rahmenwerk zur Verfügung, das durch die Beschreibung von Strategien in einem vereinheitlichtem Format die Möglichkeit bietet, den Wechsel der Strategie durch den Austausch der Strategiebeschreibung zu erreichen. Die Beschreibung einer Strategie erfolgt durch einen gerichteten azyklischen Graph, der *Voting-Struktur*. Wegen der baumähnlichen Struktur der Voting-Struktur wird die Terminologie für Bäume verwendet. Die Knoten der Voting-Struktur sind *physikalische Knoten* und *virtuelle Knoten*. Ein physikalischer Knoten ist ein Rechner, der ein Replikat besitzt und verwaltet. Ein virtueller Knoten dient als Gruppierungsinstrument für physikalische Knoten, ist kein physikalischer Rechner und besitzt auch kein Replikat. Je zwei Knoten sind durch eine gerichtete Kante verbunden. Der Quellknoten der gerichteten Kante wird Vaterknoten und der Zielknoten wird Kindknoten genannt. Der einzige Knoten in der Voting-Struktur, der keine eingehenden Kanten besitzt, wird als Wurzelknoten bezeichnet. Jedem Knoten der Voting-Struktur wird eine Stimmenanzahl und je eine Mindeststimmenanzahl für Lese- und Schreibquoren zugeordnet, die durch die Summe der Stimmen seiner Kindknoten erreicht werden muss. Wenn ein Knoten keine Kindknoten besitzt, dann sind die Mindeststimmenanzahlen null. Für die abgehenden Kanten eines Knotens kann die Reihenfolge des Zugriffs auf die Kindknoten durch die Angabe einer natürlichen Ganzzahl als Priorität festgelegt werden. Je kleiner die angegebene Ganzzahl, desto höher ist die Priorität der Kante. Ist für eine Kante keine Priorität definiert, dann wird die Priorität mit unendlich angenommen, was konzeptionell der niedrigsten Priorität entspricht. Für gleich priorisierte Kanten ist die Reihenfolge des Zugriffs nicht festgelegt und erfolgt zufällig. Die gleiche Kante kann für Lese- und Schreiboperationszugriffe verschieden priorisiert sein. In Abbildung 2.9 ist die Voting-Struktur gemäß dem ROWA-Verfahren mit fünf Knoten dargestellt. Physikalische Knoten sind durch Quadrate repräsentiert und virtuelle Knoten durch Quadrate mit abgerundeten Ecken. Die Knoten werden im Allgemeinen mit R für physikalische oder V für virtuelle Knoten und einer nachfolgenden Ganzzahl benannt, die der Unterscheidung der Knoten dient. Die physikalischen Knoten $R1$, $R2$, $R3$, $R4$ und

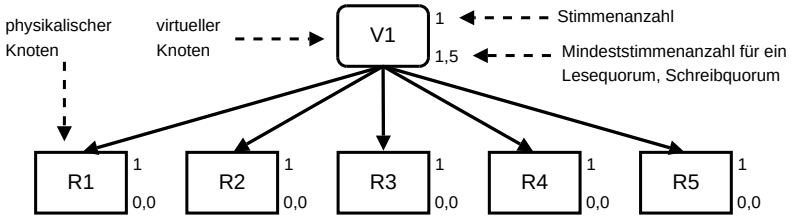
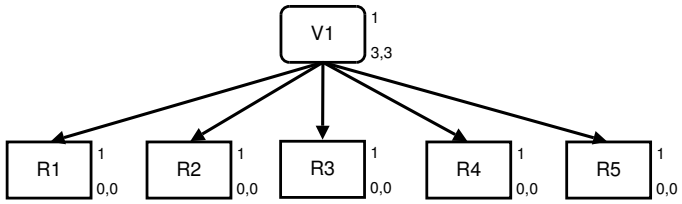
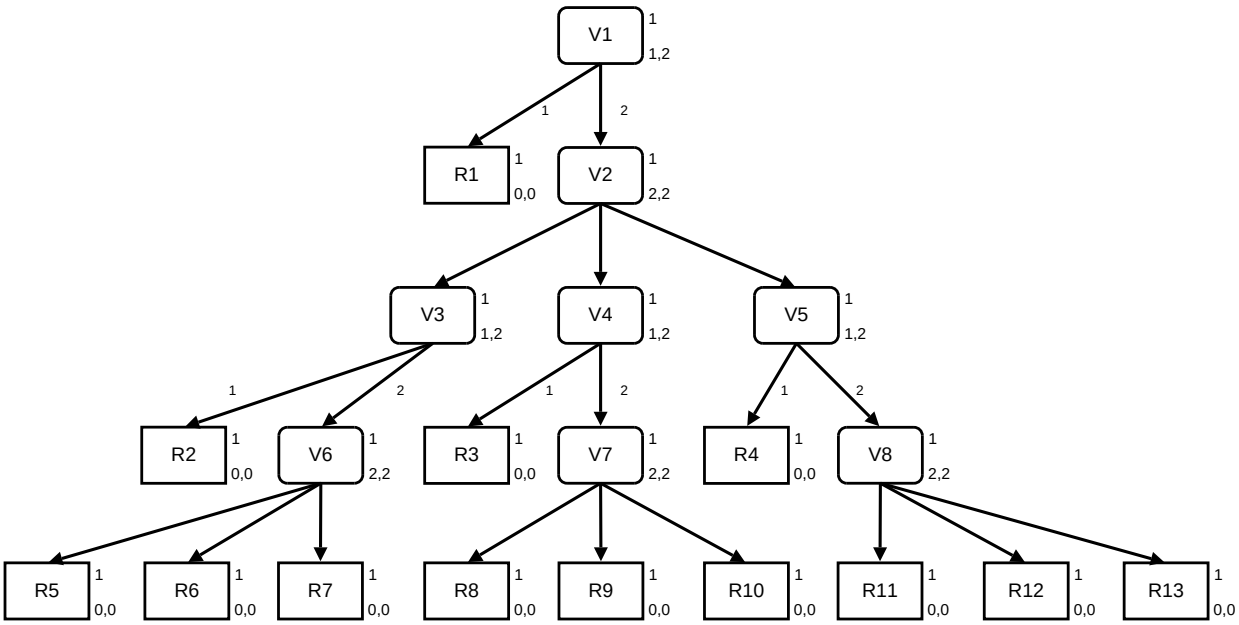


Abbildung 2.9: Voting-Struktur des ROWA-Verfahrens mit fünf Knoten

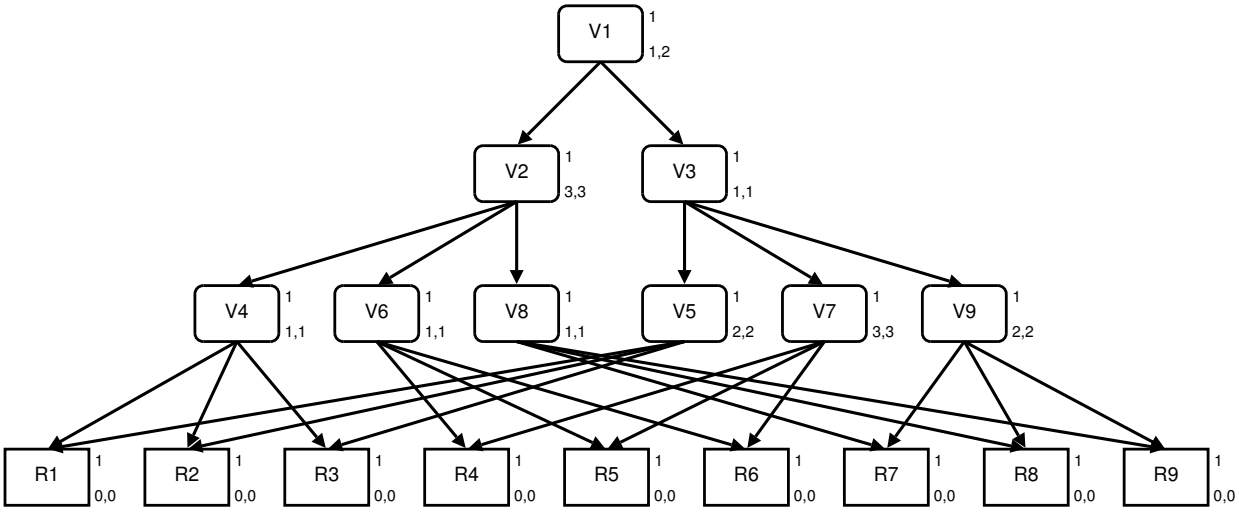
$R5$ besitzen alle die Stimmenanzahl eins und die Mindeststimmenanzahlen sind sowohl für Lese- als auch für Schreibquoren auf null gesetzt, da die Knoten keine Kindknoten besitzen. Der virtuelle Wurzelknoten $V1$ ist durch fünf ausgehende und gleich priorisierte Kanten mit allen fünf physikalischen Knoten verbunden. Gemäß dem ROWA-Verfahren, bei dem auf alle Knoten geschrieben wird, aber nur von einem Knoten gelesen werden muss, sind die Mindeststimmenanzahlen des Wurzelknotens $V1$ für Lesequoren eins und für Schreibquoren fünf.
Zur Konstruktion von Lese- oder Schreibquoren wird ausgehend vom Wurzelknoten der Struktur gefolgt. Der Wurzelknoten erfragt in der Reihenfolge, die durch die Prioritäten der abgehenden Kanten festgelegt ist, die Zustimmung seiner Kindknoten zur Durchführung der jeweiligen Operation. Falls die befragten Kindknoten selbst keine Kindknoten besitzen und die Stimmensumme der zustimmenden Kindknoten die jeweilige Mindeststimmenanzahl des Wurzelknotens für die Operation erreicht, dann gibt der Wurzelknoten seine Stimme für die Durchführung der Operation und die Operation kann mit allen zustimmenden Knoten als Quorum durchgeführt werden. Andernfalls kann die Operation nicht durchgeführt werden und der Wurzelknoten verweigert seine Stimme zur Durchführung der Operation.
Wenn die Kindknoten ihrerseits Kindknoten besitzen, dann werden die Kindknoten der Kindknoten, ebenfalls in der durch die Kantenprioritäten festgelegten Reihenfolge, um die Zustimmung befragt usw. bis eine Menge von Kindknoten befragt wird, die selbst keine Kindknoten besitzen. Wenn die Stimmensumme der zustimmenden Knoten aus dieser Menge die jeweilige Mindeststimmenanzahl des Vaterknotens für die Operation erreicht, dann gibt der Vaterknoten seine Stimme für die Durchführung der Operation seinem Vaterknoten, sonst wird die Abgabe der Stimme für die Durchführung der Operation verweigert. Wenn die Menge der zustimmenden Kindknoten dieses Vaterknotens die jeweilige Mindeststimmenanzahl für die Operation erreicht, dann gibt dieser Vaterknoten seine Stimme für die Durchführung der Operation seinem Vaterknoten usw. bis hin zum Wurzelknoten. Wenn die Summe der zustimmenden Kindknoten des Wurzelknotens die jeweilige Mindeststimmenanzahl für die Operation erreicht, dann gibt der Wurzelknoten seine Stimme für die Durchführung der Operation und die Operation kann durchgeführt werden. Andernfalls kann die Operation nicht durchgeführt werden und der Wurzelknoten verweigert seine Stimme zur Durchführung der Operation.
In Abbildung 2.9 ist für die Durchführung einer Schreiboperation Gemäß dem ROWA-Verfahren die Zustimmung des virtuellen Knotens $V1$ und aller Knoten aus der Menge $\{R1, R2, R3, R4, R5\}$ nötig. Eine Leseoperation benötigt dagegen nur die Zustimmung des virtuellen Knotens $V1$ und eines weiteren Knotens aus der Menge $\{R1, R2, R3, R4, R5\}$.
In Abbildung 2.10 sind die Voting-Strukturen für das MCV-Verfahren mit fünf Knoten, für das TQP-Verfahren mit dreizehn Knoten und für das 3×3 GP-Verfahren mit neun Knoten dargestellt.



(a) Voting-Struktur gemäß dem MCV-Verfahren mit fünf Knoten [TS99]



(b) Voting-Struktur gemäß dem TQP-Verfahren mit dreizehn Knoten [TS99]

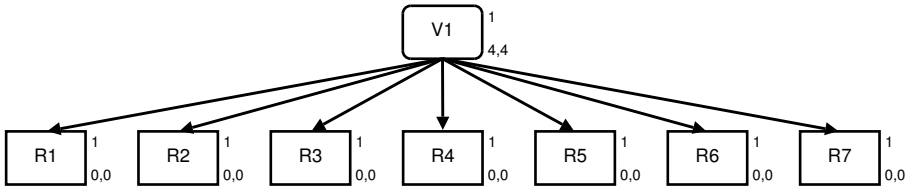


(c) Voting-Struktur gemäß dem 3×3 GP-Verfahren mit neun Knoten [The93a]

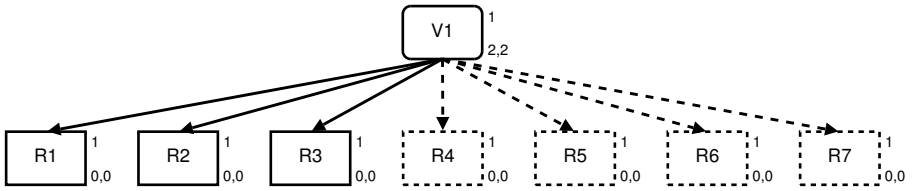
Abbildung 2.10: Voting-Strukturen vorgestellter Replikationsverfahren

2.3 Dynamische Verfahren

Statische Replikationsverfahren basieren auf einer fixen Anzahl Replikate, die zu Beginn des Verfahrens festgelegt wird. Wenn mehr Rechner ausgefallen sind als die vom Replikationsverfahren verwendete Strategie zur Durchführung von Lese- oder Schreiboperationen tolerieren kann, dann ist bei Rechnerausfällen der Zugriff auf die Daten nicht mehr möglich. Die Verfügbarkeit aller statischen Verfahren ist durch die eine symetrische Abhängigkeit des Verfügbarkeitsgrades von Lese- und Schreiboperationen beschränkt [TP98]. Um diese Beschränkung aufzuheben und die Verfügbarkeit zu erhöhen, müssen die Verfahren an geänderte Knotenanzahlen, die z.B. durch Knotenausfälle oder Wiederherstellung von ausgefallenen Knoten verursacht werden, anpassbar sein und die verwendete Strategie auf die neue Knotenanzahl adaptieren können.
Alle dynamischen Verfahren haben nach Ausfall, Anpassung an die neue Anzahl verfügbarer Knoten und anschließender Wiederherstellung ausgefallener Knoten das Problem die Konsistenz zu wahren, da die wiederhergestellten Knoten die angepasste Strategie nicht kennen und nach der nicht mehr gültigen alten Strategie verfahren. In Abbildung 2.11(a) ist die



(a) einfaches Mehrheitsverfahren mit sieben Knoten vor dem Ausfall und nach der Wiederherstellung der Knoten $R4$, $R5$, $R6$ und $R7$

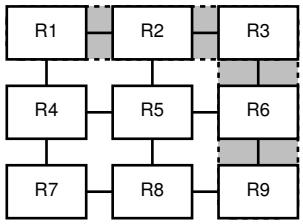


(b) einfaches Mehrheitsverfahren mit drei Knoten nach dem Ausfall der Knoten $R4$, $R5$, $R6$ und $R7$

Abbildung 2.11: Voting-Strukturen eines einfachen Mehrheitsverfahrens mit sieben und drei Knoten

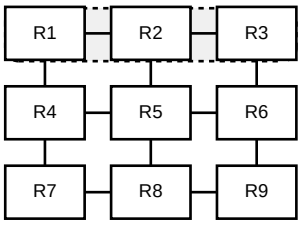
Voting-Struktur gemäß eines einfachen Mehrheitsverfahrens mit sieben Knoten dargestellt. Nach dem Ausfall der Knoten $R4$, $R5$, $R6$ und $R7$ darf nicht zu einer neuen Strategie mit den Knoten $R1$, $R2$ und $R3$ gewechselt werden, wie sie in Abbildung 2.11(b) dargestellt ist, da die Knoten $R4$, $R5$, $R6$ und $R7$ nach deren Wiederherstellung ein gültiges Lese- und sogar ein gültiges Schreibquorum der alten Strategie darstellen. Wenn einer dieser Knoten dann eine Leseoperation mit dem für diesen Knoten gültigen Lesequorum $\{R4, R5, R6, R7\}$ durchführen würde, dann liest der Knoten wahrscheinlich ein veraltetes Datum, da keine Überschneidung mit der Knotenmenge $\{R1, R2, R3\}$ vorhanden ist, obwohl die Knoten $R1$, $R2$ und $R3$ Schreiboperationen durchgeführt haben könnten und somit eine neuere Version des Replikats besitzen könnten.
Zur Vermeidung dieses Problems werden *Epochen* [LR93] benutzt. Eine Epoche ist eine Submenge der Menge aller am Replikationsverfahren teilnehmenden Knoten, die zu einer

bestimmten Zeit verfügbar und verbunden sind. Initial sind alle Knoten in der *aktuellen Epoche* mit der *Epochennummer* null. Die Epochennummer ist eine positive Ganzzahl und dient zur Unterscheidung von Epochen. Jede dynamische Anpassung des Verfahrens führt zu einem *Epochenwechsel*, der zu einer neuen aktuellen Epoche mit einer um eins erhöhten Epochennummer führt. Der Epochenwechsel kann nur mit der Zustimmung einer Menge von Knoten durchgeführt werden, die einem Schreibquorum der bisherigen und der neuen Epoche entspricht. Während des Epochenwechsels wird die neue Strategie und die neue Epochennummer den Knoten der Vereinigungsmenge der beiden Schreibquoren bekannt gemacht. Operationen dürfen nur von Knoten ausgeführt werden, die Teil der aktuellen Epoche sind und somit die aktuelle Strategie kennen. Dazu wird die Epochennummer aller an einem Quorum beteiligten Knoten erfragt und die maximale Epochennummer identifiziert. Wenn der die Operation durchführende Knoten eine kleinere als die maximale Epochennummer besitzt, dann muss die Durchführung der Operation abgebrochen werden und der Knoten aktualisiert die Epochennummer und die Strategie anhand eines Knotens mit der maximalen Epochennummer. Anschließend kann ein erneuter Versuch zur Durchführung der Operation gestartet werden.
In Abbildung 2.12 ist die Notwendigkeit dargestellt, einen Epochenwechsel mit einem Schreibquorum der alten und der neuen Epoche durchzuführen. Zu Anfang wird das in Abbildung 2.12(a) dargestellte 3×3 GP-Verfahren mit neun Knoten benutzt. Alle Knoten sind in der Epoche mit der Epochennummer null. In dieser Epoche finden einige Schreiboperationen statt. Dann fallen die Knoten $R4$, $R5$, $R7$ und $R8$ aus. Die verbleibenden Knoten $R1$, $R2$, $R3$, $R6$ und $R9$ stellen ein gültiges Schreibquorum dar, sodass ein Epochen- und Strategiewechsel zum MCV-Verfahren mit fünf Knoten möglich ist, da die Menge $\{R1, R2, R3\}$ auch ein Schreibquorum der neuen Strategie ist. Die Knoten $R1$, $R2$, $R3$, $R6$ und $R9$ erhöhen ihre Epochennummer auf eins und werden im Zuge des Epochenwechsels gegebenenfalls aktualisiert, sodass alle ein aktuelles Replikat besitzen und die neue Strategie kennen. Die neue Epoche eins mit fünf Knoten ist in Abbildung 2.12(b) dargestellt. Anschließend werden einige Schreiboperationen mit dem Schreibquorum $\{R1, R2, R3\}$ durchgeführt, sodass die Knoten $R6$ und $R9$ eine ältere Version des Replikats besitzen als die Knoten $R1$, $R2$ und $R3$. Die ausgefallen Knoten $R4$, $R5$, $R7$ und $R8$, die die Epochennummer null haben, sind wieder repariert und sollen in das Verfahren aufgenommen werden, das wieder das 3×3 GP-Verfahren mit neun Knoten sein soll. Mit dem Schreibquorum $\{R1, R2, R3\}$ aus Abbildung 2.12(b), das gleichzeitig ein Lesequorum in Abbildung 2.12(c) darstellt, wird der Epochenwechsel durchgeführt. Die Knoten $R1$, $R2$ und $R3$ erhöhen ihre Epochennummer auf zwei. Keiner der Knoten muss während des Epochenwechsels aktualisiert werden, da die letzten Schreiboperationen in der vorherigen Epoche nur mit diesem Schreibquorum stattfanden. Mit einem beliebigen für eine Leseoperation benötigtem C-Cover aus der Menge $\{R4, R5, R6, R7, R8, R9\}$ kann in Abbildung 2.12(c) zum einen eine Operation in einer alten Epoche ausgeführt werden und zum anderen ein veraltetes Datum gelesen werden, selbst wenn sich die Knoten $R4$, $R5$, $R7$ und $R8$ mit der Epochennummer null anhand der Knoten $R6$ oder $R9$ auf die Epochennummer eins und dem neueren Replikat dieser Knoten aktualisieren würden. Wird die Identität des Schreibquorums der alten und des Lesequorums der neuen Epoche verboten, wie in Abbildung 2.12(d) zu sehen ist, dann besteht das Problem mit dem gültigen Lesequorum $\{R7, R8, R9\}$ der neuen Epoche trotzdem noch. Ein Schreibquorum der alten und ein Lesequorum der neuen Epoche reicht nicht aus, um die Konsistenz zu wahren. Wäre der Epochenwechsel von Abbildung 2.12(b) zu Abbildung 2.12(c) bzw. Abbildung 2.12(d) mit einem Schreibquorum der alten und einem

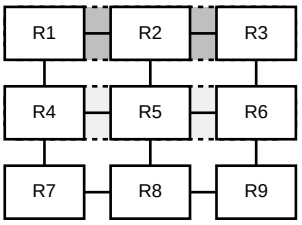


(a) Ausgangssituation: 3×3 GP-Verfahren mit neun Knoten, die alle in der Epoche null sind. Die Knoten $\{R1, R2, R3, R6, R9\}$ bilden ein Schreibquorum

(b) Epochen- und Strategiewechsel zum MCV-Verfahren mit fünf Knoten nach dem Ausfall der Knoten $R4$, $R5$, $R7$ und $R8$. Die Knoten $\{R1, R2, R3\}$ bilden ein Schreibquorum der neuen Epoche eins



(c) $\{R1, R2, R3\}$ ist ein Schreibqorum der alten Epoche eins und ein Lesequorum der neuen Epoche zwei



(d) $\{R1, R2, R3\}$ ist ein Schreibqorum der alten Epoche eins. $\{R4, R5, R6\}$ ist ein Lesequorum der neuen Epoche zwei

Abbildung 2.12: Notwendigkeit, einen Epochenwechsel mit einem Schreibquorum der alten und der neuen Epoche durchzuführen

Schreibquorum der neuen Epoche, nämlich z.B. mit $\{R1, R2, R3\}$ und $\{R1, R2, R3, R6, R9\}$ durchgeführt worden, bei dem diese Knoten eine aktuelle Version des Replikats und die Epochennummer zwei bekommen hätten, dann würde jedes gültige Lesequorum in Abbildung 2.12(c) bzw. Abbildung 2.12(d) mindestens einen Knoten der aktuellen Epoche mit einer aktuellen Version des Replikats enthalten. Es ist also ein Schreibquorum der alten und ein Schreibquorum der neuen Epoche nötig, um die Konsistenz zu erhalten.

In den folgenden beiden Abschnitten werden zwei dynamische Verfahren vorgestellt, die ihre statischen Pendants um die Dynamikeigenschaft erweitern. Abschließend wird ein generalisiertes dynamisches Verfahren vorgestellt, das alle vorgestellten Replikationsverfahren emulieren kann.

2.3.1 Unstrukturierte Verfahren

Ein Lese- oder Schreibquorum unstrukturierter Replikationsverfahren besteht aus einer Submenge der am Replikationsverfahren teilnehmenden Knoten, deren Stimmensumme die Mindeststimmenanzahl für die Durchführung der Lese- oder Schreiboperation erfüllt.

Dynamic Voting-Verfahren

Das *Dynamic Voting (DV)*-Verfahren [JM87a] erweitert das Majority Consensus Voting-Verfahren um die Dynamikeigenschaft. Ebenso wie beim MCV-Verfahren erhält jeder Knoten die Stimmenanzahl eins. Zusätzlich zur Versionsnummer verwaltet jeder Knoten die *update site cardinality (usc)* anhand der festgestellt werden kann wie viele Knoten an der letzten erfolgreichen Schreiboperation teilgenommen haben, die zur Erhöhung der Versionsnummer führte. Initial ist die usc jedes Knotens auf die Gesamtanzahl am Replikationsverfahren teilnehmender Knoten gesetzt. Hat z.B. der Knoten $R1$ die Versionsnummer 6 und seine usc ist 7, dann waren 7 Knoten an der Schreiboperation beteiligt, die zur Versionsnummer 6 führte.
Eine Partition wird *Majority Partition (MP)* genannt, wenn die Mehrheit der Knoten mit Replikaten der aktuellen Versionsnummer darin enthalten ist. Zu jedem Zeitpunkt existiert nur eine MP. Lese- und Schreiboperationen dürfen nur von Knoten in der MP durchgeführt werden. Operationen von Knoten, die sich nicht in der MP befinden, werden nicht erlaubt. Das Konzept der MP entspricht konzeptionell dem der Epochen. Um eine Operation ausführen zu dürfen, muss ein Knoten seine Mitgliedschaft in der MP erfolgreich prüfen. Dazu kontaktiert der Knoten alle Knoten mit denen Kommunikation möglich ist und ermittelt deren Versionsnummer und usc. Aus den Antworten wird die maximale Versionsnummer und die Menge der Knoten ermittelt, die an der letzten Schreiboperation teilgenommen haben und ein aktuelles Replikat mit der maximalen Versionsnummer besitzen. Aus dieser Menge wird anschließend die größte usc ermittelt. Falls die Anzahl der Knoten, die an der letzten Schreiboperation teilgenommen haben, größer als die Hälfte der größten usc ist, dann befindet sich der Knoten in der MP und darf die Operation durchführen. Wenn die Operation eine Schreiboperation ist, dann wird nach Durchführung der Schreiboperation auf allen teilnehmenden Knoten die Versionsnummer auf die um eins erhöhte maximale Versionsnummer gesetzt und die usc aller teilnehmenden Knoten wird auf die Anzahl der an der Schreiboperation beteiligten Knoten gesetzt. Leseoperationen dürfen auch ausgeführt werden wenn die Anzahl der Knoten, die an der letzten Schreiboperation teilgenommen haben, gleich der Hälfte der größten usc ist, da in keiner Partition aufgrund gleicher Anzahl aktueller Knoten eine Schreiboperation möglich ist.
Schreiboperationen dürfen nur auf aktuellen Replikaten durchgeführt werden. Knoten mit veraltetem Replikat müssen daher das Replikat aktualisieren um an zukünftigen Schreiboperationen teilzunehmen. Dazu ermittelt jeder Knoten in einem festgelegtem Intervall von jedem anderen Knoten dessen Versionsnummer und usc. Wenn die Versionsnummer eines der befragten Knoten größer als die eigene Versionsnummer ist, dann ist das Replikat veraltet. Befindet sich der Knoten in der MP, dann wird die Aktualisierung (*Catchup*) anhand eines Knotens mit höherer Versionsnummer durchgeführt. Sonst darf keine Aktualisierung stattfinden. Nach erfolgter Aktualisierung wird die Versionsnummer auf die aktuelle Versionsnummer gesetzt und auf allen Knoten in der MP wird die usc auf die um eins erhöhte größte usc gesetzt.
Das DV-Verfahren ermöglicht zur Laufzeit das Hinzufügen eines neuen Knotens in das Replikationsverfahren und das Entfernen eines vorhandenen Knotens daraus. Ein neue hinzugefügter Knoten setzt die Versionsnummer und die usc auf einen negativen Wert, da dieser nicht mit gültigen Werten verwechselt werden kann. Befindet sich der neue Knoten in der MP, dann kann eine Aktualisierung des Replikats durchgeführt werden und der neue Knoten wird Teil des Replikationsverfahrens. Sonst wird der neue Knoten bis zur Wieder-

vereinigung der Partition des neuen Knotens mit der MP nicht in das Replikationsverfahren aufgenommen. Ein Knoten mit veraltetem Replikat darf jederzeit aus dem Replikationsverfahren entfernt werden, indem das Replikat gelöscht wird und die anderen Knoten über die Entfernung des Knotens informiert werden. Wenn das Replikat des zu entfernenden Knotens aktuell ist, dann darf dieser Knoten nur entfernt werden wenn sich alle aktuellen Replikate oder eine Mehrheit der aktuellen Replikate in dessen Partition befinden. Dann erhöhen alle anderen Knoten in der Partition des zu entfernenden Knoten die Versionsnummer um eins und erniedrigen die usc um eins.
Das DV-Verfahren erlaubt die Durchführung von Schreiboperationen nur wenn die Anzahl der Knoten, die an der letzten Schreiboperation teilgenommen haben, größer als die Hälfte der größten usc ist. Das *Dynamic Linear Voting*-Verfahren [JM90] erlaubt auch bei Gleichheit der Anzahl mit der usc Schreiboperationen durchzuführen. Wenn eine gerade Anzahl Knoten an einer Schreiboperation teilnimmt, dann wird ein Knoten aus der Menge der daran teilnehmenden Knoten eindeutig bestimmt und allen teilnehmenden Knoten bekannt gemacht. Dieser Knoten wird *distinguished site* genannt. In der Partition des distinguished site-Knotens werden Schreiboperationen bei Gleichheit der usc mit der Anzahl Knoten, die an der letzten Schreiboperation teilgenommen haben, erlaubt, während in anderen Partitionen Schreiboperationen nicht erlaubt werden.
Das *Dynamic Weighted Voting*-Verfahren [Dav89] ordnet im Gegensatz zum DV-Verfahren jedem Knoten eine individuelle Stimme zu und ist das dynamische Gegenstück zum statischen Weighted Voting-Verfahren.
Die Kosten- und Verfügbarkeitsanalyse für das Dynamic Voting- und das Dynamic Weighted Voting-Verfahren ist in [JM87b] zu finden.

2.3.2 Strukturbasierte Verfahren

Strukturbasierte Replikationsverfahren ordnen die daran teilnehmenden Knoten in einer logischen Struktur an. Ein Lese- oder Schreibquorum besteht aus einer Submenge der am Replikationsverfahren teilnehmenden Knoten, die sich in bestimmten Positionen in der Struktur befinden müssen. Nur die Position eines Knotens in der Struktur und nicht dessen Stimmenanzahl wird für die Bildung der Lese- und Schreibquorenmenge berücksichtigt. Die Stimmenanzahl jedes Knotens wird entweder mit eins angenommen oder z.B. zur Positionierung der Knoten in der Struktur benutzt, falls den Knoten individuelle Stimmen zugeordnet wurden.

Dynamic Grid Protocol-Verfahren

Das *dynamic Grid Protocol (dGP)*-Verfahren [RL92] ordnet die am Replikationsverfahren teilnehmenden Knoten ebenso wie das statische GP-Verfahren logisch in einem $M \times N$ Gitter an, das aus M Spalten und N Zeilen besteht. Für die Durchführung einer Leseoperation ist die Zustimmung eines Knotens aus jeder Spalte (C-Cover) oder die Zustimmung aller Knoten einer Spalte des Gitters (CC-Cover) nötig. Die Durchführung einer Schreiboperation verlangt die Zustimmung eines Knotens aus jeder Spalte und die Zustimmung aller Knoten einer Spalte des Gitters. Das dGP-Verfahren erlaubt im Gegensatz zum GP-Verfahren unbesetzte Stellen im Gitter und ermöglicht die Anwendung des Verfahrens auch für Knotenanzahlen, die sich nicht durch $M * N$ darstellen lassen, indem die zum voll besetzten Gitter fehlenden Knoten einer Spalte als ausgefallene Knoten betrachtet werden. Es wird

ein $M \times N$ Gitter berechnet, das minimal mehr Stellen besitzt als es Knoten gibt. Für z.B. elf Knoten wird ein 4×3 Gitter berechnet, das voll besetzt zwölf Knoten aufnehmen kann und für elf Knoten entsprechend eine freie Stelle besitzt. Die Knoten werden von links nach rechts und von oben nach unten in das Gitter eingefügt, sodass sich nicht besetzte Stellen im Gitter rechtsbündig in der unteren Zeile befinden. Der Algorithmus zur Berechnung einer einem Lese- oder Schreibquorum entsprechenden Knotenmenge ändert sich gegenüber dem GP-Verfahren insofern, als dass keine unbesetzten Stellen in der Knotenmenge enthalten sein dürfen. Wenn in einer Spalte des Gitters Stellen unbesetzt sind, dann besteht das CC-Cover der Spalte aus allen mit Knoten besetzten Stellen. Ein C-Cover darf ebenfalls nur aus mit Knoten besetzten Stellen jeder Spalte gebildet werden. Durch die Berechnung eines $M \times N$ Gitters mit minimal mehr Stellen als Knoten passt sich das Verfahren nach Knotenausfällen oder der Wiederherstellung von Knoten der jeweils neuen Knotenanzahl an.
In [RL92] wird die Analyse der Kosten und Operationsverfügbarkeit des dGP-Verfahrens durchgeführt.

2.3.3 Generalisierung dynamischer Verfahren

Ebenso wie das statische GSV-Verfahren trennt das *dynamic General Structured Voting (dGSV)*-Verfahren [TS99, TS98, Boc00] die Strategie des Verfahrens von der Implementation und ermöglicht das Anpassen der Strategie ohne die Implementation verändern zu müssen. Das dGSV-Verfahren ist die Erweiterung des GSV-Verfahrens um die Dynamikeigenschaft. Bei einem Epochenwechsel wird aus der initial vom Administrator festgelegten Voting-Struktur, der *Master Voting-Struktur (MVS)*, die Voting-Struktur für die neue Epoche abgeleitet, die *Current Voting-Struktur (CVS)*. Die MVS fungiert als statische Schablone für die CVS. Die physikalischen Knoten in der MVS sind Platzhalter, denen im Verlauf der Konstruktion der CVS am Replikationsverfahren teilnehmende Knoten zugewiesen werden. Da die MVS statisch ist, kann jede CVS maximal so viele Knoten enthalten wie die MVS Platzhalter besitzt. Die maximale Anzahl wird bei der Erstellung der MVS festgelegt und ist zur Laufzeit nicht veränderbar. Die Vorgehensweise zur Konstruktion von Lese- oder Schreibquoren aus der CVS entspricht der des GSV-Verfahrens.
Für einige Replikationsverfahren sind bestimmte Knoten sehr wichtig. Für das TQP-Verfahren ist so ein Knoten z.B. der Wurzelknoten. Fällt einer dieser wichtigen Knoten aus, dann sollte dessen Position durch den zuverlässigsten unter den verbleibenden Knoten ersetzt werden. Das dGSV-Verfahren ermöglicht die Angabe einer Intaktwahrscheinlichkeit $\varphi \in (0, 1]$ für jeden Knoten, der die für die Zuordnung von Knoten zu Platzhalterknoten in der MVS benutzt wird. Je geringer φ eines Knoten ist, desto wahrscheinlicher ist ein Ausfall des Knotens. Umgekehrt ist die Wahrscheinlichkeit eines Ausfalls des Knotens umso geringer, je größer φ für diesen Knoten ist.
Die Ableitung der CVS aus der MVS während des Epochenwechsels geschieht in drei Schritten:

Schritt 1: Knotenzuweisung Im ersten Schritt werden die an der neuen Epochen teilnehmenden Knoten auf die Platzhalterknoten in der MVS verteilt. Zuerst wird für jeden physikalischen Platzhalterknoten und jeden virtuellen Knoten in der MVS ein Gewichtungswert $\tau \in \mathbb{N}$ bestimmt, der die Wichtigkeit des Knotens im Replikationsverfahren anzeigt.

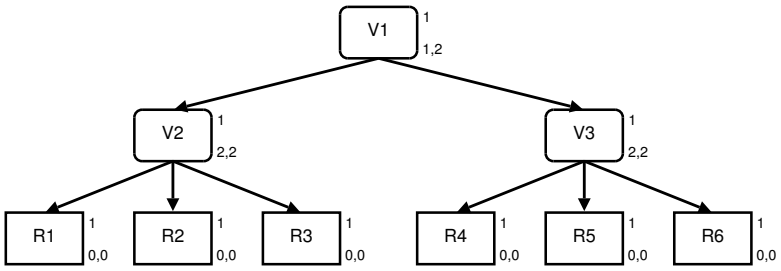
Je geringer τ, desto wichtiger ist der Knoten. Für die Bestimmung von τ gelten folgende Bedingungen:

1. Für jeden Knoten ist der Gewichtungswert τ höher als die jeweiligen Gewichtungswerte seiner Kindknoten.
2. Der Gewichtungswert eines Knotens ist höher (gleich) dem Gewichtungswert seines Bruderknotens, wenn die Kante mit der der Knoten mit seinem Vaterknoten verbunden ist eine höhere (gleiche) Priorität hat als die Kante seines Bruderknotens zum Vaterknoten. Sind Lese- und Schreibprioritäten angegeben, dann wird die Summe verglichen.

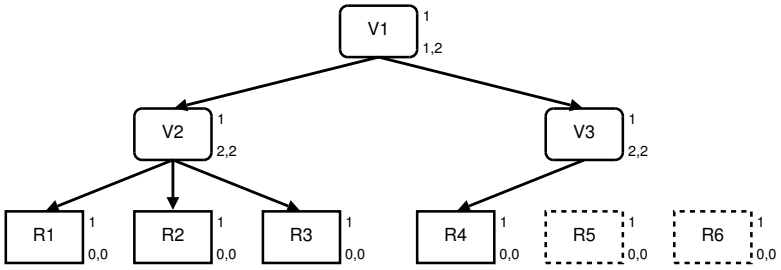
Alle am Replikationsverfahren der neuen Epoche beteiligten Knoten werden absteigend gemäß ihrer Intaktwahrscheinlichkeit φ sortiert. Knoten mit gleicher Intaktwahrscheinlichkeit werden gemäß eines eindeutigen Merkmals wie z.B. dem Namen absteigend sortiert, um eine eindeutige Reihenfolge zu erhalten. Alle physikalischen Platzhalterknoten der MVS werden absteigend gemäß ihres Gewichtungswertes τ sortiert. Die beiden entstandenen Listen werden zusammengeführt: jeweils der i-te Knoten der Liste mit den Knoten der neuen Epoche wird dem i-ten Knoten in der MVS Platzhalterknotenliste zugewiesen. So werden die für das Replikationsverfahren wichtigsten Knoten mit den zuverlässigsten am Replikationsverfahren teilnehmenden Knoten besetzt. Jeder Platzhalterknoten in der MVS ohne zugewiesenen Knoten hat dann nur Platzhalterknoten als Nachfolger, denen ebenfalls keine Knoten zugewiesen wurden.

Schritt 2: Umstrukturierung Wenn nicht allen Platzhalterknoten gleichen Gewichtungswertes Knoten zugeordnet werden können, dann kann es zu einer Verteilung der Knoten auf die Platzhalterknoten kommen, die bestimmte Knoten erheblich wichtiger werden lässt als es in der Strategie vorgesehen ist. Das ist z.B. dann der Fall, wenn ein virtueller Knoten nur einen Knoten als Sohnknoten zugewiesen bekommt, sein virtueller Bruderknoten aber deutlich mehr. Diese Situation ist in Abbildung 2.13(b) für die in Abbildung 2.13(a) dargestellte MVS, die das *Multi Level Voting*-Verfahren [FKT91] für sechs Knoten beschreibt, illustriert. Der virtuelle Knoten $V2$ hat drei Kindknoten, während sein Bruderknoten $V3$ nur den Kindknoten $R4$ besitzt. Für die Durchführung einer Schreiboperation muss der Kindknoten $R4$ immer verfügbar sein, während einer der drei Kindknoten von $V2$ für die Durchführung der Schreiboperation nicht benötigt wird.
Es wird ein *horizontaler Ausgleich* durchgeführt, bei dem die Knoten, die an der neuen Epoche teilnehmen, möglichst gleichmäßig auf die Platzhalterknoten gleichen Gewichtungswertes verteilt werden. Das ist zulässig, da alle Platzhalterknoten gleichen Gewichtungswertes auf einer horizontalen Ebene liegen. Die bisher berechnete CVS kann Platzhalterknoten enthalten, denen kein Knoten zugewiesen wurde. Diese Platzhalterknoten werden entfernt. Wenn virtuelle Knoten keinen oder nur noch einen Sohnknoten haben, dann werden diese virtuellen Knoten ebenfalls entfernt oder durch den einzigen Sohnknoten ersetzt.

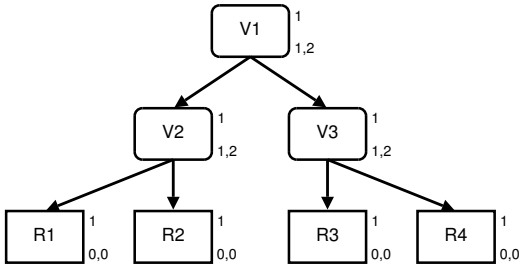
Schritt 3: Quorenneuberechnung Durch die Umstrukturierung kann es Knotenmengen geben, die die Mindeststimmenanzahlen für Lese- und Schreiboperationen des Vaterknotens nicht mehr erfüllen. Der Kindknoten $R4$ in Abbildung 2.13(b) erfüllt weder die Mindeststimmenanzahl für Schreiboperationen noch die Mindeststimmenanzahl für Leseoperationen seines Vaterknotens $V3$. Für Vaterknoten, deren Kindknoten die Mindeststimmenanzahlen



(a) Master Voting-Struktur des Multi Level Voting-Verfahrens mit sechs Knoten [Boc00]



(b) Current Voting-Struktur mit zwei unbesetzten Platzhalterknoten $R5$ und $R6$ für zwei ausgefallene Knoten vor dem horizontalen Ausgleich [Boc00]



(c) Current Voting-Struktur nach dem horizontalen Ausgleich [Boc00]

Abbildung 2.13: Ableitung der CVS mit vier Knoten aus der MVS für sechs Knoten wegen des Aufalls zweier Knoten

nicht erfüllen, werden neue Mindeststimmenanzahlen berechnet, die möglichst der Strategie entsprechen und durch die Kindknoten erfüllbar sein müssen. Wenn z.B. die Mindeststimmenanzahl für Leseoperationen eines Vaterknotens in der MVS einer einfachen Mehrheit der Stimmen seiner Kindknoten entspricht, dann entspricht die Mindeststimmenanzahl des Vaterknotens in der CVS nach der Quorenneuberechnung möglichst ebenfalls der einfachen Mehrheit der Stimmen seiner Kindknoten. In Abbildung 2.13(c) wären die Mindeststimmenanzahlen der beiden Vaterknoten $V2$ und $V3$ jeweils zwei, da die einfache Mehrheit von zwei Knoten zwei ist. Die Überschneidungseigenschaft ist jedoch auch mit den Mindeststimmenanzahlen eins für Leseoperationen und zwei für Schreiboperationen gesichert, sodass zugunsten der Leseoperation die Mindeststimmenanzahl für Leseoperationen der beiden Vaterknoten $V2$ und $V3$ auf eins gesetzt wird.
Die endgültige aus der MVS abgeleitete CVS ist in Abbildung 2.13(c) dargestellt. Die CVS wird durch den Epochenwechsel auf die Knoten der Vereinigungsmenge der beiden für den Epochenwechsel nötigen Schreibquoren geschrieben.

2.4 Zusammenfassung

Statische Replikationsverfahren basieren auf einer fixen Anzahl Replikate, die zu Beginn des Verfahrens festgelegt wird. Wenn mehr Rechner ausgefallen sind als die vom Replikationsverfahren verwendete Strategie zur Durchführung von Lese- oder Schreiboperationen tolerieren kann, dann ist bei Rechnerausfällen der Zugriff auf die Daten nicht mehr möglich. Die Verfügbarkeit aller statischen Verfahren ist durch eine symetrische Abhängigkeit des Verfügbarkeitsgrades von Lese- und Schreiboperationen beschränkt. Dynamische Replikationsverfahren überwinden diese Einschränkung durch Adaption der Strategie an geänderte Knotenanzahlen.
Nicht jedes Verfahren ist für alle Anwendungsszenarien gleichermaßen gut geeignet. Wenn sich die Anforderungen des Anwendungsszenarios an das Replikationsverfahren mit der Zeit ändern, dann muss das Verfahren manuell den neuen Anforderungen angepasst werden, was kostenintensiv, aufwendig und fehlerträchtig ist. Die Trennung der Strategie des Verfahrens von der Implementation ermöglicht das Austauschen der Strategie ohne die Implementation anpassen zu müssen. Das statische GSV-Verfahren und dessen dynamisches Pendant, das dGSV-Verfahren, ermöglichen durch die Beschreibung von Strategien in einem generischen und vereinheitlichtem Format, der Voting-Struktur, den Wechsel der Strategie nur durch den Austausch der Beschreibung zu erreichen.
Das dynamische dGSV-Verfahren hat jedoch den Nachteil zur Laufzeit auf eine vorher in der Master Voting-Struktur definierten maximalen Anzahl zu verwaltender Replikate bzw. Knoten beschränkt zu sein.
In folgenden Kapitel wird ein Verfahren vorgestellt, das auf dem dGSV-Verfahren aufbaut und es erweitert, um zur Laufzeit Knoten dem Replikationsverfahren hinzuzufügen oder Knoten daraus zu entfernen. Darüberhinaus wird die homogene Dynamik des dGSV-Verfahrens insofern erweitert, als dass für jede Knotenanzahl eine andere Strategie zum Einsatz kommen kann.

3 Ein adaptives, dynamisches und generalisiertes Voting-Verfahren

In diesem Kapitel wird das *adaptive dynamic General Structured Voting (adGSV)*-Verfahren vorgestellt. Das adGSV-Verfahren ist eine Erweiterung des dGSV-Verfahrens und basiert ebenso wie das dGSV-Verfahren auf den Konzepten von Voting-Strukturen und Epochen. Zur Laufzeit können neue Knoten in das Replikationsverfahren integriert und daran teilnehmende Knoten entfernt werden, wodurch im Gegensatz zum dGSV-Verfahren eine flexible Anzahl Replikate zur Laufzeit verwaltet werden kann. Die Dynamik des dGSV-Verfahrens ist homogen in dem Sinne, dass für alle Knotenanzahlen die gleiche durch die Master Voting-Struktur festgelegte Strategie benutzt wird. Die Master Voting-Struktur und somit die Strategie ist zur Laufzeit nicht veränderbar. Das adGSV-Verfahren ermöglicht zum einen den Strategiewechsel im homogenen Sinne und zum anderen die Benutzung einer inhomogenen Strategie, in der für jede Knotenanzahl eine andere Strategie benutzt werden kann.

Im folgenden Abschnitt wird die Generierung von Voting-Strukturen durch Strukturgeneratoren und der Mechanismus zur Auswahl eines Strukturgenerators beschrieben. Abschnitt 3.2 beschreibt die Funktionen zum Lesen und Schreiben eines Replikats, zur Aktualisierung der Verfahrensinformationen bzw. des Replikats eines Knotens und zur Durchführung eines Epochenwechsels. Daran anschließend wird im Abschnitt Abschnitt 3.3 das Verfahren zum Hinzufügen eines neuen Knotens in das Replikationsverfahren und das Entfernen eines Knoten daraus vorgestellt. Die Emulation des dGSV-Verfahrens durch das adGSV-Verfahren wird in Abschnitt 3.4 vorgestellt. Erweiterungen des Basiskonzeptes, die zum Teil in das Rahmenwerk implementiert wurden, das in Kapitel 4 vorgestellt wird, werden im Abschnitt 3.5 ausgeführt.

3.1 Generierung von Voting-Strukturen

Das adGSV-Verfahren stellt die Funktionen `create_replica()` und `delete_replica()`, die in Abschnitt 3.3 beschrieben werden, zum Hinzufügen neuer Knoten in das Replikationsverfahren und zur Entfernung bereits daran teilnehmender Knoten zur Laufzeit bereit. Für jede durch Hinzufügen und Entfernen von Knoten erreichbare Knotenanzahl muss dem Verfahren eine Voting-Struktur bekannt sein. Konzeptionell werden dazu Voting-Strukturen für jede mögliche Knotenanzahl in einem *Register* eingetragen. Das Register liefert bei Eingabe einer Knotenanzahl die entsprechende dort eingetragene Voting-Struktur. Um nicht manuell potenziell unendlich viele Voting-Strukturen erzeugen und in das Register eintragen zu müssen, wird ein *Strukturgenerator* benutzt, der als Eingabe eine Knotenanzahl erwartet und als Ausgabe eine Voting-Struktur für diese Knotenanzahl gemäß einer bestimmten Strategie liefert, die im Algorithmus des Strukturgenerators festgelegt ist. Für jede Strategie die verwendet werden soll ist ein Strukturgenerator zu implementieren, der Voting-Strukturen gemäß dieser Strategie erzeugt. Eine inhomogene Strategie wird erreicht, indem für verschiedene Knotenanzahlen verschiedene Strukturgeneratoren benutzt werden.

Um eine Entscheidung treffen zu können welcher Strukturgenerator für eine bestimmte Knotenanzahl benutzt werden soll, kann ein Strukturgenerator mit Knotenanzahlen attributiert werden für die dieser Strukturgenerator benutzt werden soll. Wenn zwei oder mehr Strukturgeneratoren mit der gleichen Knotenanzahl attributiert sind, dann wird ein Strukturgenerator aus dieser Menge per Zufall ausgewählt. Für die Knotenanzahlen für die kein Strukturgenerator attributiert ist wird der *Standardstrukturgenerator* benutzt, der vom Administrator aus der Menge der vorhandenen Strukturgeneratoren ausgewählt wird. Wenn für alle Knotenanzahlen der Standardstrukturgenerator zur Erzeugung von Voting-Strukturen benutzt wird, dann implementiert das Register eine homogene Strategie. Es muss mindestens ein Strukturgenerator zur Verfügung stehen, der dann automatisch als Standardstrukturgenerator benutzt wird. Das adGSV-Verfahren bietet Mechanismen zum Hinzufügen weiterer Strukturgeneratoren und zum Entfernen von Strukturgeneratoren zur Laufzeit an, die in Abschnitt 3.2.4 beschrieben werden. Das Entfernen eines Strukturgenerators ist nur dann möglich, wenn danach noch mindestens ein Strukturgenerator vorhanden ist, der dann automatisch als Standardstrukturgenerator ausgewählt wird.

Für die Benutzung von Voting-Strukturen, die nicht von einem vorhandenem Strukturgenerator erzeugt werden können, können Einträge im Register durch eine manuell erstellte Voting-Struktur belegt werden. Die manuell eingetragenen Voting-Strukturen haben Vorrang vor der Benutzung einer von einem Strukturgenerator erstellten Voting-Struktur, auch wenn Strukturgeneratoren für diese Knotenanzahl attributiert sind. Insbesondere zum schnellen Testen von Strategien ohne einen Strukturgenerator implementieren zu müssen bietet sich die manuelle Konstruktion und Eintragung von Voting-Strukturen in das Register an.

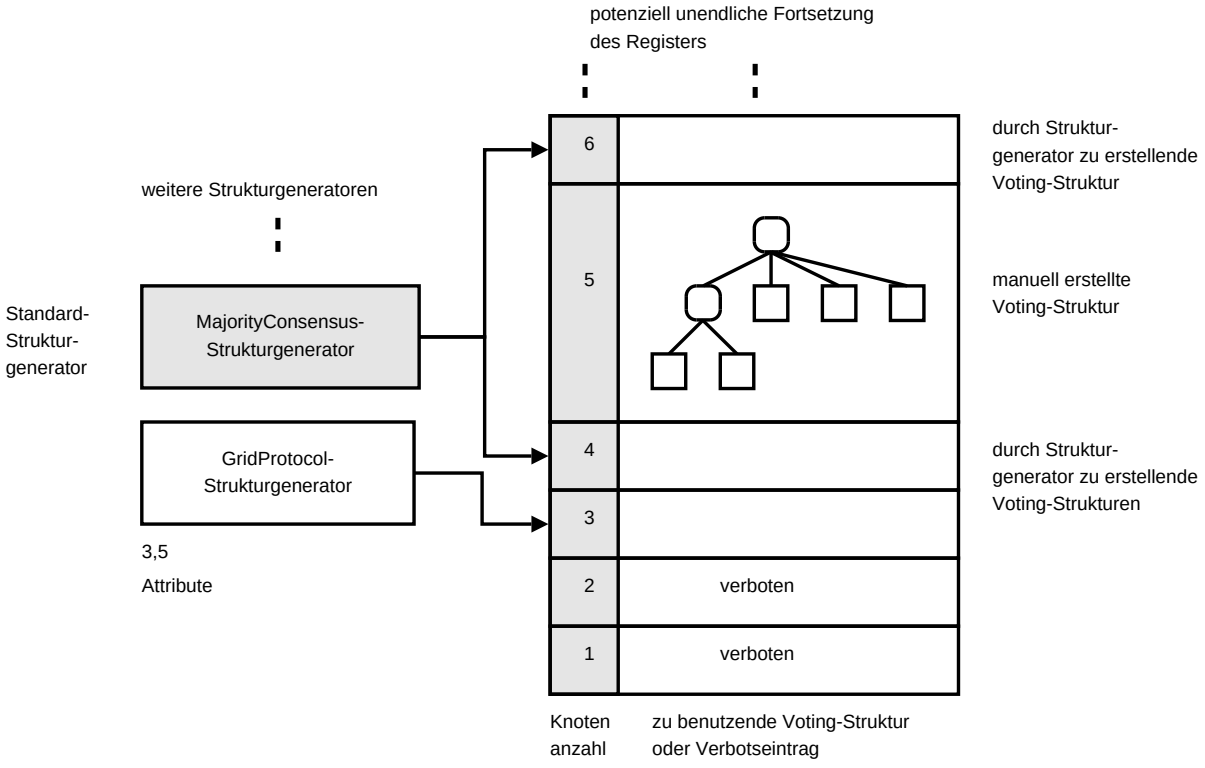


Abbildung 3.1: Register und Zuordnung von Strukturgeneratoren zu Registereinträgen

Für bestimmte Knotenanzahlen kann es sinnvoll sein eine andere als die eventuell im Register für diese Knotenanzahlen festgelegten Voting-Strukturen zu benutzen und die Erzeugung von Voting-Strukturen durch Strukturgeneratoren für diese Knotenanzahlen zu verbieten, auch wenn es Strukturgeneratoren gibt, die entsprechend attributiert sind. Für eine Knotenanzahl wird dazu der entsprechende Eintrag im Register als verboten markiert und die Voting-Struktur des nächst höheren nicht verbotenen Eintrags wird benutzt oder ein entsprechender Strukturgenerator wird zur Erzeugung der Voting-Struktur des nächst höheren nicht verbotenen Eintrags ausgewählt und benutzt. Die Voting-Struktur enthält dann einen Knoten mehr als am Replikationsverfahren teilnehmende Knoten vorhanden sind. Dieser Knoten wird als ausgefallen betrachtet. Wenn ein Bereich im Register als verboten markiert ist und die gegebenenfalls durch einen Strukturgenerator erzeugte Voting-Struktur des nächst höheren nicht verbotenen Eintrags benutzt wird, dann enthält die Voting-Struktur entsprechend der Länge des Bereichs mehr Knoten als am Replikationsverfahren teilnehmen. Diese Knoten werden ebenfalls als ausgefallen betrachtet. In einem als verboten markierten Bereich wird die Dynamik verhindert.
In Abbildung 3.1 sind zwei Strukturgeneratoren und das Register mit drei Einträgen dargestellt. Der MajorityConsensus-Strukturgenerator ist der Standardstrukturgenerator und nicht attributiert. Die manuell erstellte und in das Register in Zeile fünf eingetragene Voting-Struktur wird für die Knotenanzahl fünf benutzt obwohl der GridProtocol-Strukturgenerator mit fünf attributiert ist, da manuell in das Register eingetragene Voting-Strukturen Vorrang vor der Benutzung einer von einem Strukturgenerator erzeugten Voting-Struktur haben. Der GridProtocol-Strukturgenerator ist auch für die Knotenanzahl drei attributiert und wird für diese Knotenanzahl zur Erzeugung der Voting-Struktur benutzt, da kein weiterer Strukturgenerator mit dieser Knotenanzahl attributiert ist und der Standardstrukturgenerator nur für Knotenanzahlen benutzt wird für die kein anderer Strukturgenerator attributiert ist. Gäbe es einen weiteren Strukturgenerator, der auch mit drei attributiert ist, dann würde per Zufall entweder dieser oder der GridProtocol-Strukturgenerator zur Erzeugung der Voting-Struktur ausgewählt und benutzt werden. Die Einträge für einen und zwei Knoten sind als verboten gekennzeichnet. Für die Knotenanzahlen eins, zwei und drei wird die Voting-Struktur für die Knotenanzahl drei benutzt, zu deren Erzeugung der GridProtocol-Strukturgenerator aufgrund der Attributierung für die Knotenanzahl drei benutzt wird. Für Knotenanzahlen kleiner oder gleich drei Knoten wird so das statische GP-Verfahren für drei Knoten benutzt. Für alle anderen Knotenanzahlen wird der Standardstrukturgenerator zur Erzeugung von Voting-Strukturen benutzt, da keine Einträge im Register vorhanden sind und auch kein Strukturgenerator entsprechend attributiert ist. In Abbildung 3.1 wird der Standardstrukturgenerator nur zur Benutzung für die Knotenanzahlen vier und sechs dargestellt, er wird jedoch auch für alle Knotenanzahlen größer als sechs benutzt, die durch Hinzufügen neuer Knoten in das Replikationsverfahren erreicht werden können.

Im folgenden Beispiel wird ein Algorithmus für den MCV-Strukturgenerator und die manuelle Beschreibung einer Voting-Struktur gemäß dem MCV-Verfahren für fünf Knoten vorgestellt.

Beispiel eines Majority Consensus Voting-Strukturgenerators Eine Voting-Struktur gemäß dem Majority Consensus Voting-Verfahren enthält n physikalische Knoten und einen virtuellen Knoten, der über n Kanten, die alle die gleiche Lese- und Schreibpriorität besitzen, mit den physikalischen Knoten verbunden ist. Die Stimme aller Knoten ist eins. Die Mindeststimmenanzahl des virtuellen Knotens für ein Lese- und Schreibquorum ist $\lfloor \frac{n+1}{2} \rfloor$. Der Algorithmus des Majority Consensus Voting-Strukturgenerators ist im Pseudocode in Listing 3.1 zu sehen. Die Syntax ist an Python[1] angelehnt.

```
1  def CalcVotingStructure(nodes_phys=None, nodes_virt=None):
2
3     if |nodes_virt| > 0:
4        rootnode = nodes_virt[0]
5        del nodes_virt[1:]
6     else:
7        rootnode = NodeVirt()
8        nodes_virt += [rootnode]
9
10    rootnode.vote = 1
11    rootnode.edges = []
12    rootnode.quorum_read = rootnode.quorum_write = (|nodes_phys|+1) / 2
13
14    nodes_phys = sort_by_attr(nodes_phys, "reliability")
15
16    for physnode in nodes_phys:
17       physnode.vote = 1
18       physnode.quorum_read = physnode.quorum_write = 0
19       rootnode.AddEdge(target=physnode, prio_read=0, prio_write=0)
20
21    return nodes_phys, nodes_virt
```

Listing 3.1: Pseudocode für einen Majority Consensus Voting-Strukturgenerator

Die Funktion `CalcVotingStructure()` berechnet aus der Liste der übergebenen physikalischen Knoten `nodes_phys` und der Liste der übergebenen virtuellen Knoten `nodes_virt` die Voting-Struktur für die Anzahl an Knoten, die der Länge der `nodes_phys`-Liste entspricht. Falls die Liste der virtuellen Knoten mehr als ein Element enthält wird das erste Element der Liste als Wurzelknoten definiert und der Rest der Liste gelöscht (Zeilen 3 bis 5). Wenn die Liste leer ist, dann wird ein neuer virtueller Knoten erzeugt und als Wurzelknoten definiert (Zeilen 6 bis 8). In Zeile 10 wird die Stimme des Wurzelknotens auf eins gesetzt. In Zeile 11 wird die Liste der ausgehenden Kanten des Wurzelknotens geleert. In Zeile 12 wird die Mindeststimmenanzahl des Wurzelknotens für ein Lese- und Schreibquorum berechnet. In Zeile 14 werden die physikalischen Knoten gemäß ihrer Zuverlässigkeit sortiert. Die zuverlässigsten Knoten befinden sich danach am Anfang der Liste. Die Schleife in den Zeilen 16 bis 19 iteriert über die Liste der physikalischen Knoten, wobei die Stimme des Knotens auf eins und die Mindeststimmenanzahlen für Lese- und Schreibquoren auf null gesetzt werden (Zeile 17 und 18). Danach wird in Zeile 19 der Wurzelknoten mit dem physikalischen Knoten

[1] Python `http://www.python.org` [06.2005]

verbunden. In Zeile 21 werden die beiden Listen als Ergebnis zurückgegeben.

Das folgende Listing 3.2 zeigt die Beschreibung der in Abbildung 3.2 dargestellten Voting-Struktur. Die Syntax der Beschreibungssprache zur Definition von Voting-Strukturen entspricht der *DOT*-Beschreibungssprache von GraphViz[2].

```
1  digraph "MajorityConsensus" {
2
3     // Anzahl physikalischer Knoten für die
4     // diese Voting-Struktur benutzt werden soll.
5     numphysicalnodes=5;
6
7     // Definition der physikalischen Knoten
8     // (in Klammern Standardwerte bei Nichtangabe)
9     // Attribut 'type': immer "physical"
10    // Attribut 'vote': Stimmenanzahl (Standard: 1)
11
12    R1 [type="physical"];
13    R2 [type="physical"];
14    R3 [type="physical"];
15    R4 [type="physical"];
16    R5 [type="physical"];
17
18    // Definition der virteller Knoten:
19    // Attribut 'type': immer "virtual"
20    // Attribut 'vote': Stimmenanzahl (Standard: 1)
21    // Attribut 'quorum_read': Lesequorum (Standard: 0)
22    // Attribut 'quorum_write': Schreibquorum (Standard: 0)
23
24    V1 [vote=1,type="virtual",quorum_read=3,quorum_write=3];
25
26    // Definition der Kanten zwischen virtuellen und
27    // physikalischen Knoten:
28    // Attribut 'prio_read': Lesepriorität (Standard: 0)
29    // Attribut 'prio_write': Schreibpriorität (Standard: 0)
30    V1->R1 [prio_read=0, prio_write=0];
31    V1->R2 [prio_read=0, prio_write=0];
32    V1->R3 [prio_read=0, prio_write=0];
33    V1->R4 [prio_read=0, prio_write=0];
34    V1->R5 [prio_read=0, prio_write=0];
35 }
```

Listing 3.2: Definition einer Voting-Struktur gemäß des MCV-Verfahrens für fünf physikalische Knoten

Zeile 1 enthält das Schlüsselwort `digraph` und den Namen der Voting-Struktur. Die abschließende Klammer leitet den Definitionsteil ein. In Zeile 5 wird festgelegt für welche

[2]GraphViz `http://www.graphviz.org` [11.2005]

Knotenanzahl die Voting-Struktur in das Register eingetragen werden soll. In den Zeilen 12 bis 16 werden die fünf physikalischen Knoten $R1$, $R2$, $R3$, $R4$ und $R5$ mit der Standardstimme eins definiert. In Zeile 24 wird der virtuelle Wurzelknoten $V1$ mit der Stimme eins und der Mindeststimmenanzahl von drei Knoten für Lese- und Schreibquoren definiert.

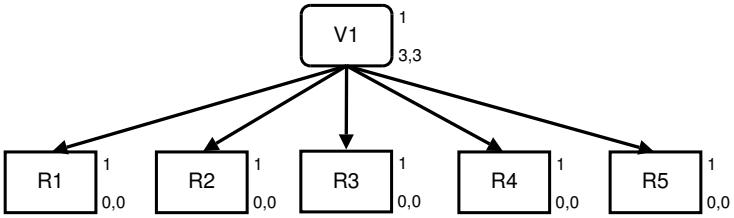


Abbildung 3.2: Voting-Struktur des Majority Consensus Voting-Verfahrens mit fünf physikalischen Knoten

Die Kanten von $V1$ zu den fünf Knoten werden in den Zeilen 30 bis 34 mit der Lese- und Schreibpriorität von jeweils null definiert. Die abschließende Klammer in Zeile 35 schließt den Definitionsteil ab.

Im folgenden Abschnitt wird die Durchführung der Lese-, Schreib- und Aktualisierungsoperation sowie die Durchführung des Epochenwechsels beschrieben.

3.2 Durchführung von Operationen

In diesem Abschnitt werden die Funktionen zum Lesen und Schreiben eines Replikats, zur Aktualisierung der Verfahrensinformationen bzw. des Replikats eines Knotens und zur Durchführung eines Epochenwechsels beschrieben.
Es wird angenommen, dass die Funktionen auf einem Knoten aufgerufen werden, der Teil des Replikationsverfahrens ist. Dieser Knoten wird Koordinator genannt. Indem ein am Replikationsverfahren teilnehmender Knoten als Koordinator und Vermittler fungiert, können Operationen auch von Knoten ausgeführt werden, die nicht Teil des Replikationsverfahrens sind. Dazu nimmt der Koordinator die Anfrage zur Durchführung einer Operation von einem nicht am Replikationsverfahren teilnehmenden Knoten an, führt die Operation aus und liefert das Ergebnis an den Knoten zurück.
Jeder am Replikationsverfahren teilnehmende Knoten besitzt Replikatinformationen, Verfahrensinformationen und Sperren:

Replikatinformationen Die Replikatinformationen bestehen aus dem Replikat und der Versionsnummer des Replikats.

Verfahrensinformationen Die Verfahrensinformationen bestehen aus der Epochennummer, der Voting-Struktur der aktuellen Epoche, dem Register, der Menge von Strukturgeneratoren und dem Standardstrukturgenerator.

Sperren Zur Sicherstellung des gegenseitigen Ausschlusses von Lese- und Schreiboperationen, Schreib- und Schreiboperationen und Lese- bzw. Schreib- und Epochenwechseloperation besitzt jeder Knoten eine Lesesperre, eine Schreibsperre und eine Epochensperre.

3.2.1 Durchführung der Leseoperation

Das Lesen des Datums eines Replikats mittels der `read()`-Funktion wird in fünf Schritten durchgeführt:

(Schritt 1) Der Koordinator ermittelt aus der Voting-Struktur eine Menge von Knoten, die einem Lesequorum entsprechen, und sendet eine Leseanfrage an diese Knoten. Wenn ein befragter Knoten der Durchführung der Leseoperation zustimmt, dann wird dessen Lesesperre gesetzt und der Knoten antwortet mit dem Tripel Versionsnummer, Epochennummer und gesetztem Zustimmungsmarker. Sonst antwortet der Knoten mit dem Tripel Versionsnummer, Epochennummer und nicht gesetztem Zustimmungsmarker. Ein befragter Knoten kann der Durchführung der Leseoperation nur zustimmen, wenn keine der Sperren gesetzt ist. Wenn nach einer definierten Zeitspanne keine Antwort eines Knotens erfolgt, dann wird die Ablehnung des Knotens zur Durchführung der Leseoperation angenommen und der Knoten wird als ausgefallen betrachtet.

(Schritt 2) Wenn nicht alle befragten Knoten zugestimmt haben, dann wird Schritt 1 wiederholt, wobei eine alternative Menge von Knoten gebildet wird, die einem Lesequorum entspricht und in der möglichst viele Knoten enthalten sind, die bereits zugestimmt haben. Dieser Prozess wird solange wiederholt bis entweder alle Knoten zustimmen oder die Leseoperation muss erfolglos abgebrochen werden. Der Koordinator sendet dann allen Knoten, die zugestimmt haben, die Nachricht die Lesesperre freizugeben.

(Schritt 3) Aus der Menge der zustimmenden Knoten wird die höchste Epochennummer identifiziert. Wenn der Koordinator eine kleinere Epochennummer besitzt, dann bricht der Koordinator die Leseoperation gemäß den Epochenbedingungen (siehe Abschnitt 2.3) ab, sendet allen zustimmenden Knoten die Nachricht die Lesesperre freizugeben und führt die Aktualisierung seiner Verfahrensinformationen anhand der Menge der Knoten mit der höchsten Epochennummer durch (siehe Abschnitt 3.2.3).

(Schritt 4) Aus der Menge der zustimmenden Knoten wird die Menge der Knoten mit der höchsten Versionsnummer identifiziert. Wenn der Koordinator Teil dieser Menge ist, dann wird eine lokale Leseoperation auf dem Replikat durchgeführt und anschließend die Lesesperre freigegeben. Sonst wird von einem Knoten aus der Menge der Wert des Replikats gelesen und dieser Knoten gibt anschließend die Lesesperre frei. Die Auswahl des Knotens von dem gelesen wird kann z.B. per Zufall oder anhand der Entfernung vom Koordinator getroffen werden. Der Koordinator sendet allen Knoten aus der Menge der zustimmenden Knoten, mit Ausnahme des Knotens von dem gelesen wurde, die Benachrichtigung über den erfolgreichen Abschluss der Leseoperation, woraufhin die Knoten die Lesesperre freigeben.

(Schritt 5) Wenn in Schritt 1 ausgefallene Knoten erkannt wurden, dann wird ein Epochenwechsel vom Koordinator initiiert (siehe Abschnitt 3.2.4). Sonst wird aus der Menge der in Schritt 1 kontaktierten Knoten die Menge der Knoten mit kleinerer Versions- bzw. Epochennummer als die in Schritt 3 und Schritt 4 ermittelten höchsten Epochen- und Versionsnummer identifiziert. Dabei werden

auch Knoten berücksichtigt, die der Durchführung der Leseoperation nicht zugestimmt haben. Die Knoten dieser Menge werden instruiert sich anhand der Knoten mit den höchsten Versions- und Epochennummern zu aktualisieren (siehe Abschnitt 3.2.3).

In Schritt 1 ist der Koordinator nach Möglichkeit in der gebildeten Knotenmenge enthalten, um die Kommunikationskosten durch die unnötige Kontaktierung eines anderen Knotens zu senken. Wenn zwei Knoten zur gleichen Zeit eine Leseoperation initiieren und keiner von beiden die Zustimmung einer einem Lesequorum entsprechenden Menge von Knoten bekommt, weil benötigte Knoten bereits der Durchführung der Leseoperationen des jeweils anderen Knotens zugestimmt und die Lesesperre gesetzt haben und auch keine alternative Menge von Knoten gebildet werden kann, dann befindet sich das Verfahren in einer *Livelock*-Situation [BHG87]. Die Leseoperation ist prinzipiell noch verfügbar, wird aber durch die gesetzten Lesesperren blockiert. Die Verhinderung einer Livelock-Situation muss durch das verwendete Kommunikationsprimitiv verhindert werden. Beides gilt ebenso für die im nächsten Abschnitt dargestellte Durchführung einer Schreiboperation als auch für die Durchführung eines Epochenwechsels.

3.2.2 Durchführung der Schreiboperation

Das Schreiben eines Datums auf ein Replikat, wobei das bisherige Datum ersetzt wird, geschieht mittels der `write()`-Funktion in fünf Schritten:

(Schritt 1) Der Koordinator ermittelt aus der Voting-Struktur eine Menge von Knoten, die einem Schreibquorum entsprechen, und sendet eine Schreibanfrage an diese Knoten. Wenn ein befragter Knoten der Durchführung der Schreiboperation zustimmt, dann wird dessen Schreibsperre gesetzt und der Knoten antwortet mit dem Tripel Versionsnummer, Epochennummer und gesetztem Zustimmungsmarker. Ein befragter Knoten kann der Durchführung der Schreiboperation nur zustimmen, wenn keine seiner Sperren gesetzt ist. Sonst antwortet der Knoten mit dem Tripel Versionsnummer, Epochennummer und nicht gesetztem Zustimmungsmarker. Wenn nach einer definierten Zeitspanne keine Antwort eines Knotens erfolgt, dann wird die Ablehnung des Knotens zur Durchführung der Schreiboperation angenommen und der Knoten wird als ausgefallen betrachtet.

(Schritt 2) Wenn nicht alle befragten Knoten zugestimmt haben, dann wird Schritt 1 wiederholt, wobei eine alternative Menge von Knoten gebildet wird, die einem Schreibquorum entspricht und in der möglichst viele Knoten enthalten sind, die bereits zugestimmt haben. Dieser Prozess wird solange wiederholt bis entweder alle Knoten zustimmen oder die Schreiboperation muss erfolglos abgebrochen werden. Der Koordinator sendet dann allen Knoten, die zugestimmt haben, die Nachricht die Schreibsperre freizugeben.

(Schritt 3) Aus der Menge der zustimmenden Knoten wird die höchste Epochennummer identifiziert. Wenn der Koordinator eine kleinere Epochennummer besitzt, dann bricht der Koordinator die Schreiboperation gemäß den Epochenbedingungen ab, sendet allen zustimmenden Knoten die Nachricht die Schreibsperre freizugeben und führt die Aktualisierung seiner Verfahrensinformationen anhand der Menge der Knoten mit der höchsten Epochennummer durch.

(Schritt 4) Aus der Menge der zustimmenden Knoten wird die höchste Versionsnummer identifiziert. Der neue Wert des Replikats wird unter Verwendung eines Commit-Protokolls auf die Replikate der Knoten des Schreibquorums geschrieben und die Versionsnummer der Replikate dieser Knoten wird auf die um eins erhöhte höchste Versionsnummer gesetzt. Anschließend besitzen alle Knoten des Schreibquorums ein aktuelles Replikat und geben die Schreibsperre frei.

(Schritt 5) Wenn in Schritt 1 ausgefallene Knoten erkannt wurden, dann wird ein Epochenwechsel vom Koordinator initiiert. Sonst wird aus der Menge der in Schritt 1 kontaktierten Knoten die Menge der Knoten mit kleinerer Versions- bzw. Epochennummer als die in Schritt 3 und Schritt 4 ermittelten höchsten Epochen- und Versionsnummer identifiziert. Dabei werden auch Knoten berücksichtigt, die der Durchführung der Schreiboperation nicht zugestimmt haben. Die Knoten dieser Menge werden instruiert sich anhand der Knoten mit den höchsten Versions- und Epochennummern zu aktualisieren.

3.2.3 Durchführung der Aktualisierungsoperation

Ein Knoten R_o mit veralteten Replikat- bzw. Verfahrensinformationen aktualisiert diese durch Ausführung der `catchup()`-Funktion. Entweder wird R_o von einem anderen Knoten instruiert die `catchup()`-Funktion auszuführen oder R_o nimmt selbstständig eine Aktualisierung vor. Der erste Fall tritt dann ein, wenn R_o Teil eines Lese- oder Schreibquorums ist und vom Koordinator der Operation als veraltet erkannt wurde, weil die Versions- bzw. Epochennummer kleiner als die höchste Versions- bzw. Epochennummer im Lese- oder Schreibquorum ist. Die Menge der Knoten, anhand derer sich R_o aktualisieren kann, wird R_o in der Nachricht des Koordinators zur Aktualisierung mit übermittelt. Der zweite Fall tritt zum einen dann ein, wenn R_o Koordinator einer Lese- oder Schreiboperation ist und während der Ermittlung der höchsten Epochennummer feststellt dass seine Epochennummer kleiner ist. Gemäß den Epochenbedingungen muss R_o die Operation abbrechen, die Knoten des Lese- oder Schreibquorums informieren ihre Sperre freizugeben, seine Verfahrensinformationen aktualisieren und kann anschließend einen erneuten Versuch unternehmen die Operation durchzuführen. Die Menge der Knoten, anhand derer sich R_o aktualisieren kann, ist durch die Ermittlung der höchsten Epochennummer bekannt. Zum anderen wird die `catchup()`-Funktion beim Hinzufügen eines neuen Knotens in das Replikationsverfahren von diesem neuen Knoten ausgeführt, um sich mit den aktuellen Replikat- und Verfahrensinformationen zu initialisieren. Ein Knoten, anhand dessen sich R_o aktualisieren kann, wird R_o durch den Namensdienst genannt (siehe Abschnitt 3.3.1).
Jeder Knoten des Replikationsverfahrens verwaltet einen *Aktualisierungszähler*, der initial null ist und anzeigt wie viele veraltete Knoten sich gerade anhand des Knotens aktualisieren. Um nicht einen Knoten mit zu vielen Aktualisierungsoperationen zu belasten, kontaktiert R_o alle bekannten aktuellen Knoten und wählt den Knoten R_x mit dem kleinsten Zählerwert aus. Falls alle Knoten identische Zählerwerte haben, wird ein Knoten R_x z.B. per Zufall oder anhand der Entfernung zu R_o ausgewählt. Wenn die Liste der aktuellen Knoten aus nur einem Knoten R_x besteht, dann wird dieser Knoten ausgewählt. Wenn die Schreib- oder Epochensperre des ausgewählten Knotens R_x nicht gesetzt ist, dann inkrementiert R_x den Aktualisierungszähler um eins und sendet die Replikat- und Verfahrensinformationen an R_o, woraufhin sich R_o mit den empfangenen Daten aktualisiert bzw. initialisiert. Anschließend dekrementiert R_x den Aktualisierungszähler um eins. Da Leseoperationen das Replikat nicht

modifizieren, darf R_x auch bei gesetzter Lesesperre andere Knoten aktualisieren. Wenn die Schreib- oder Epochensperre des ausgewählten Knotens R_x hingegen gesetzt ist, dann wird die Aktualisierung verweigert und der Knoten R_o muss sich anhand eines anderen Knotens aktualisieren oder, falls alle Knoten aus der Liste die Aktualisierung verweigern, nach einer Wartephase eine neue Aktualisierungsanfrage an R_x senden.

3.2.4 Durchführung der Epochenwechseloperation

Der Epochenwechsel dient im adGSV-Verfahren zusätzlich zur Bekanntmachung der zu verwendenden Strategie in der neuen Epoche der Verteilung der Konfiguration des Replikationsverfahrens. Dazu werden die Verfahrensinformationen während des Epochenwechsels auf alle Knoten übertragen, die Teil eines oder beider der für den Epochenwechsel nötigen Schreibquoren sind. Die Verfahrensinformationen eines Knotens, der nicht Teil eines der beiden Schreibquoren ist, werden entweder im Anschluss an die erfolgreiche Durchführung des Epochenwechsels aktualisiert, falls der Knoten während der Quorenbildung eines der beiden Schreibquoren befragt wurde und der Durchführung des Epochenwechsels nicht zugestimmt hat, oder die Verfahrensinformationen des Knotens werden durch die Schritte 3 oder 5 der Lese- bzw. Schreiboperation aktualisiert. Änderungen an den Verfahrensinformationen wie z.B. die Änderung des Standardstrukturgenerators oder das Einfügen eines neuen Eintrags in das Register werden durch die anschließende Ausführung eines Epochenwechsels allen am Replikationsverfahren teilnehmenden Knoten bekannt gemacht.
Ein Epochenwechsel wird mittels der `epoch_change()`-Funktion in acht Schritten wie folgt durchgeführt:

(Schritt 1) Der Koordinator ermittelt aus der Voting-Struktur eine Menge von Knoten, die einem Schreibquorum entsprechen, und sendet eine Epochenwechselanfrage an diese Knoten. Wenn ein befragter Knoten der Durchführung des Epochenwechsels zustimmt, dann wird dessen Epochensperre gesetzt und der Knoten antwortet mit dem Tripel Versionsnummer, Epochennummer und gesetztem Zustimmungsmarker. Ein befragter Knoten kann der Durchführung des Epochenwechsels nur zustimmen, wenn keine seiner Sperren gesetzt ist. Sonst antwortet der Knoten mit dem Tripel Versionsnummer, Epochennummer und nicht gesetztem Zustimmungsmarker. Wenn nach einer definierten Zeitspanne keine Antwort eines Knotens erfolgt, dann wird die Ablehnung des Knotens zur Durchführung des Epochenwechsels angenommen und der Knoten wird als ausgefallen betrachtet.

(Schritt 2) Wenn nicht alle befragten Knoten zugestimmt haben, dann wird Schritt 1 wiederholt, wobei eine alternative Menge von Knoten gebildet wird, die einem Schreibquorum entspricht und in der möglichst viele Knoten enthalten sind, die bereits zugestimmt haben. Dieser Prozess wird solange wiederholt bis entweder alle Knoten zustimmen oder der Epochenwechsel muss erfolglos abgebrochen werden. Der Koordinator sendet dann allen Knoten, die zugestimmt haben, die Nachricht die Epochensperre freizugeben.

(Schritt 3) Aus der Menge der zustimmenden Knoten wird die höchste Epochen- und Versionsnummer identifiziert. Wenn der Koordinator eine kleinere Epochennummer besitzt, dann bricht der Koordinator den Epochenwechsel gemäß den Epochenbedingungen ab, sendet allen zustimmenden Knoten die Nachricht die

Epochensperre freizugeben und führt die Aktualisierung seiner Verfahrensinformationen anhand der Menge der Knoten mit der höchsten Epochennummer durch.

(Schritt 4) Wenn sich die Knotenanzahlen der teilnehmenden Knoten in der bisherigen und der neuen Epoche unterscheiden, dann erzeugt der Koordinator die Voting-Struktur der neuen Epoche. Sonst wird die Voting-Struktur der bisherigen Epoche weiterhin verwendet.

(Schritt 5) Der Koordinator ermittelt aus der Voting-Struktur der neuen Epoche eine Menge von Knoten, die einem Schreibquorum der neuen Epoche entspricht, und sendet eine Epochenwechselanfrage an diese Knoten. In diesem Schreibquorum sind möglichst viele Knoten enthalten, die auch schon im Schreibquorum der alten Epoche enthalten sind. So wird zum einen die Anzahl der Knoten minimiert, die die Durchführung eines Epochenwechsels ablehnen können, weil diese Knoten bereits ihre Zustimmung in Schritt 1 gegeben haben, und zum anderen nehmen weniger Knoten am Commit-Protokoll in Schritt 7 teil. Wenn ein befragter Knoten der Durchführung des Epochenwechsels zustimmt, dann wird dessen Epochensperre gesetzt und der Knoten antwortet mit dem Tripel Versionsnummer, Epochennummer und gesetztem Zustimmungsmarker. Sonst antwortet der Knoten mit dem Tripel Versionsnummer, Epochennummer und nicht gesetztem Zustimmungsmarker. Ein befragter Knoten kann der Durchführung der Operation nur zustimmen, wenn entweder keine Sperre oder die Epochensperre gesetzt ist, weil er Teil des Schreibquorums aus Schritt 1 ist.

(Schritt 6) Wenn nicht alle befragten Knoten zustimmen, dann wird Schritt 6 wiederholt, wobei eine alternative Menge von Knoten gebildet wird, die einem Schreibquorum entspricht und in der möglichst viele Knoten enthalten sind, die bereits zugestimmt haben. Dieser Prozess wird solange wiederholt bis entweder alle Knoten zustimmen oder der Epochenwechsel muss erfolglos abgebrochen werden. Der Koordinator sendet dann allen Knoten, die dem Epochenwechsel zugestimmt haben, die Nachricht die Epochensperre freizugeben.

(Schritt 7) Die Epochennummer wird auf die um eins erhöhte höchste Epochennummer gesetzt und die neuen Verfahrensinformationen werden unter Verwendung eines Commit-Protokolls auf die Knoten der Vereinigung der Schreibquoren der alten und der neuen Epoche geschrieben.

(Schritt 8) Aus der Menge der in Schritt 1 und Schritt 5 kontaktierten Knoten wird die Menge der Knoten mit kleinerer Epochen- oder Versionsnummer als die in Schritt 3 ermittelten höchsten Epochen- bzw. Versionsnummer identifiziert. Dabei werden auch Knoten berücksichtigt, die der Durchführung des Epochenwechsels nicht zugestimmt haben. Die Knoten dieser Menge werden instruiert sich anhand der Knoten mit der höchsten Epochen- und Versionsnummer zu aktualisieren.

3.3 Hinzufügen und Entfernen von Knoten

In diesem Abschnitt werden die Funktionen zum Hinzufügen eines Knotens in das Replikationsverfahren und das Entfernen eines Knoten daraus vorgestellt.

3.3.1 Hinzufügen eines Knotens

Um einen neuen physikalischen Knoten R_a dem Replikationsverfahren mittels der `create_replica()`-Funktion hinzuzufügen, muss der neue Knoten R_a mit den aktuellen Replikat- und Verfahrensinformationen initialisiert werden und durch einen Epochenwechsel, bei dem R_a in die Voting-Struktur der neuen Epoche integriert wird, in das Replikationsverfahren aufgenommen werden.
Der Knoten R_a kontaktiert einen Namensdienst, der alle am Replikationsverfahren teilnehmenden Knoten kennt. Die Adresse des Namensdienstes ist dem Knoten z.B. durch die explizite Angabe seitens des Administrators bekannt. Der Namensdienst fügt R_a der Menge der am Replikationsverfahren teilnehmenden Knoten hinzu und liefert die Adresse eines bereits am Replikationsverfahren teilnehmenden Knotens R_x zurück. Der Auswahlmodus des Knotens R_x durch den Namensdienst ist für das Verfahren nicht relevant und kann z.B. anhand der Entfernung des Knotens R_a von R_x oder anhand der Knotenverfügbarkeit stattfinden. Wenn die Menge der am Replikationsverfahren teilnehmenden Knoten leer ist, dann besitzt R_a als initialer Knoten des Replikationsverfahrens bereits die aktuellen Replikat- und Verfahrensinformationen, der Namensdienst fügt R_a als initiales Element in die Menge der am Replikationsverfahren teilnehmenden Knoten ein und liefert einen Rückgabewert der anzeigt dass R_a der initiale Knoten ist. Sonst führt der Knoten R_a die Aktualisierungsoperation `catchup()` anhand des Knotens R_x durch. Anschließend initiiert R_a einen Epochenwechsel bei dem eine neue Voting-Struktur erzeugt wird in der R_a enthalten ist. Wenn kein Epochenwechsel durchgeführt werden kann, dann wird nach einer Wartephase so lange ein erneuter Versuch gestartet bis der Epochenwechsel erfolgreich durchgeführt werden kann. Falls die Replikat- oder Verfahrensinformationen von R_a dann veraltet sind oder R_a bereits mit veralteten Replikat- oder Verfahrensinformationen von R_x initialisiert wurde, dann aktualisiert sich R_a während des Epochenwechsels in Schritt 3.
Die komplementäre Operation zum Hinzufügen, das Entfernen eines Knotens aus dem Replikationsverfahren, wird im nächsten Abschnitt vorgestellt.

3.3.2 Entfernen eines Knotens

Die Entfernung eines physikalischen Knotens R_d aus dem Replikationsverfahren ist vergleichbar mit der Entdeckung des Ausfalls des Knotens R_d während der Quorenbildung zur Durchführung einer Operation. Nach der Durchführung der Operation führt der Koordinator der Operation einen Epochenwechsel durch bei dem eine neue Voting-Struktur ohne den ausgefallenen Knoten erzeugt wird. Beim Entfernen des Knotens ist der „Ausfall“ hingegen vorher bekannt und der Knoten R_d kann vorher gegebenenfalls andere Knoten aktualisieren, falls R_d der einzige Knoten mit aktuellen Replikat- bzw. Verfahrensinformationen ist oder es zu wenig andere aktuelle Knoten gibt. Dazu führt R_d die `delete_replica()`-Funktion aus, die drei Schritte beinhaltet:

(Schritt 1) Der Knoten R_d ermittelt aus seiner Voting-Struktur eine Menge von Knoten, die einem Schreibquorum entsprechen, und befragt diese Knoten nach deren

Epochen- und Versionsnummern. Dabei setzt keiner der befragten Knoten eine Sperre. Wenn nach einer definierten Zeitspanne keine Antwort eines der befragten Knotens erfolgt, dann wird solange ein alternatives Schreibquorum, in dem möglichst viele Knoten des ersten Schreibquorums enthalten sind, ermittelt und befragt bis eine Menge von Knoten, die einem Schreibquorum entspricht, erfolgreich befragt wurde. Sonst bricht R_d die Ausführung der `delete_replica()`-Funktion erfolglos ab und muss weiterhin am Replikationsverfahren teilnehmen.

(Schritt 2) Aus der Menge der zustimmenden Knoten wird die höchste Epochen- und Versionsnummer identifiziert. Wenn R_d eine kleinere Epochen- bzw. Versionsnummer hat, dann hat R_d veraltete Replikat- bzw. Verfahrensinformationen und kann sich aus dem Replikationsverfahren entfernen. Dazu initiiert R_d einen Epochenwechsel, bei dem R_d nicht in der Voting-Struktur der neuen Epoche vorhanden ist, löscht danach seine lokalen Daten bezüglich des Replikationsverfahrens und meldet dem Namensdienst den Eintrag für den Knoten R_d zu entfernen. Wenn kein Epochenwechsel durchgeführt werden kann, dann bricht R_d die Ausführung der `delete_replica()`-Funktion erfolglos ab, aktualisiert sich anhand der Menge der Knoten mit der höchsten Epochen- und Versionsnummer und muss weiterhin am Replikationsverfahren teilnehmen.

(Schritt 3) Aus der Menge der in Schritt 1 kontaktierten Knoten wird die Menge der Knoten mit kleinerer Epochen- oder Versionsnummer als die in Schritt 2 ermittelten höchsten Epochen- bzw. Versionsnummer identifiziert. Die Knoten dieser Menge werden instruiert sich anhand des Knoten R_d synchron aktualisieren. Nachdem alle Knoten aktualisiert wurden initiiert R_d einen Epochenwechsel, bei dem R_d nicht in der Voting-Struktur der neuen Epoche vorhanden ist, löscht danach seine lokalen Daten bezüglich des Replikationsverfahrens und meldet dem Namensdienst den Eintrag für den Knoten R_d zu entfernen. Wenn keine Epochenwechsel durchgeführt werden kann, dann bricht R_d die Ausführung der `delete_replica()`-Funktion erfolglos ab und muss weiterhin am Replikationsverfahren teilnehmen.

Wenn die `delete_replica()`-Funktion abgebrochen werden musste, dann wird nach einer definierten Zeitspanne solange ein erneuter Versuch gestartet bis die Operation erfolgreich durchgeführt werden kann.

3.4 Emulation des dynamic General Structured Voting-Verfahrens

Das adGSV-Verfahren emuliert das Verhalten des dGSV-Verfahrens, wenn ausschließlich ein Strukturgenerator zum Erzeugen von Voting-Strukturen benutzt wird, die Registereinträge nicht berücksichtigt werden und die maximale Anzahl der am Replikationsverfahren teilnehmenden Knoten begrenzt ist. Dazu wird ein *Knotenzähler* den Verfahrensinformationen hinzugefügt. Der dGSV-Emulationsmodus wird durch die Initialisierung des Knotenzählers mit einer negativen Ganzzahl aktiviert. Der Betrag des Knotenzählerwertes entspricht der maximalen Anzahl Knoten die dem Replikationsverfahren hinzugefügt werden dürfen. Bei jedem Hinzufügen eines neuen Knotens wird der Knotenzähler um eins erhöht und die Anzahl hinzufügbarer Knoten wird entsprechend um eins verringert. Wenn der Knotenzähler

den Wert null erreicht, dann wird das Hinzufügen weiterer Knoten abgelehnt. Standardmäßig ist der Knotenzähler mit eins initialisiert und der dGSV-Emulationsmodus ist nicht aktiviert. Im dGSV-Emulationsmodus ist das Entfernen eines Knotens nicht möglich, da dies im dGSV-Verfahren auch nicht möglich ist. Die `delete_replica()`-Funktion wird ohne Effekt abgebrochen. Die einzige Möglichkeit einen Knoten im dGSV-Emulationsmodus dauerhaft zu entfernen ist das manuelle Löschen der Daten bezüglich des Replikationsverfahrens. Dann wird der Knoten bei einer Quorenbildung zur Durchführung einer Operation als ausgefallen erkannt und durch den anschließenden Epochenwechsel aus dem Replikationsverfahren entfernt. Da der Knotenzähler nicht dekrementiert wird und somit kein neuer Knoten dem Verfahren hinzugefügt werden kann, verringert sich die maximale Anzahl am Replikationsverfahren teilnehmender Knoten um eins.

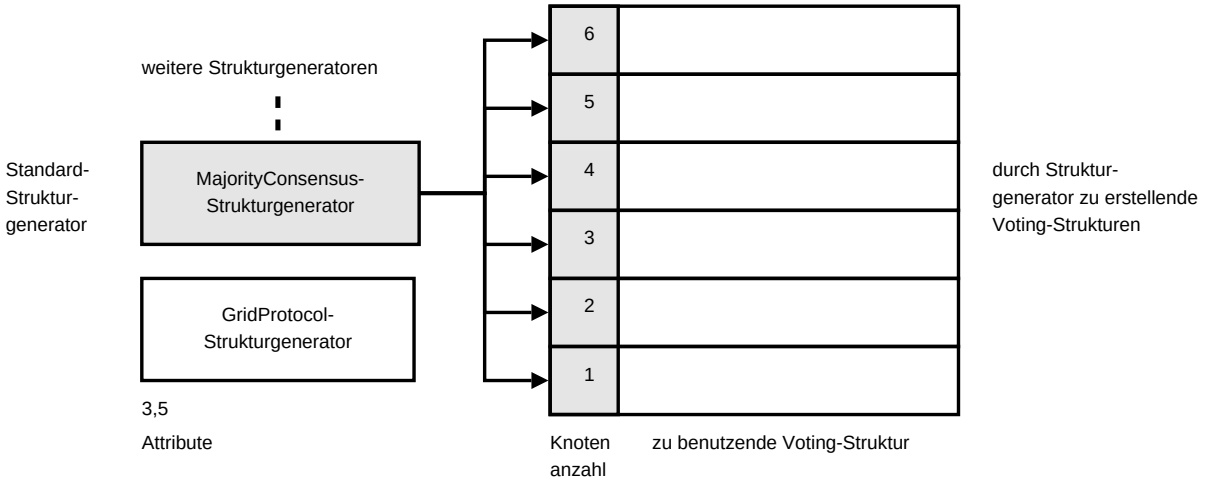


Abbildung 3.3: Emulation des dGSV-Verfahrens

Um die Verwendung einer homogen Strategie zu erreichen, muss für alle Knotenanzahlen der gleiche Strukturgenerator benutzt werden. Dazu wird der Auswahlprozess des Strukturgenerators zur Erzeugung einer Voting-Strukturen erweitert. Wenn der Knotenzähler einen Wert kleiner oder gleich null enthält, dann wird immer der Standardstrukturgenerator zur Erzeugung von Voting-Strukturen benutzt. Die Registereinträge und die Attribute der vorhanden Strukturgeneratoren werden nicht berücksichtigt. Konzeptionell entspricht das einer Konfiguration in der das das Register keine Einträge enthält und es nur einen Strukturgenerator gibt, der zugleich der Standardstrukturgenerator ist.
In Abbildung 3.3 ist eine Konfiguration des adGSV-Verfahrens mit dem MajorityConsensus-Strukturgenerator als Standardstrukturgenerator und dem initialen Knotenzählerwert -6 dargestellt. Aufgrund des negativen Wertes des Knotenzählers wird immer der Standardstrukturgenerator zur Erzeugung von Voting-Strukturen benutzt. Der für die Knotenanzahlen drei und fünf attributierte GridProtocol-Strukturgenerator wird ebenso wie eventuell weitere vorhandene Strukturgeneratoren und Registereinträge nicht benutzt.

3.5 Erweiterungen

In diesem Abschnitt werden funktionale und konzeptionelle Erweiterungen des bisher vorgestellten Basiskonzeptes beschrieben. Einige funktionale Erweiterungen sind bereits in das in das Rahmenwerk integriert, das im folgenden Kapitel vorgestellt wird.

3.5.1 Unterstützung für blockweisen Lese- und Schreibzugriff

Die meisten Dateisysteme erlauben wahlfreien blockweisen Lese- und Schreibzugriff auf Dateien. Eine Datei ist in Blöcke gleicher Größe eingeteilt, auf die unabhängig voneinander zugegriffen werden kann. Eine Leseoperation liefert nicht den kompletten Dateiinhalt, sondern den Inhalt eines Blocks der Datei zurück. Ebenso ersetzt eine Schreiboperation den Inhalt eines Blocks und nicht den gesamten Dateiinhalt. Im Kontext von Replikationsverfahren wird dieses Verhalten *partial writes* [RL92] genannt. Im Gegensatz dazu ersetzen Schreiboperationen mit *total writes*-Semantik den gesamten Dateiinhalt und Leseoperationen liefern entsprechend den gesamten Dateiinhalt zurück. Die total writes-Semantik entspricht der partial writes-Semantik, wenn die Blockgröße gleich der Dateigröße ist.
Blockweiser Zugriff ist für Operationen auf Replikaten mit großer Dateigröße sinnvoll, da nicht der gesamte Dateiinhalt sondern nur ein Block übertragen werden muss. Insbesondere für Schreiboperationen ist die Netzwerklast durch das Übertragen des neuen Datums auf alle an der Schreiboperation teilnehmenden Knoten durch die Übertragung eines im Vergleich zum gesamten Dateiinhalts kleinen Blocks wesentlich geringer. Für die Durchführung einer Schreiboperationen mit Unterstützung für blockweisen Schreibzugriff muss neben dem neuen Datum des Blocks noch die Blocknummer des zu schreibenden Blocks angegeben werden. Jeder Schreiboperation lässt sich so ein geschriebener Block zuordnen und umgekehrt lässt sich jedem Block eine Schreiboperation zuordnen, die den Block geschrieben hat. Durch die Erhöhung der Versionsnummer bei jeder Schreiboperation existiert so eine eindeutige Zuordnung von Versionsnummer, zugehöriger Schreiboperation und geschriebenem Block. Diese Zuordnung wird von jedem an der Schreiboperation teilnehmenden Knoten der knotenlokalen Historie der Schreiboperationen hinzugefügt. Für je zwei Knoten mit Replikaten gleicher Versionsnummer muss die Historie auf beiden Knoten identisch sein. Für je zwei Knoten mit Replikaten verschiedener Versionsnummer muss die Historie der Schreiboperationen auf beiden Knoten bis zur kleineren der beiden Versionsnummern identisch sein. Sonst sind potenziell verschiedene Daten gleichen Versionsnummern zugeordnet und die Datenkonsistenz ist nicht mehr gegeben. Die Durchführung einer Schreiboperation ist daher nur auf einer Menge von Knoten mit aktuellem Replikat möglich. Wenn ein Knoten ein aktuelles Replikat besitzt, dann entspricht die Versionsnummer der *realen Versionsnummer*. Sonst entspricht die Versionsnummer der *gewünschten Versionsnummer* und das Replikat ist als veraltet markiert. Den Replikatinformationen wird dazu ein *Veralterungsmarker* hinzugefügt. Bei gesetztem Veralterungsmarker entspricht die Versionsnummer der gewünschten Versionsnummer. Um eine Schreiboperation auf mehreren Blöcken durchzuführen, müssen der Schreiboperation entsprechend mehrere Blocknummern und Daten übergeben werden. Einer Versionsnummer ist dann einer Schreiboperation und mehreren Blöcken zugeordnet. Die in Abschnitt 3.2.2 beschriebene Vorgehensweise zur Durchführung einer Schreiboperation muss zur Unterstützung für blockweisen Schreibzugriff in den Schritten 4 und 5 angepasst werden. Die Schritte 1 bis 3 sind identisch mit der Durchführung einer Schreiboperation ohne Unterstützung für blockweisen Schreibzugriff.

(Schritt 4) Aus der Menge der zustimmenden Knoten wird die Menge der Knoten mit der höchsten realen Versionsnummer identifiziert. Auf allen anderen Knoten wird der Veralterungsmarker gesetzt die Versionsnummer auf die höchste reale Versionsnummer gesetzt. Der neue Wert des Blocks wird unter Verwendung eines Commit-Protokolls auf die Replikate der Knoten der Menge mit der höchsten realen Versionsnummer geschrieben, die Versionsnummer der Replikate dieser Knoten wird um eins erhöht und die Schreiboperation auf jedem Knoten der Historie hinzugefügt.

Die größte reale Versionsnummer muss gleich oder größer als jede gewünschte Versionsnummer eines Knotens aus der Menge der zustimmenden Knoten sein. Andernfalls gibt es außerhalb des Schreibquorums mindestens einen aktuelleren Knoten und die Schreiboperation muss abgebrochen werden. Der Koordinator sendet dann allen zustimmenden Knoten die Nachricht die Schreibsperre freizugeben und führt die Aktualisierung seiner Replikatinformationen anhand des aktuelleren Knotens durch.

(Schritt 5) Wenn in Schritt 1 ausgefallene Knoten erkannt wurden, dann wird ein Epochenwechsel vom Koordinator initiiert. Sonst wird aus der Menge der in Schritt 1 kontaktierten Knoten die Menge der Knoten mit kleinerer Versions- bzw. Epochennummer als die in Schritt 3 und Schritt 4 ermittelten höchsten Epochen- und Versionsnummer identifiziert. Die Knoten mit gesetztem Veralterungsmarker werden auch der Menge hinzugefügt. Dabei werden auch Knoten berücksichtigt, die der Durchführung der Schreiboperation nicht zugestimmt haben. Die Knoten dieser Menge werden instruiert sich anhand der Knoten mit den höchsten Versions- und Epochennummern zu aktualisieren.

Zur Unterstützung für blockweisen Lesezugriff wird in Schritt 4 der in Abschnitt 3.2.1 beschriebene Vorgehensweise zur Durchführung einer Leseoperation die Menge der Knoten mit der höchsten Versionsnummer anhand der größten realen Versionsnummer ausgewählt. Die größte reale Versionsnummer muss ebenfalls gleich oder größer als jede gewünschte Versionsnummer eines Knotens aus der Menge der zustimmenden Knoten sein. Andernfalls gibt es außerhalb des Lesequorums mindestens einen aktuelleren Knoten und die Leseoperation muss abgebrochen werden, der Koordinator muss die Replikatinformationen aktualisieren und kann dann einen erneuten Versuch zur Durchführung der Leseoperation vornehmen.
Die in Abschnitt 3.2.3 beschriebene Vorgehensweise zur Durchführung einer Aktualisierungsoperation muss erweitert werden, um den Veralterungsmarker nach der Aktualisierung anhand des Replikats eines Knotens, dessen reale Versionsnummer der gewünschten Versionsnummer des zu aktualisierenden Replikats entspricht, zurückzusetzen.
Der Mechanismus zur Unterstützung für blockweisen Lese- und Schreibzugriff ist in das Rahmenwerk integriert.

3.5.2 Unterstützung paralleler Leseoperationen

Im Gegensatz zur parallelen Durchführung zweier Schreiboperationen oder von Lese- und Schreiboperationen gefährdet die parallele Ausführung mehrerer Leseoperationen die Konsistenz des Replikats nicht. Das gilt auch für nicht parallele aber sich überlappende Operationen. Im Folgenden werden parallele und sich überlappende Operationen unter dem Begriff parallel zusammengefasst. Die Ausführung paralleler Leseoperationen steigert die

Performanz, allerdings zu Lasten von Schreiboperationen, die erst nach Beendigung aller Leseoperationen stattfinden dürfen. Insbesondere in Anwendungskontexten, in denen viele Leseoperationen und nur wenige Schreiboperationen durchgeführt werden, bietet es sich an paralleles Lesen zu erlauben. In Abbildung 3.4 wird z.B. paralleles Lesen für das MCV-

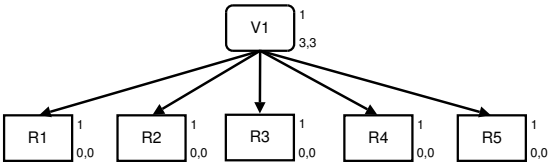


Abbildung 3.4: Voting-Struktur des Majority Consensus Voting-Verfahrens mit fünf physikalischen Knoten

Verfahren mit fünf physikalischen Knoten ermöglicht, wenn der virtuelle Knoten $V1$ statt der Mindeststimmenanzahlen $3, 3$ die Mindeststimmenanzahlen $2, 4$ besitzt. Durch zwei Lesequoren wie z.B. $\{R1, R2\}$ und $\{R4, R5\}$ ist die gleichzeitige Durchführung zweier Leseoperationen möglich. Um den gegenseitigen Ausschluss von Lese- und Schreiboperationen zu gewährleisten muss die Mindeststimmenanzahl für Schreiboperationen auf vier Knoten gesetzt werden.
Das Verfahren, durch Anpassung der Mindeststimmenanzahlen paralleles Lesen zu ermöglichen, ist zum einen strategiespezifisch und zum anderen ist die maximale Anzahl möglicher Quoren für parallele Leseoperationen inhärent durch die Voting-Struktur begrenzt. In Abbildung 3.4 ist die maximale Anzahl möglicher Quoren für parallele Leseoperationen auf fünf begrenzt, wenn der virtuelle Knoten $V1$ statt der Mindeststimmenanzahlen $3, 3$ die Mindeststimmenanzahlen $1, 5$ besitzt, was allerdings zur Degeneration des MCV-Verfahrens zum ROWA-Verfahren führt.
Das adGSV-Verfahren ermöglicht die Ausführung paralleler Leseoperationen ohne die strategiespezifische Anpassung der Voting-Struktur und unterliegt so gleichzeitig nicht der Beschränkung auf eine maximale Anzahl paralleler Leseoperationen. Jeder Knoten des Replikationsverfahrens besitzt einen *Leseoperationszähler*, der mit null initialisiert wird. Bereits an einer Leseoperation teilnehmende Knoten haben die Lesesperre gesetzt, stimmen einer weiteren Leseanfrage zu und erhöhen den Leseoperationszähler um eins. Wenn ein Knoten eine Leseanfrage erhält, der an keiner Operation teilnimmt und deshalb keine Sperre gesetzt hat, dann stimmt dieser Knoten der Leseanfrage zu, setzt die Lesesperre und erhöht den Leseoperationszähler auf eins. Sobald eine Leseoperation abgeschlossen ist erniedrigen die daran teilnehmenden Knoten den Leseoperationszähler um eins. Wenn der Leseoperationszähler wieder null erreicht, dann ist die letzte Leseoperation abgeschlossen und die Lesesperre wird freigegeben. Auf diese Weise sind beliebig viele parallele Leseoperationen durchführbar, allerdings können Schreiboperationen ausgehungert werden, weil zu viele Knoten die Lesesperre gesetzt haben und somit die Durchführung einer Schreiboperation ablehnen müssen. Deshalb stimmt ein Knoten parallelen Leseanfragen nur bis zu einem vom Administrator festgelegtem Maximum zu und lehnt danach alle Leseanfragen ab bis alle Leseoperationen beendet sind, der Leseoperationszähler den Wert null hat und die Lesesperre freigegeben wird. Dann haben sowohl Schreib- als auch erneute Leseanfragen die Möglichkeit die Stimme des Knotens zur Durchführung der Operation zu bekommen.
Durch Setzen des Maximums paralleler Leseoperationen auf eins wird paralleles Lesen ver-

hindert und die maximale Anzahl paralleler Leseoperationen wird durch die Voting-Struktur bestimmt.
Der Mechanismus zur Unterstützung paralleler Leseoperationen ist in das Rahmenwerk integriert.

3.5.3 Epochenverwaltung durch Epochenüberwachungsknoten

Bei den im Abschnitt 3.2 und 3.3 vorgestellten Operationen wird ein Epochenwechsel immer vom Koordinator der jeweiligen Operation durchgeführt. Wenn der zeitliche Abstand zwischen zwei Operationen sehr groß ist und zwischenzeitlich so viele Knoten ausgefallen sind dass kein Schreibquorum mehr gebildet werden kann, dann kann bis zur Wiederherstellung der fehlenden Knoten für ein Schreibquorum keine Schreiboperation und kein Epochenwechsel durchgeführt werden. Abhängig von der verwendeten Strategie und der Anzahl noch verfügbarer Knoten sind Leseoperationen gegebenenfalls noch möglich. In diesem Fall bietet sich die Benutzung eines Epochenüberwachungsknotens an, der in vom Administrator definierten Zeitabständen die Notwendigkeit für einen Epochenwechsel prüft, indem alle am Replikationsverfahren teilnehmenden Knoten kontaktiert werden. Wenn Knoten vom Epochenüberwachungsknoten als ausgefallen erkannt werden, dann führt der Epochenüberwachungsknoten einen Epochenwechsel als Koordinator durch. Während des Epochenwechsels wird ein neuer Epochenüberwachungsknoten aus Menge der am Schreibquorum der neuen Epoche teilnehmenden Knoten z.B. anhand der Knotenverfügbarkeit bestimmt und der bisherige Epochenüberwachungsknoten stellt den Epochenüberwachungsdienst ein. Der Ausfall des Epochenüberwachungsknotens wird von Knoten durch Ausbleiben der Prüfnachricht und anschließendem erfolglosen Kontaktversuch erkannt. In diesem Fall führt ein Knoten den Epochenwechsel als Koordinator durch, bei dem ein neuer Epochenüberwachungsknoten bestimmt wird. Auf diese Weise muss der Epochenüberwachungsknoten nicht selbst überwacht werden. Wenn der Koordinator einer Operation die Notwendigkeit für einen anschließenden Epochenwechsel erkennt, dann instruiert der Koordinator den Epochenüberwachungsknoten den Epochenwechsel durchzuführen statt selbst den Epochenwechsel durchzuführen.
Nach einer Partitionierung bei der in einer Partition, in der der Epochenüberwachungsknoten nicht enthalten ist, ein Epochenwechsel durchgeführt und ein Epochenüberwachungsknoten bestimmt wurde und anschließender Wiederverbindung des Netzwerks kann es zu einer Situation kommen in der zwei Epochenüberwachungsknoten aktiv sind. Die Wiederverbindung des Netzwerks wird von einem der beiden Epochenüberwachungsknoten erkannt und dieser führt einen Epochenwechsel durch bei dem ein neuer Epochenüberwachungsknoten bestimmt wird, woraufhin der Epochenüberwachungsdienst des den Epochenwechsel initiierenden Knotens eingestellt wird. Der verbleibende Epochenüberwachungsknoten ist entweder im Schreibquorum der alten oder neuen Epoche enthalten und erfährt so vom neuen Epochenüberwachungsknoten, woraufhin der Epochenüberwachungsdienst eingestellt wird, oder empfängt die Prüfnachricht des neuen Epochenüberwachungsknotens, woraufhin ein Epochenwechsel eingeleitet wird bei dem beide Epochenüberwachungsknoten im Schreibquorum der neuen Epoche enthalten sind. Während des Epochenwechsels wird ein neuer Epochenüberwachungsknoten bestimmt und die beiden bisherigen Epochenüberwachungsknoten stellen den Epochenüberwachungsdienst ein. Da ein Epochenwechsel nur exklusiv ausgeführt werden kann, ist das Vorhandensein mehrerer Epochenüberwachungskno-

ten nicht konsistenzgefährdend sondern erzeugt lediglich größere Netzwerklast durch das erhöhte Nachrichtenaufkommen.

3.5.4 Verzögerung des Epochenwechsels nach dem Hinzufügen oder Entfernen eines Knotens

Nach dem Hinzufügen eines Knotens in das Replikationsverfahren oder nach dem Entfernen eines Knotens daraus wird anschließend ein Epochenwechsel durchgeführt, der zu einer neuen aktuellen Epoche führt. Der entfernte Knoten nimmt nicht mehr am Replikationsverfahren teil und ist nicht Teil der neuen Epoche. Ein hinzugefügter Knoten wird dagegen in die neue Epoche integriert. Im Rahmen von nicht zeitkritischen administrativen Wartungsarbeiten ist die sofortige Integration oder Entfernung eines Knotens nicht immer notwendig. In solchen Fällen kann der nötige Epochenwechsel zum Hinzufügen oder Entfernen von Knoten verzögert werden. Ein neu hinzugefügter Knoten wird z.B. erst dann durch einen Epochenwechsel in das Replikationsverfahren integriert, wenn ein ausgefallener Knoten vom Koordinator einer Lese- oder Schreiboperation entdeckt wird und ein Epochenwechsel durchgeführt wird. Wenn mehrere Knoten hinzugefügt werden sollen, dann können alle neuen Knoten durch nur einen Epochenwechsel, der nach dem Hinzufügen aller Knoten initiiert wird, in das Replikationsverfahren integriert werden. Sonst müsste nach jedem Hinzufügen eines Knoten ein Epochenwechsel durchgeführt werden. Wenn ein Knoten entfernt wird und danach kein Epochenwechsel durchgeführt wird, dann wird der Knoten erst dann durch einen Epochenwechsel aus dem Replikationsverfahren entfernt, wenn der Koordinator einer Lese- oder Schreiboperation den Knoten als ausgefallen erkennt.
Das in Abschnitt 3.3.1 beschriebene Verfahren zum Hinzufügen eines Knotens und das in Abschnitt 3.2.4 beschriebene Verfahren zur Durchführung eines Epochenwechsels muss erweitert werden. Jeder Knoten verwaltet eine Liste hinzugefügter Knoten, die initial leer ist. Der Knoten R_x, anhand dessen sich ein neu hinzugefügter Knoten R_a initialisiert, fügt den Knoten R_a der Liste hinzugefügter Knoten hinzu. Wenn der Knoten R_a anschließend einen Epochenwechsel durchführt, dann wird der Knoten sofort in das Replikationsverfahren integriert. Sonst wird der Knoten R_a erst beim nächsten Epochenwechsel, bei dem der Knoten R_x Teil eines der beiden Schreibquoren ist, in das Replikationsverfahren integriert. Der Knoten R_x übermittelt dem Koordinator des Epochenwechsels zusätzlich zur Versionsnummer, der Epochennummer und dem gesetztem Zustimmungsmarker die Liste der hinzugefügten Knoten. Die Knoten, die in den übermittelten Listen aller am Epochenwechsel teilnehmenden Knoten enthalten sind, werden in die Liste der hinzugefügten Knoten des Koordinators eingefügt und in die Voting-Struktur der neuen Epoche integriert. Die neuen Replikatinformationen werden durch das Commit-Protokoll zusätzlich zu den Knoten der Vereinigung der beiden Schreibquoren auf die in der Liste der hinzugefügten Knoten des Koordinators enthaltenen Knoten geschrieben und die Liste der hinzugefügten Knoten wird auf allen Knoten geleert. Die neuen Knoten sind nach Abschluss des Epochenwechsels in das Replikationsverfahren integriert.
Um auch die neu hinzugefügten Knoten in das Replikationsverfahren zu integrieren, deren Initialisierungsknoten R_x nicht im Schreibquorum der alten Epoche enthalten ist, werden alle nicht im Schreibquorum der alten Epoche enthaltenen Knoten nach deren Liste der hinzugefügten Knoten befragt und in die Liste der hinzugefügten Knoten des Koordinators eingefügt.
Um trotz des Ausfalls des Knotens R_x in das Replikationsverfahren aufgenommen zu wer-

den, prüft ein neu hinzugefügter Knoten R_a periodisch den Status des Knotens R_x und initialisiert sich beim Ausfall des Knotens R_x anhand eines anderen Knotens, der den Knoten R_a daraufhin in die Liste der hinzugefügten Knoten aufnimmt.
Zur Verzögerung des Epochenwechsels nach dem Entfernen eines Knotens R_d aktualisiert der Knoten R_d gegebenenfalls andere Knoten, wie in in Abschnitt 3.3.2 beschrieben, führt aber abschließend keinen Epochenwechsel durch. Der Knoten R_d wird dann vom Koordinator einer Lese- oder Schreiboperation als ausgefallener Knoten erkannt und durch den anschließenden Epochenwechsel aus dem Replikationsverfahren entfernt.
Der Mechanismus zur Verzögerung des Epochenwechsels nach dem Hinzufügen oder Entfernen eines Knotens ist in das Rahmenwerk integriert.

3.5.5 Überlauf der Versions- bzw. Epochennummer

Die Versionsnummer des Replikats und die Epochennummer werden bei jeder Schreiboperation bzw. jedem Epochenwechsel um eins erhöht und steigen stetig. Durch die Beschränktheit von Datentypen kommt es bei Überschreitung des Wertebereichs des Datentyps zum Überlauf und die an der Schreiboperation oder dem Epochenwechsel teilnehmenden Knoten bekommen die Versions- bzw. Epochennummer null zugewiesen während alle anderen nicht daran teilnehmenden Knoten einen Wert kleiner dem Maximalwert des Datentyps besitzen. Diese Knoten werden aufgrund der höheren Versions- bzw. Epochennummern fälschlicherweise als aktueller angenommen, denn eine der Schreiboperation nachfolgende Leseoperation liest immer von einem Replikat mit der höchsten Versionsnummer und liefert somit nicht den zuletzt geschriebenen Wert des Replikats. Zur Ermittlung der aktuellen Strategie des Verfahrens muss ein Knoten mit der höchsten Epochennummer kontaktiert werden. Nach dem Überlauf der Epochennummer besitzen die Knoten der aktuellen Epoche jedoch eine Epochennummer, die kleiner als die der Knoten ist, die nicht am letzten Epochenwechsel teilgenommen haben. Die ermittelte Strategie entspricht also nicht der aktuell zu verwendenden Strategie.
Zur Vermeidung des Überlaufs der Versions- bzw. Epochennummer wird der Wertebereich des verwendeten Datentyps in zwei Phasen eingeteilt. In der ersten Phase wird die Versionsnummer bei Schreiboperationen bzw. die Epochennummer bei Epochenwechseln ungeachtet der Überlaufproblematik hochgezählt. In der zweiten Phase wird versucht über das für die Operation nötige Quorum hinaus alle anderen Knoten zu kontaktieren und deren Versionsnummer bzw. Epochennummer auf null zu setzen während die Knoten des Quorums der Schreiboperation oder des Epochenwechsels die Versions- bzw. Epochennummer eins bekommen. Die Länge der zweiten Phase muss entsprechend groß gewählt werden, um die Möglichkeit alle Knoten kontaktieren zu können zu gewährleisten.
Aufgrund des prototypischen Charakters des Rahmenwerkes ist dieser Mechanismus nicht implementiert.

3.5.6 Generalisierung von Strukturgeneratoren

Strukturgeneratoren erzeugen Voting-Strukturen für alle Knotenanzahlen gemäß der durch den jeweiligen Algorithmus implementierten Strategie. Für jede Strategie, die vom adGSV-Verfahren unterstützt werden soll, muss ein Strukturgenerator implementiert werden. Die Trennung von Strategie und Implementation der Strukturgeneratoren ermöglicht die Konfiguration von Strategien durch eine Strategiebeschreibung statt sie implementieren zu müs-

sen. Ein generischer Strukturgenerator interpretiert dazu die Strategiebeschreibung und generiert eine entsprechende Voting-Struktur. Um eine neue Strategie dem adGSV-Verfahren verfügbar zu machen genügt dann die Spezifikation der Strategie mittels der Beschreibungssprache.

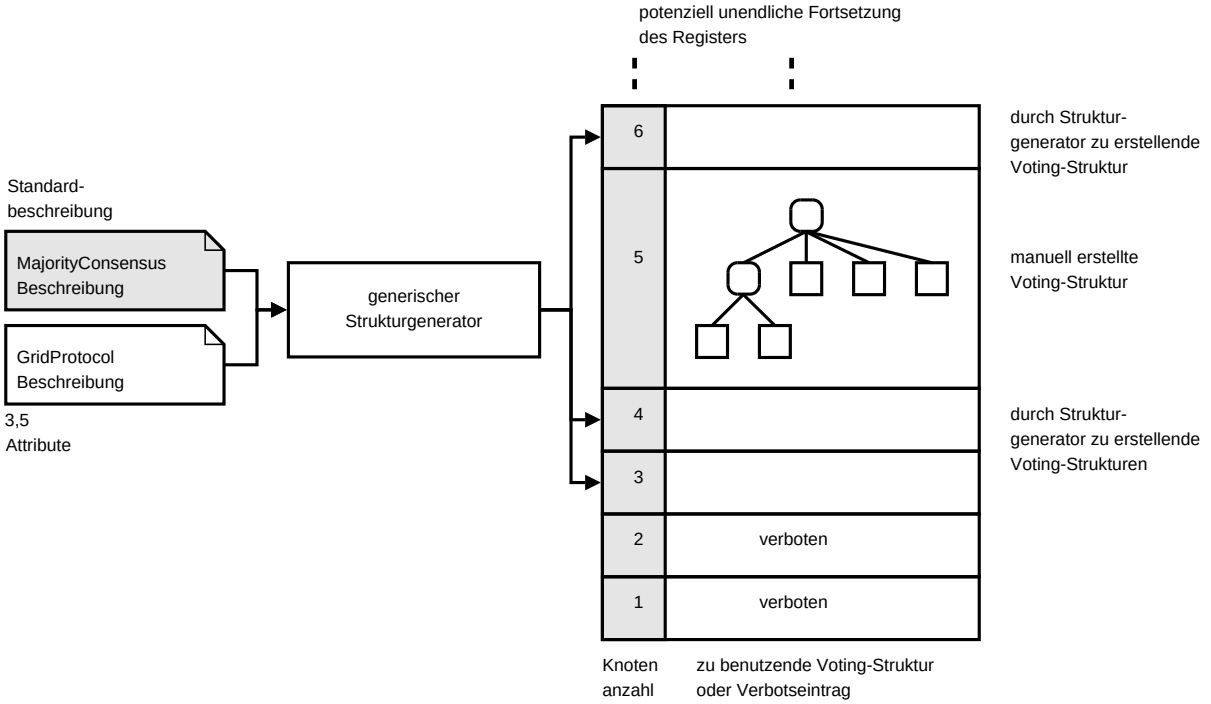


Abbildung 3.5: Integration des generischen Strukturgenerators in das adGSV-Verfahren

In Abbildung 3.5 ist die konzeptionelle Integration eines generischen Strukturgenerators in das adGSV-Verfahren dargestellt. Statt mehrerer Strukturgeneratoren gibt es nur noch einen generischen Strukturgenerator. Anstelle der Strukturgeneratoren treten die Strategiebeschreibungen, die genau wie Strukturgeneratoren attributiert werden können. Ebenso wie es mindestens einen Strukturgenerator geben muss, der dann gleichzeitig Standardstrukturgenerator ist, muss es auch mindestens eine Strategiebeschreibung geben, die dann auch die Standardstrategiebeschreibung ist. In Abbildung 3.5 ist die GridProtocol-Beschreibung für die Knotenanzahlen drei und fünf attributiert. Die Standardstrategiebeschreibung ist die MajorityConsensus-Beschreibung, die nicht attributiert ist. Bis auf den Unterschied dass anstelle von Strukturgeneratoren Strategiebeschreibungen und der generische Strukturgenerator verwendet werden sind keine Änderungen an den vorgestellten Mechanismen des adGSV-Verfahrens nötig.
Im Rahmen dieser Arbeit war die Entwicklung einer Strategiebeschreibungssprache nicht möglich, sodass das funktional äquivalente Konzept der Strukturgeneratoren eingesetzt wurde.

3.6 Zusammenfassung

Das in diesem Kapitel vorgestellte adaptive dynamic General Structured Voting-Verfahren erlaubt die Verwaltung einer zur Laufzeit flexiblen Anzahl Replikate. Dazu wurden zwei Funktionen zum Hinzufügen von Knoten in das Replikationsverfahren und zum Entfernen von Knoten daraus vorgestellt. Durch das Konzept der Strukturgeneratoren und den in Abschnitt 3.1 vorgestellten Mechanismen können inhomogene dynamische Strategien benutzt werden.
Im folgenden Kapitel wird das Rahmenwerk vorgestellt, das die Konzepte des adGSV-Verfahrens implementiert.

4 Architektur und Realisierung des Rahmenwerkes

Für das im vorherigen Kapitel vorgestellte adGSV-Verfahren wurde ein Rahmenwerk implementiert, um die praktische Durchführbarkeit des Konzeptes zu belegen und die Eigenschaften des adGSV-Verfahrens testen zu können.

Im folgenden Abschnitt wird die Entscheidung zugunsten der verwendeten Technologien für das Rahmenwerk begründet. Abschnitt 4.2 beschreibt die Konzeption und den Entwurf des Rahmenwerkes. Die zentralen Komponenten werden daran anschließend erläutert. Im anschließenden Abschnitt 4.3 werden Erweiterungen für das Rahmenwerk vorgestellt, die nicht mehr realisiert werden konnten.

4.1 Technologien

Die Wahl der Sprache zur Implementierung des Rahmenwerkes zwischen C++ und Python fiel zugunsten von Python[1] in der Version 2.4.1 aus. Python wird mittlerweile auch im wissenschaftlich-akademischen Bereich als ernst zu nehmende Alternative zu den etablierten Sprachen akzeptiert[2]. Das liegt zum einen an der Einfachheit der Sprache an sich, die jedoch keineswegs die Mächtigkeit im Vergleich zu anderen Sprachen einschränkt, und an der dadurch erheblich größeren Entwicklungsgeschwindigkeit auch für den Einsatz in Gebieten, in denen bisher C++ dominiert hat[3]. Zum anderen sind viele Konstrukte bereits sprachinhärent, die in anderen Sprachen zum großen Teil selbst zu implementieren sind. Ein Nachteil der Implementierung des Rahmenwerkes in einer interpretierten Sprache wie Python ist die im Allgemeinen geringere Ablaufgeschwindigkeit im Vergleich zu kompilierten Sprachen. Durch die Möglichkeit zeitkritische Teile des Rahmenwerkes in C++ zu schreiben[4] [5] wird dieser Nachteil zu großen Teilen kompensiert. Der größte Engpass bezüglich der Performanz des Rahmenwerkes liegt im Netzwerkverkehr und entsprechenden Latenzzeiten, sodass der Nachteil der geringeren Ablaufgeschwindigkeit einer interpretierten Sprache nicht so schwer wiegt.
Objektpersistenz ist für die Entwicklung des Rahmenwerkes unabdinglich. Für C++ muss Objektpersistenz entweder durch externe Bibliotheken wie der Object Oriented Property Stream Library (OopsLib)[6] bereitgestellt werden oder selbst implementiert werden. In Python ist die Persistenz von Objekten bereits Bestandteil der Sprache.
Die Möglichkeit der Kommunikation der Knoten über ein Netzwerk ist für das Rahmenwerk elementar. Als Kommunikationsbibliothek kommt die *Common Object Request Broker*

[1] Python `http://www.python.org` [06.2005]
[2] z.B. numarray `http://www.stsci.edu/resources/software_hardware/numarray` [06.2005]
[3] Eric C. Newton, *Python for Critical Applications, a Case Study* `http://www.metaslash.com/brochure/recall.html` [06.2005]
[4] Duncan Booth, *Integrating Python, C and C++* `http://www.suttoncourtenay.org.uk/duncan/accu/integratingpython.html` [06.2005]
[5] pyrex `http://www.cosc.canterbury.ac.nz/~greg/python/Pyrex/` [06.2005]
[6] OopsLib `http://www.rcs.hu/rIDE/rOops.htm` [06.2005]

Architecture (CORBA) zum Einsatz. Die CORBA-Spezifikation wird von der Object Management Group (OMG)[7] definiert, die sich die Entwicklung von herstellerunabhängigen und plattformunabhängigen Standards zum Ziel gesetzt hat. Der CORBA-Standard hat bereits weite Verbreitung und Akzeptanz gefunden und ist von zahlreichen Herstellern, sowohl kommerziell als auch frei, implementiert worden. Als CORBA-ORB kommt der CORBA 2.1 konforme und freie omniORB[8] in der Version 4.0.6 in mit der Python-Anbindung omniORBpy 2.6 zum Einsatz.

4.2 Konzeption, Entwurf und Komponenten

Trotz des Prototyp-Charakters des Rahmenwerkes soll es den folgenden Ansprüchen genügen:

- modularer Aufbau und leichte Erweiterbarkeit
  - Ohne großen Aufwand sollen neue Strukturgeneratoren dem System verfügbar gemacht werden können.
  - Der Typ der replizierten Daten soll variabel sein. Im Rahmen dieser Arbeit wurde exemplarisch der Typ „Datei“ implementiert. Denkbar sind z.B. der Typ „Datenbanktabelle“ oder „Prozess“.
- Rekonfiguration zur Laufzeit
  - Neue Knoten sollen dem System hinzugefügt oder aus dem System entfernt werden können.
  - Strukturgeneratoren sollen hinzugefügt oder entfernt werden können.
  - Manuell spezifizierte Voting-Strukturen sollen hinzugefügt oder entfernt werden können.

Diese Anforderungen erfordern eine strukturierte Architektur des Rahmenwerkes, die mit Hilfe der *OMG Unified Modelling Language (UML)*[9] und der Verwendung von Entwurfsmustern erstellt wurde. Entwurfsmuster (*Pattern*) [GHJV94] sind bewährte Lösungsvorschläge für immer wiederkehrende Entwurfsaufgaben in der Softwarearchitektur und unterstützen das organisierte und strukturierte Entwickeln einer Architektur. *Erzeugungsmuster (Creational Patterns)* befassen sich mit der Erzeugung von Objekten. *Strukturmuster (Structural Patterns)* bieten Lösungen für die Anordnung und Komposition von Klassen an, während *Verhaltensmuster (Behavioural Patterns)* die objektinterne Dynamik und Objektinteraktion behandeln. Die Entwurfsmuster *Singleton*, *Factory* und *Facade* kommen auch in der Architektur des Rahmenwerkes zur Anwendung. Das Singleton-Entwurfsmuster gehört zu den Erzeugungsmustern und stellt sicher, dass von einer Klasse nur genau ein Objekt erzeugt wird auf das global zugegriffen werden kann. Insbesondere für zentrale Verwaltungskomponenten wird dieses Entwurfsmuster benutzt. Das Facade-Entwurfsmuster gehört zu den Strukturmustern und bietet eine einheitliche und vereinfachte Schnittstelle zu einer Menge von Schnittstellen eines Subsystems, wodurch die Schnittstellen des Rahmenwerkes übersichtlicher und somit einfacher zu handhaben sind. Das Factory-Entwurfsmuster gehört zu

[7] OMG `http://www.omg.org` [06.2005]
[8] omniORB `http://omniORB.sourceforge.net` [11.2005]
[9] OMG UML `http://www.uml.org` [06.2005]

den Erzeugungsmustern und produziert Objekte eines bestimmten Typs, die je nach Kontext verschiedene spezialisierte Objekte des gemeinsamen Typs sind. Einige Entwurfsmuster wie z.B. das Factory-Entwurfsmuster sind in Python sprachinhärent, andere lassen sich leicht anwenden [Sav97].

In den folgenden Abschnitten werden die Komponenten und Klassen des Rahmenwerkes vorgestellt. Die Erläuterungen betrachten nur die konzeptionelle Seite des Rahmenwerkes. Genauere Informationen sowie technische Details sind der API-Dokumentation des Rahmenwerkes zu entnehmen.

Abbildung 4.1 zeigt das Klassendiagramm des Rahmenwerkes, das einen Überblick über die Komponenten und deren Zusammenspiel gibt. Wegen der besseren Übersichtlichkeit sind die Signaturen der Methoden und konzeptionell nicht entscheidende Methoden und Attribute nicht dargestellt.

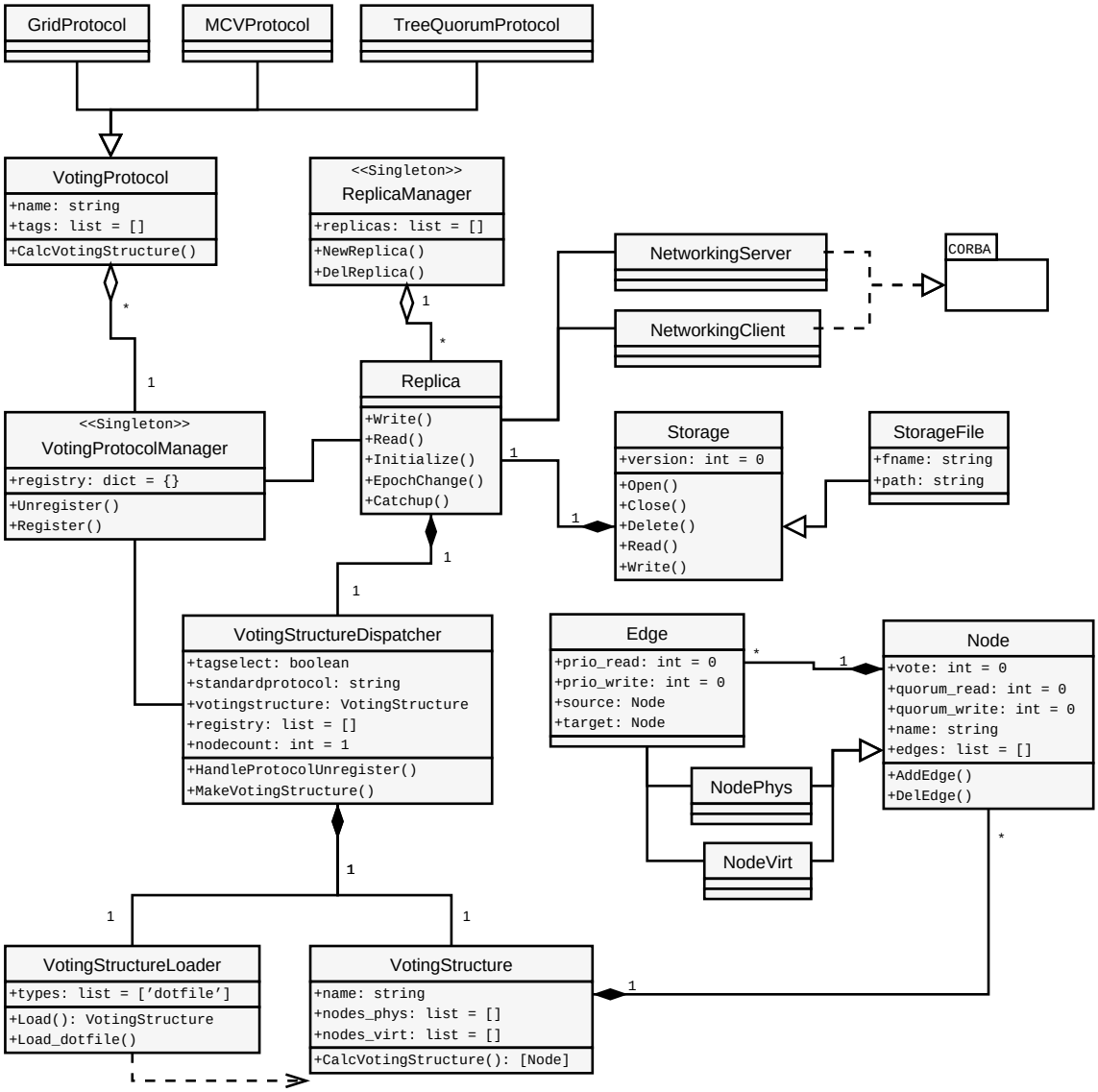


Abbildung 4.1: Klassendiagramm des Rahmenwerkes

Replica und ReplicaManager Die REPLICA-Klasse ist die zentrale Verwaltungsstelle für einen an einem Replikationsverfahren teilnehmenden Knoten. Die Methoden `EpochChange()`, `CatchUp()`, `Read()` und `Write()` implementieren die Durchführung eines Epochenwechsels, die Durchführung der Aktualisierungsoperation sowie die Durchführung einer Lese- und Schreiboperation gemäß den Beschreibungen in Abschnitt 3.2.
Eine REPLICAMANAGER-Instanz verwaltet zentral alle REPLICA-Instanzen eines physikalischen Rechners. Ein physikalischer Rechner kann mehrere Knoten verwalten, die an verschiedenen Replikationsverfahren teilnehmen. Durch die `NewReplica()`-Methode wird ein neuer Knoten erzeugt. Der neue Knoten initialisiert sich dann mittels der `Initialize()`-Methode der REPLICA-Klasse wie in Abschnitt 3.3.1 beschrieben anhand eines bereits an einem Replikationsverfahren teilnehmenden Knotens. Die `DelReplica()`-Methode entfernt einen Knoten aus einem Replikationsverfahren gemäß der Beschreibung in Abschnitt 3.3.2.

Node, NodePhys, NodeVirt, Edge und VotingStructure Die Klasse NODE ist die abstrakte Basisklasse für NODEPHYS- und NODEVIRT-Klassen. Die NODEPHYS-Klasse repräsentiert einen physikalischen Knoten und die NODEVIRT-Klasse einen virtuellen Knoten. In der NODE-Klasse werden gemeinsame Attribute für die NODEPHYS- und NODEVIRT-Klassen wie die Stimmenanzahl `vote`, die Mindeststimmenanzahl für ein Lesequorum `quorum_read`, die Mindeststimmenanzahl für ein Schreibquorum `quorum_write` und die Liste der ausgehenden Kanten `edges` definiert. Die Klasse EDGE beschreibt eine Kante zwischen zwei NODE-Klassen mit den entsprechenden Attributen einer Kante wie der Lese- und Schreibpriorität in `prio_read` und `prio_write` sowie den Quell- und Zielknoten der Kante in `source` und `target`. Die VOTINGSTRUCTURE-Klasse ist ein Container für eine Menge von NODEPHYS- und NODEVIRT-Instanzen und stellt den Datentyp für Voting-Strukturen dar. Die Menge der physikalischen Knoten wird in der Liste `nodes_phys` und die Menge der virtuellen Knoten in der Liste `nodes_virt` gespeichert. Die `CalcQuorum()`-Methode der VOTINGSTRUCTURE-Klasse berechnet eine Menge von NODEPHYS-Instanzen, die einem Lese- oder Schreibquorum entspricht.

VotingProtocol Die VOTINGPROTOCOL-Klasse ist als abstrakte Klasse definiert und dient als Basisklasse und Vorlage für die Implementierung von Strukturgeneratoren. Die Methode `CalcVotingStructure()` berechnet aus der ihr übergebenen Menge von NODEVIRT- und NODEPHYS-Instanzen eine strategiekonforme Voting-Struktur für eine Knotenanzahl, die durch die Anzahl der übergebenen NODEPHYS-Instanzen definiert wird. Diese Methode muss von jedem abgeleitetem Strukturgenerator überschrieben werden. Das Attribut `tags` enthält die Attributierung des Strukturgenerators. Im Klassendiagramm in Abbildung 4.1 sind die drei Strukturgeneratoren für das GP-Verfahren, das TQP-Verfahren und das MCV-Verfahren dargestellt.
Das folgende Listing zeigt die Methode `CalcVotingStructure()` der MCVPROTOCOL-Klasse, die Voting-Strukturen gemäß dem MCV-Verfahren konstruiert.

```
1  def CalcVotingStructure(self, nodes_phys=None, nodes_virt=None):
2     def sort_by_attr(seq, attr):
3        intermed = [(getattr(seq[i], attr), i, seq[i]) for i in xrange(len(seq))]
4        intermed.sort()
5        return [ tup[-1] for tup in intermed ]
6
7     if len(nodes_virt) > 0:
8        rootnode = nodes_virt[0]
9        del nodes_virt[1:]
10    else:
11       rootnode = NodeVirt()
12       nodes_virt += [rootnode]
13    rootnode.edges = []
14    rootnode.quorum_read = rootnode.quorum_write = int(math.ceil( float(len(nodes_phys)+1) / 2 )
          )
15
16    nodes_phys = sort_by_attr(nodes_phys, "reliability")
17    nodes_phys.reverse()
18
19    for iterator in nodes_phys:
20       iterator.edges = []
21       iterator.vote = 1
22       iterator.quorum_read = iterator.quorum_write = 0
23       rootnode.AddEdge(target=iterator, prio_read=0, prio_write=0)
24
25    return nodes_phys, nodes_virt
```

Listing 4.1: VotingProtocol für das Majority Consensus Voting-Verfahren

Die Funktion `sort_by_attr()` in den Zeilen 3 bis 5 sortiert die als ersten Parameter übergebene Liste gemäß des als zweiten Parameters übergebenen Attributs und liefert die sortierte Liste zurück. Falls die Liste der virtuellen Knoten mehr als ein Element enthält, wird das erste Element der Liste als Wurzelknoten definiert und der Rest der Liste gelöscht (Zeilen 8 und 9). Wenn die Liste leer ist, dann wird ein neuer virtueller Knoten erzeugt und als Wurzelknoten definiert (Zeilen 11 und 12). In Zeile 13 wird die Liste der ausgehenden Kanten des Wurzelknotens geleert. In Zeile 14 wird die Mindeststimmenanzahl des Wurzelknotens für ein Lese- und Schreiboperationen berechnet. In Zeile 16 werden die physikalischen Knoten gemäß ihrer Zuverlässigkeit sortiert. Die zuverlässigsten Knoten befinden sich danach am Anfang der Liste (Zeile 17). Die Schleife in den Zeilen 20 bis 23 iteriert über die Liste der physikalischen Knoten, wobei die die Liste der ausgehenden Kanten gelöscht wird, die Stimme des Knotens auf eins gesetzt wird und die Mindeststimmenanzahlen auf null gesetzt werden. Danach wird in Zeile 23 der Wurzelknoten mit dem physikalischen Knoten verbunden. In Zeile 25 werden die beiden Listen als Ergebnis zurückgegeben.

VotingProtocolManager Die VotingProtocolManager-Klasse ist der zentrale Verwaltungspunkt für VotingProtocol-Instanzen und für das dynamische Laden und Entladen von VotingProtocol-Instanzen zuständig. Durch den Plugin-Mechanismus können

zur Laufzeit neue Strukturgeneratoren mittels der `Register()`-Methode in das Rahmenwerk geladen und vorhandene Strukturgeneratoren durch die `Unregister()`-Methode entfernt werden. Das Attribut `registry` enthält Verweise auf alle zur Verfügung stehenden VotingProtocol-Instanzen. Es darf nur eine VotingProtocolManager-Instanz pro Knoten geben.

VotingStructureDispatcher und VotingStructureLoader Die VotingStructureDispatcher-Klasse ist der zentrale Verwaltungspunkt für manuell spezifizierte Voting-Strukturen und Strukturgeneratoren. Aufgrund der Attribute dieser Klasse wird entschieden wie eine Voting-Struktur erzeugt wird. Das Attribut `registry` realisiert das in Abschnitt 3.1 beschriebene Register. Der Standardstrukturgenerator ist im Attribut `standardprotocol` festgelegt. Das Attribut `tagselect` bestimmt ob bei einem fehlenden Eintrag im Register der Standardstrukturgenerator oder ein entsprechend attributierter Strukturgenerator benutzt wird. Durch das Setzen des Attributs `nodecount` auf einen negativen Wert wird der dGSV-Emulationsmodus aktiviert. Die Methode `MakeVotingStructure()` implementiert den Entscheidungsalgorithmus. Wenn zur Laufzeit eine VotingProtocol-Instanz durch Aufruf der `Unregister()`-Methode der VotingProtocolManager-Klasse entfernt wird, dann wird vom VotingProtocolManager die Methode `HandleProtocolUnregister()` jeder VotingStructureDispatcher-Instanz aufgerufen, in der ein neuer Standardstrukturgenerator ausgewählt wird, falls dieser entfernt wurde.
Die VotingStructureLoader-Klasse lädt mittels der `Load()`-Methode eine manuell spezifizierte Voting-Struktur und registriert diese im Register der VotingStructureDispatcher-Instanz. Die generische `Load()`-Methode delegiert das eigentliche Laden der Voting-Struktur an spezialisierte Methoden, die jeweils Voting-Strukturen eines bestimmten Datenformats laden können. Die unterstützen Datenformate sind im Attribut `types` definiert. Das Rahmenwerk unterstützt momentan nur das Datenformat „dotfile“, das die Syntax der *DOT*-Beschreibungssprache von GraphViz[10] benutzt. Die `Load_dotfile()`-Methode lädt Voting-Strukturen im Datenformat „dotfile“.
Das folgende Listing zeigt die Definition und Syntax einer Voting-Struktur gemäß dem MCV-Verfahren für drei physikalische Knoten.

```
1  digraph "MajorityConsensus" {
2
3     // Anzahl physikalischer Knoten für die
4     // diese Voting-Struktur benutzt werden soll.
5     numphysicalnodes=3;
6
7     // Definition der physikalischen Knoten
8     // (in Klammern Standardwerte bei Nichtangabe)
9     // Attribut 'type': immer "physical"
10    // Attribut 'vote': Stimmenanzahl (Standard: 1)
11
12    R1 [type="physical"];
13    R2 [type="physical"];
14    R3 [type="physical"];
15
```

[10] GraphViz `http://www.graphviz.org` [11.2005]

```
16    // Definition der virteller Knoten:
17    // Attribut 'type': immer "virtual"
18    // Attribut 'vote': Stimmenanzahl (Standard: 1)
19    // Attribut 'quorum_read': Lesequorum (Standard: 0)
20    // Attribut 'quorum_write': Schreibquorum (Standard: 0)
21
22    V1 [vote=1,type="virtual",quorum_read=2,quorum_write=2];
23
24    // Definition der Kanten zwischen virtuellen und
25    // physikalischen Knoten:
26    // Attribut 'prio_read': Lesepriorität (Standard: 0)
27    // Attribut 'prio_write': Schreibpriorität (Standard: 0)
28    V1->R1 [prio_read=0, prio_write=0];
29    V1->R2 [prio_read=0, prio_write=0];
30    V1->R3 [prio_read=0, prio_write=0];
31 }
```

Listing 4.2: Definition einer Voting-Struktur gemäß des MCV-Verfahrens für drei physikalische Knoten

Zeile 1 enthält das Schlüsselwort `digraph` und den Namen der Voting-Struktur. Die abschließende Klammer leitet den Definitionsteil ein. In Zeile 5 wird festgelegt für welche Knotenanzahl die Voting-Struktur in das Register eingetragen werden soll. In den Zeilen 12 bis 14 werden die drei physikalischen Knoten $R1$, $R2$ und $R3$ mit der Standardstimme eins definiert. In Zeile 22 wird der virtuelle Wurzelknoten $V1$ mit der Stimme eins und der Mindeststimmenanzahl von zwei Knoten für Lese- und Schreibquoren definiert. Die Kanten von $V1$ zu den drei physikalischen Knoten werden in den Zeilen 28 bis 30 mit der Lese- und Schreibpriorität von jeweils null definiert. Die abschließende Klammer in Zeile 31 schließt den Definitionsteil ab.

Storage und StorageFile Die abstrakte Basisklasse STORAGE definiert eine Vorlage für die Unterstützung von verschiedenen Typen replizierter Daten. Die von der STORAGE-Klasse abgeleitete Klasse STORAGEFILE implementiert die Unterstützung für den Typ „Datei“. Eine von STORAGE abgeleitete Klasse muss die Methoden `Open()`, `Close()`, `Delete()`, `Read()` und `Write()` für das Öffnen, Schließen, Löschen, Lesen und Schreiben von Daten des jeweiligen Typs implementieren.

NetworkingClient und NetworkingServer Die Klassen NETWORKINGCLIENT und NETWORKINGSERVER kapseln die CORBA-Anbindung des Rahmenwerkes. Die gesamte Kommunikation zwischen Knoten findet über Methoden statt, die in diesen beiden Klassen implementiert sind. Durch die Kapselung ist ein leichter Austausch von CORBA durch eine andere Kommunikationsbibliothek wie z.B. twisted[11] möglich.

[11] twisted `http://twistedmatrix.com/` [11.2005]

4.3 Erweiterungen

In den folgenden Abschnitten werden Erweiterungen des Funktionsumfangs des Rahmenwerkes vorgestellt, die im Rahmen dieser Arbeit nicht implementiert wurden.

Sicherheit Bei der Implementierung des Rahmenwerkes wurden keinerlei Sicherheitsaspekte wie z.B. die Zugriffskontrolle und Authentifikation von Benutzern oder die Verschlüsselung der Kommunikation zwischen den Knoten berücksichtigt. Die Erweiterung des Rahmenwerkes um Sicherheitsaspekte ist im Nachhinein durch die modulare Architektur des Rahmenwerkes möglich.

Einbettung in bestehende Systeme FUSE[12] ermöglicht für Linux- und FreeBSD-Betriebssysteme die Implementierung eines Dateisystems als Programm im Userspace. Dazu wird das FUSE-Kernelmodul geladen, das sich in der *Virtual File System (VFS)*-Schicht des Kernels registriert und als Mittler zwischen VFS-Schicht und dem Programm fungiert. Die vom Kernelmodul empfangenen Funktionsaufrufe werden an das Programm im Userspace weiterleitet, das die Aufrufe bearbeitet und das Ergebnis an das Kernelmodul zurückgibt. Ein mittels FUSE implementiertes Dateisystem verhält sich genauso wie jedes andere in der VFS-Schicht registrierte Dateisystem. Über die Python-Anbindung von FUSE kann das Rahmenwerk zu einem verteilten Dateisystem ausgebaut werden. Dazu muss eine Zwischenschicht dem Rahmenwerk hinzugefügt werden, die die Funktionsaufrufe der VFS-Schicht in entsprechende Aufrufe von Funktion des Rahmenwerkes übersetzt. Ausserdem müssen noch fehlende Funktionen zur Dateiverwaltung ergänzt werden.

verteilter Namensdienst Der verwendete CORBA-ORB omniORB bietet nur einen zentralen Namensdienst. Wenn der den Namensdienst anbietende Rechner ausfällt, dann ist z.B. das Hinzufügen neuer Knoten in das Replikationsverfahren nicht mehr möglich. Um diesen *Single Point of Failure* zu vermeiden, ist die Implementierung eines verteilten Namensdienstes oder die Verwendung einer geeigneten Erweiterung für omniORB sinnvoll.

4.4 Zusammenfassung

Das implementierte Rahmenwerk besitzt alle geforderten und wesentlichen Merkmale, um die Eigenschaften des adGSV-Verfahrens zu testen und kann als Basis für die Weiterentwicklung des adGSV-Verfahrens dienen. Die modulare Architektur erlaubt die leichte Erweiterbarkeit und Wartung des Rahmenwerkes.
Im folgenden Kapitel werden Messergebnisse des adaptive dynamic General Structured Voting-Verfahrens im Vergleich zu anderen vorgestellten Verfahren präsentiert, die mit dem im Rahmenwerk enthaltenen Simulator ermittelt wurden.

[12] FUSE `http://fuse.sourceforge.net/` [06.2005]

5 Simulation

In das im vorherigen Kapitel vorgestellte Rahmenwerk wurde ein Simulator integriert, um die Eigenschaften des adaptive dynamic General Structured Voting-Verfahrens im Vergleich zu anderen Verfahren messen zu können. Die gemessenen Ergebnisse werden in diesem Kapitel präsentiert.

Das verwendete Testverfahren und die getroffenen Annahmen werden im folgenden Abschnitt erläutert. In Abschnitt 5.2 werden zuerst die Messergebnisse der homogenen Verfahren Dynamic Voting, dynamic Grid Protocol und Tree Quorum Protocol mit jeweils neun Knoten vorgestellt. Anschließend wird eine inhomogene Konfiguration des adaptive dynamic General Structured Voting-Verfahrens, ebenfalls mit neun Knoten, für den Vergleich von homogenen und inhomogenen Verfahren präsentiert. Der Abschnitt schließt mit der direkten Gegenüberstellung der gemessenen Operationsverfügbarkeiten aller Verfahren. Abschnitt 5.3 stellt ein Beispielszenario vor, in dem der adaptive Aspekt des adGSV-Verfahrens in Bezug auf das Hinzufügen und Entfernen von Knoten in einer homogenen Konfiguration mit dem dGP-Verfahren verglichen wird.

5.1 Simulationsverfahren und Voraussetzungen

Die Simulation der Verfahren wurde mittels einer diskreten ereignisbasierten Simulation durchgeführt. Der Simulator des Rahmenwerkes benutzt dafür das Simulationsrahmenwerk SimPy[1].
Für alle vorgenommenen Messungen gelten die folgenden Annahmen und Voraussetzungen: Die durchschnittliche Zeit bis zur Reparatur eines ausgefallenen Knotens (*mean time to repair (MTTR)*) wird mit $MTTR_r = 1.201$ Tagen angenommen [LMG95]. Die durchschnittliche Zeit bis zum Ausfall eines Knotens (*mean time to failure (MTTF)*) errechnet sich aus $MTTR_r$ und dem Verfügbarkeitsgrad $pr_v \in (0, 1]$ des Knotens zu $MTTF_r = \frac{MTTR_r \cdot pr_v}{1-pr_v}$. Der Ausfall und die Wiederherstellung von Knoten wurde mit exponential verteilten Zufallszahlen simuliert, wobei der *random seed* des Zufallszahlengenerators auf 12345 gesetzt wurde. Die Anzahl der erfolgreich ausgeführten Leseoperationen (Schreiboperationen) bezeichnet or_s (ow_s). Die Anzahl der fehlgeschlagenen Leseoperationen (Schreiboperationen) bezeichnet or_f (ow_f). Daraus errechnet sich der Verfügbarkeitsgrad der Leseoperation (Schreiboperation) zu $v_r = \frac{or_s}{or_s+or_f}$ ($v_w = \frac{ow_s}{ow_s+ow_f}$). Aus dem Verfügbarkeitsgrad einer Operationen wird die MTTF der Operationen berechnet. Die MTTF für Leseoperationen (Schreiboperationen) ist $MTTF_{read} = \frac{v_r \cdot MTTR_r}{1-v_r}$ ($MTTF_{write} = \frac{v_w \cdot MTTR_r}{1-v_w}$). Operationen werden nur von Knoten initiiert, die selbst ein Replikat besitzen und am Replikationsverfahren teilnehmen. Die Auswahl des Knotens, der eine Operation durchführt, wird per Zufall getroffen, wobei nur verfügbare Knoten berücksichtigt werden. Das Intervall zwischen zwei aufeinander folgende Operationen ist auf 0.35 Zeiteinheiten gesetzt. In Abbildung 5.1 ist die durchschnittliche Anzahl Operationen, an denen ein Knoten als Teilnemher oder Initiator

[1] SimPy `http://simpy.sourceforge.net/` [11.2005]

pr_v	MTTF	Anzahl Operationen
0.1	0.133	0.3813
0.2	0.300	0.8579
0.3	0.515	1.4706
0.4	0.801	2.2876
0.5	1.201	3.4314
0.6	1.801	5.1471
0.7	2.802	8.0067
0.8	4.804	13.7257
0.9	10.809	30.8829

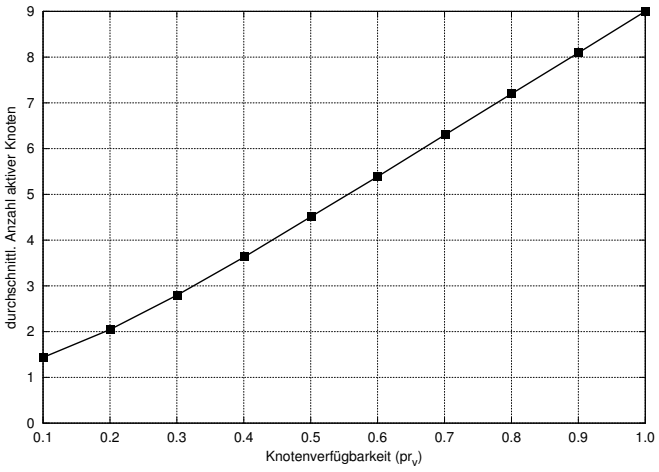
Abbildung 5.1: Resultierende MTTF und durchschnittliche Anzahl ausgeführter oder daran teilgenommener Operationen vor dem Ausfall eines Knotens in Abhängigkeit vom Verfügbarkeitsgrad pr_v des Knotens

beteiligt ist, und die resultierende MTTF in Abhängigkeit von der Knotenverfügbarkeit pr_v dargestellt. Für eine Knotenverfügbarkeit von z.B. $pr_v = 0.9$ und $MTTF_r = 10.809$ werden im Durchschnitt 30.883 Operationen, an denen ein Knoten als Teilnemher oder Initiator beteiligt ist, ausgeführt bevor der Knoten ausfällt. Bevor die Simulation gestartet wird werden alle Knoten auf einen gleichen Status initialisiert und sind verfügbar. Danach wird eine Einschwingphase von 200 Operationen durchgeführt, in der erfolgreiche oder fehlgeschlagene Operationen noch nicht gezählt werden. So werden Störeinflüsse vermieden, die durch die Zählung beginnend ab der ersten Operation entstehen. Für kleine Verfügbarkeitsgrade würden die initialen Operationen die Resultate deutlich verbessern, da anfangs noch viele Knoten verfügbar sind und somit Operationen erfolgreich ausgeführt werden können, wohingegen sie nach der Einschwingphase fehlschlagen. Das zugrunde liegende Netzwerk ist voll verbunden, d.h. jeder Knoten kann direkt mit jedem anderen Knoten kommunizieren. Die Messwerte stellen unter diesen idealen Bedingungen somit obere Schranken dar.

5.2 Messergebnisse vorgestellter Verfahren mit jeweils neun Knoten

Die Messungen der homogenen Verfahren Dynamic Voting, dynamic Grid Protocol und Tree Quorum Protocol und der inhomogenenen Konfiguration des adaptive dynamic General Structured Voting-Verfahrens wurden mit jeweils neun Knoten durchgeführt, um die Vergleichbarkeit der Messergebnisse herzustellen.
Obwohl die numerischen Unterschiede der folgenden Simulationsresultate zwischen zwei Verfahren sehr klein sein mögen, sind sie, in Tagen der Verfügbarkeit einer Operation umgerechnet, doch recht groß. Für $pv_r = 0.8$ wurde z.B. beim dGP-Verfahren die Schreiboperationsverfügbarkeit von $v_w = 0.966$ gemessen und beim adGSV-Verfahren wurde $v_w \backsim 0.987$ gemessen. Daraus ergibt sich $MTTF_{write} \backsim 34.123$ Tage beim dGP-Verfahren und $MTTF_{write} = 88.847$ Tage für das adGSV-Verfahren. Der Unterschied von $\backsim 0.021$ in den gemessenen Schreiboperationsverfügbarkeiten ergibt einen Unterschied von $\backsim 54.724$ Tagen in Bezug auf die durchschnittliche Zeit bis die Operation nicht mehr verfügbar ist.
Abbildung 5.2(a) zeigt die durchschnittliche Anzahl verfügbarer Knoten in Abhängigkeit der gemessenen Knotenverfügbarkeit $p\tilde{r}_v$ bei vorgegebener Knotenverfügbarkeit pr_v für einen bis neun Knoten. Mit steigender Knotenverfügbarkeit steigt auch die durchschnittliche Anzahl verfügbarer Knoten fast linear an.

Die aktuelle Epochennummer nach dem Ende der Simulation wird in den Messergebnissen durch e_r für Lese- und e_w für Schreiboperationen angegeben.



(a) Graph der durchschnittlichen Anzahl verfügbarer Knoten in Abhängigkeit vom gemessenen Verfügbarkeitsgrad $p\tilde{r}_v$

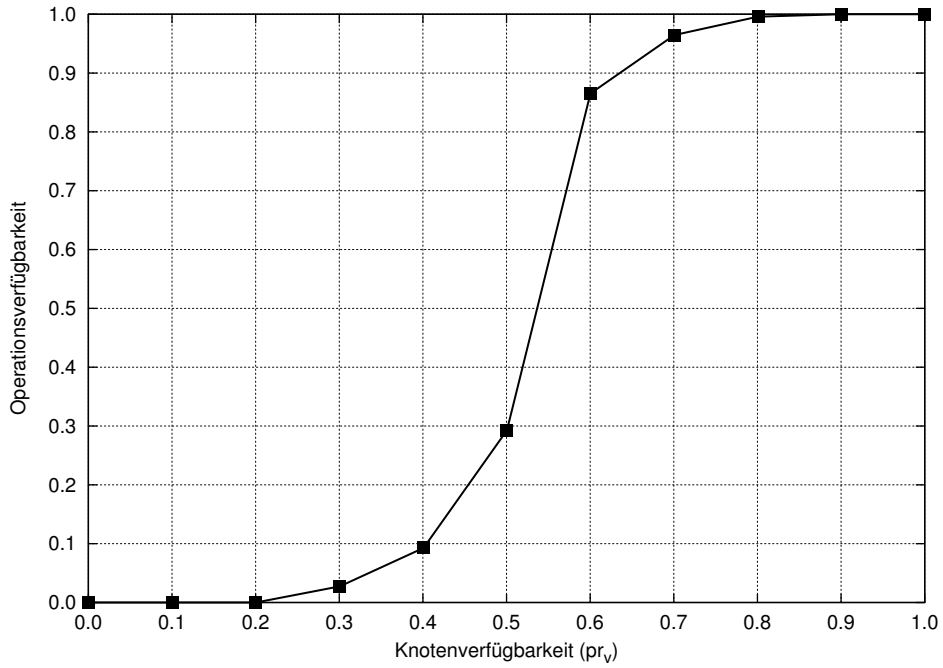
pr_v	$p\tilde{r}_v$	Anzahl Knoten
0.1	0.1002	1.4373
0.2	0.2004	2.0477
0.3	0.3007	2.7996
0.4	0.4009	3.6325
0.5	0.5005	4.5138
0.6	0.6003	5.3914
0.7	0.7007	6.3029
0.8	0.8008	7.2047
0.9	0.9005	8.0988

(b) Tabelle der durchschnittlichen Anzahl verfügbarer Knoten und gemessener Knotenverfügbarkeit $p\tilde{r}_v$ inAbhängigkeit der vorgegebenen Knotenverfügbarkeit pr_v

Abbildung 5.2: Durchschnittliche Anzahl verfügbarer Knoten und gemessene Knotenverfügbarkeit $p\tilde{r}_v$ in Abhängigkeit der vorgegebenen Knotenverfügbarkeit pr_v

5.2.1 Dynamic Voting-Verfahren

Das Dynamic Voting-Verfahren, das dynamische Gegenstück zum statischen Majority Consensus-Voting-Verfahren, dient als Referenzstrategie für alle anderen Verfahren, da es ein unstrukturiertes Verfahren ist und somit die Anzahl möglicher Schreib- und Lesequoren nur von der Anzahl verfügbarer Knoten abhängt und nicht strukturinhärent ist.
Die generierten Voting-Strukturen sind in Abbildung 5.4 dargestellt. In Abbildung 5.3 sind die gemessenen Werte tabellarisch und graphisch dargestellt.



(a) Graph der Operationsverfügbarkeit in Abhängigkeit der Knotenverfügbarkeit des Dynamic Voting-Verfahrens

pr_v	$v_{r,w}$	$e_{r,w}$
0.1	0.0000	3
0.2	0.0000	1
0.3	0.0273	58
0.4	0.0927	139
0.5	0.2927	646
0.6	0.8653	1820
0.7	0.9647	1577
0.8	0.9960	1082
0.9	1.0000	518

(b) Tabelle der Operationsverfügbarkeit in Abhängigkeit der Knotenverfügbarkeit des Dynamic Voting-Verfahrens

Abbildung 5.3: Messergebnisse des Dynamic Voting-Verfahrens mit neun Knoten

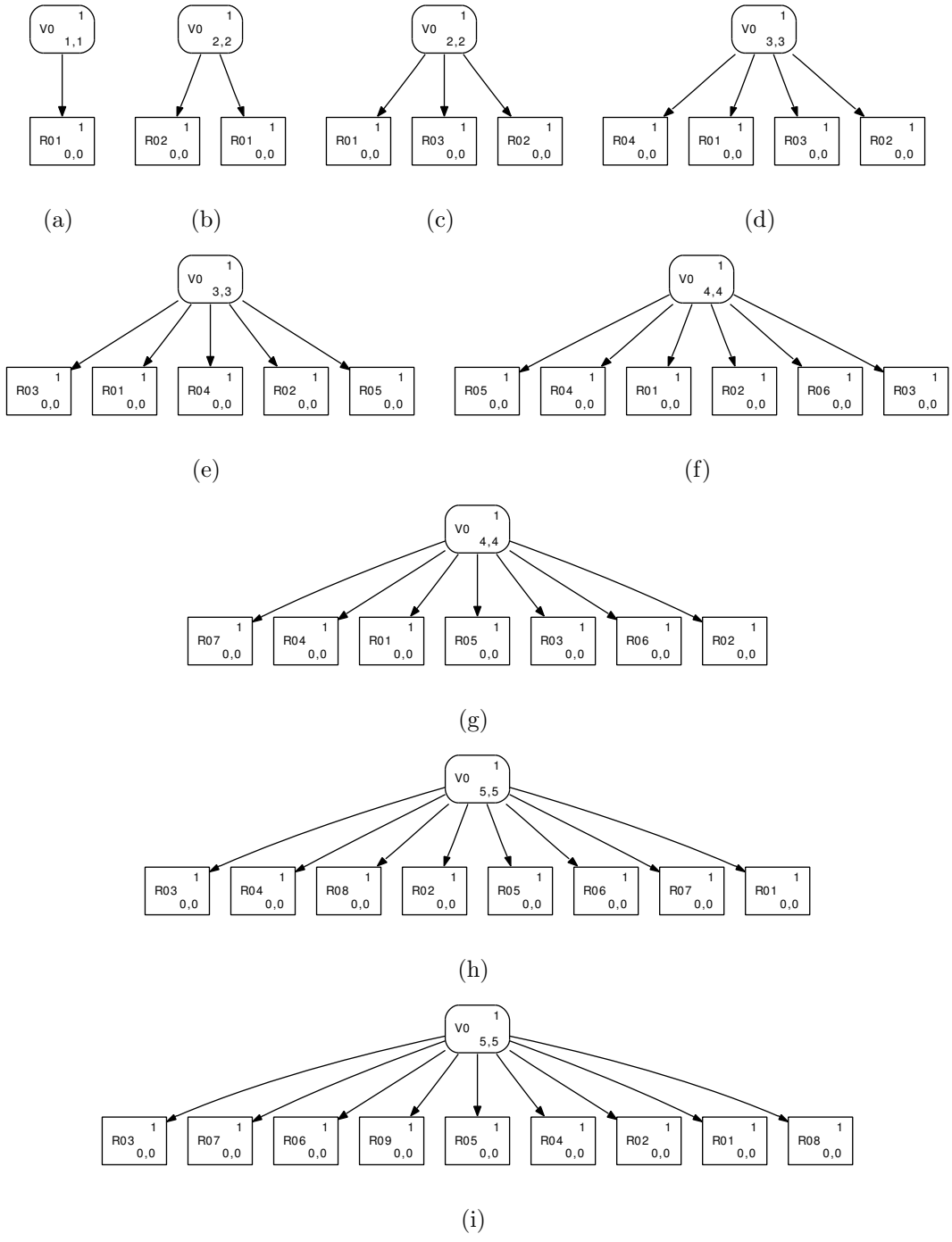
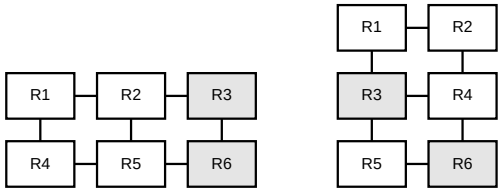


Abbildung 5.4: Voting-Strukturen des Dynamic Voting-Verfahrens für einen (a) bis neun Knoten (f)

5.2.2 dynamic Grid Protocol-Verfahren

Beim dynamic Grid Protocol-Verfahren ist die Operationsverfügbarkeit bei Knotenausfällen von der Position der Knoten in der Voting-Struktur abhängig. In Abbildung 5.5 sind zwei mögliche Anordnungen für das dGP-Verfahren mit sechs Knoten dargestellt. Bei Ausfall der Knoten R3 und R6 ist keine Konstruktion eines Schreibquorums mehr möglich, während beim DV-Verfahren mit sechs Knoten noch ein Schreibquorum gebildet werden kann. Die Anzahl möglicher Lese- und Schreibquoren ist durch die Struktur im Vergleich zum unstrukturierten DV-Verfahren kleiner. Das erklärt die geringere Operationsverfügbarkeit im Vergleich zum DV-Verfahren sowohl für Lese- als auch für Schreiboperationen.



(a) Anordnung der Knoten in einem 3×2 Gitter

(b) Anordnung der Knoten in einem 2×3 Gitter

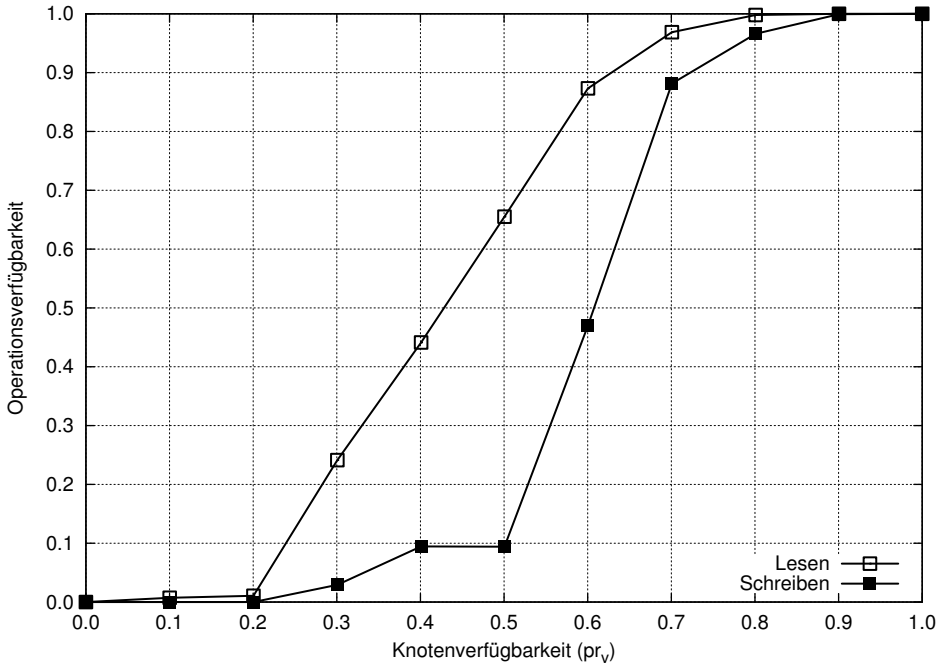
Abbildung 5.5: Zwei mögliche Knotenanordnungen für das GP-Verfahren mit 6 Knoten

	$pr_v = 0.4$	$pr_v = 0.5$
1	-	-
2	10	2
3	54	5
4	60	6
5	47	11
6	24	10
7	6	3
8	1	1
9	1	1
Σ	206	29

Abbildung 5.6: Anzahl der generierten Voting-Strukturen für die Knotenanzahlen eins bis neun für die Knotenverfügbarkeiten $pr_v = 0.4$ und $pr_v = 0.5$

Die generierten Voting-Strukturen sind in Abbildung 5.8 dargestellt. In Abbildung 5.7 sind die gemessenen Werte tabellarisch und graphisch dargestellt. Die in etwa gleichbleibende Schreiboperationsverfügbarkeit zwischen $pv_r = 0.4$ und $pv_r = 0.5$ erklärt sich durch den Vergleich der Anzahl der Epochenwechsel. Für $pv_r = 0.4$ fanden 192 Epochenwechsel statt, während für $pv_r = 0.5$ nur 36 stattfanden. Die Voting-Strukturen für $pv_r = 0.4$ enthalten durchschnittlich weniger ausgefallenene Knoten und die Bildung eines Schreibquorums, welches auch für einen Epochenwechsel nötig ist, schlägt somit nicht so oft fehl. Trotz der höheren Knotenverfügbarkeit und durchschnittlich 0.8813 mehr verfügbarer Knoten wird die Schreiboperationsverfügbarkeit für $pv_r = 0.5$ durch durchschnittlich zu viele ausgefallene Knoten in den Voting-Strukturen nicht gesteigert. In Abbildung 5.6 sind die Anzahlen

der generierten Voting-Strukturen für die Knotenanzahlen eins bis neun dargestellt. Darin sind auch die während der Einschwingphase generierten Voting-Strukturen enthalten. Da während eines Epochenwechsels nicht zwingend eine neue Voting-Struktur erzeugt wird, sind die Summen der generierten Voting-Strukturen nicht gleich der jeweiligen Anzahl der Epochenwechsel. Wenn z.B. ein Knoten ausfällt, der bis zur Reparatur nicht Teil eines Lese- oder Schreibquorums ist und so als ausgefallen erkannt wird und der Knoten sich nach seiner Reparatur aktualisiert, dann wird zwar ein Epochenwechsel eingeleitet, jedoch keine neue Voting-Struktur erzeugt.



(a) Graph der Operationsverfügbarkeit in Abhängigkeit der Knotenverfügbarkeit des dynamic Grid Protocol-Verfahrens für Lese- und Schreiboperationen

pr_v	v_r	e_r	v_w	e_w
0.1	0.0072	2	0.0000	2
0.2	0.0106	1	0.0000	2
0.3	0.2412	70	0.0293	40
0.4	0.4413	195	0.0947	192
0.5	0.6551	689	0.0940	36
0.6	0.8732	1138	0.4700	912
0.7	0.9686	1126	0.8813	1382
0.8	0.9980	754	0.9660	996
0.9	1.0000	343	0.9993	487

(b) Tabelle der Operationsverfügbarkeit in Abhängigkeit der Knotenverfügbarkeit des dynamic Grid Protocol-Verfahrens für Lese- und Schreiboperationen

Abbildung 5.7: Messergebnisse des dynamic Grid Protocol-Verfahrens mit neun Knoten

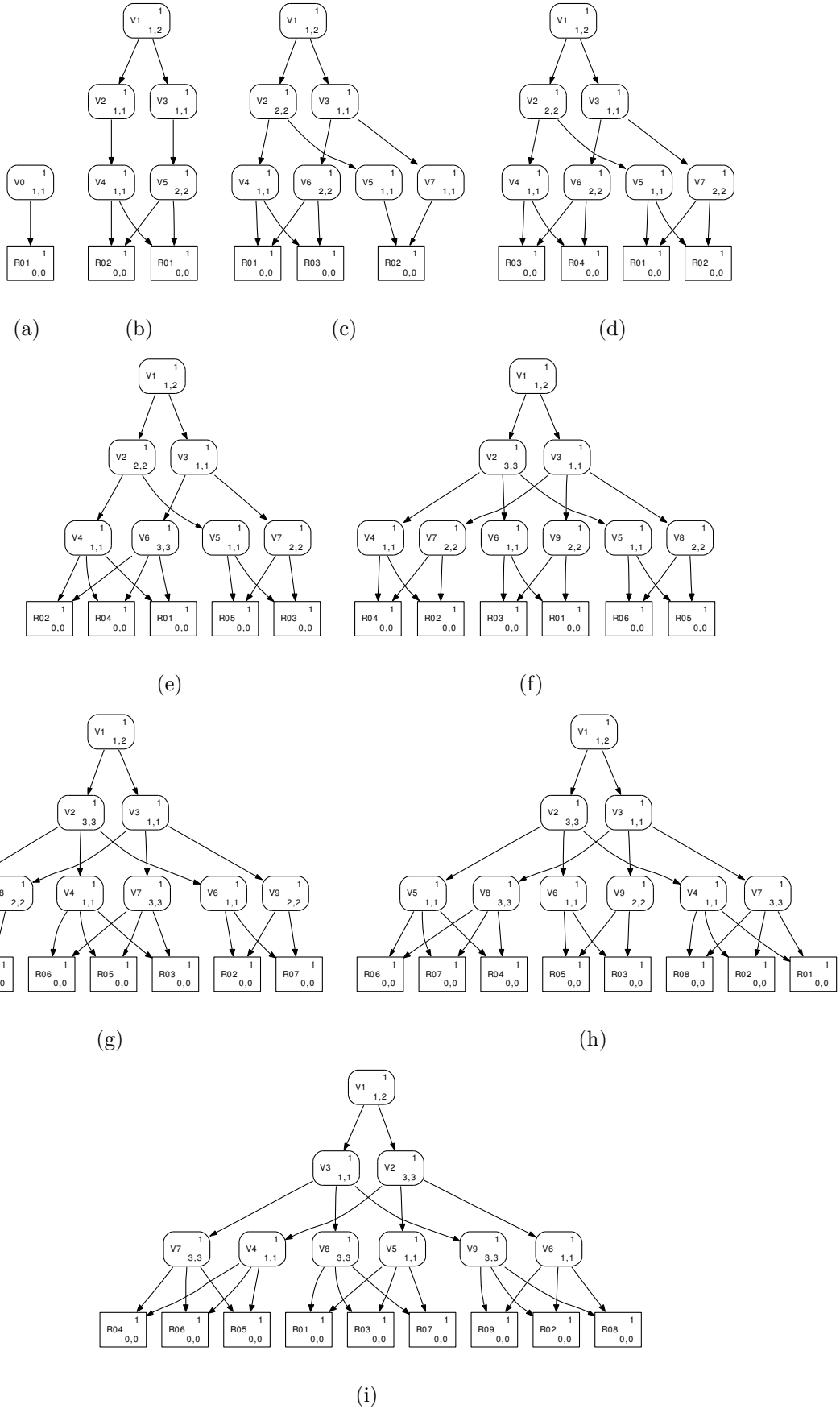
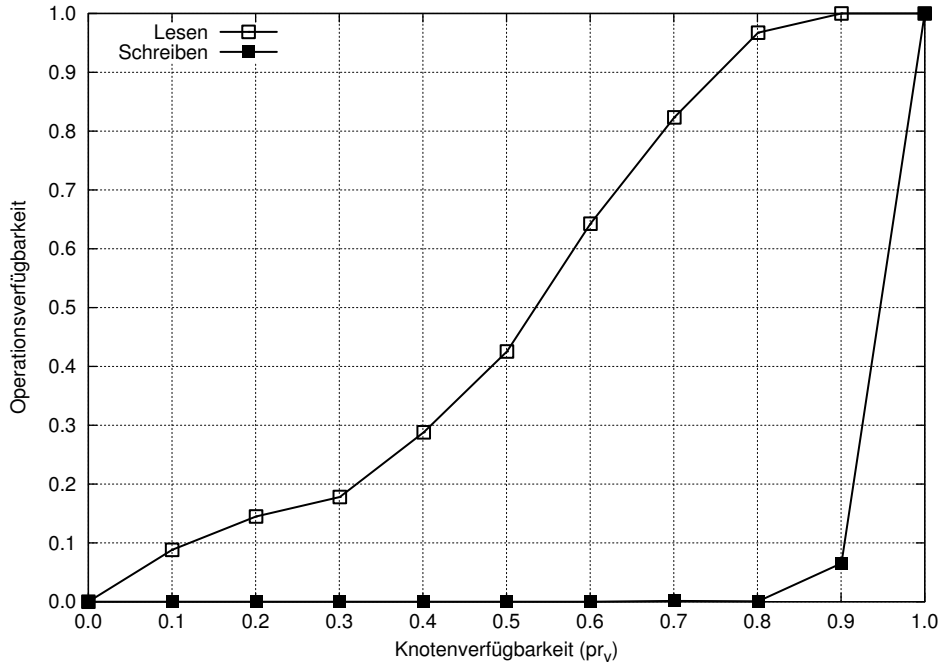


Abbildung 5.8: Voting-Strukturen des dynamic Grid Protocol-Verfahrens für einen (a) bis neun Knoten (i)

5.2.3 Tree Quorum Protocol-Verfahren

Der verwendete Strukturgenerator für das TQP-Verfahren implementiert Voting-Strukturen, die der zweiten Variante der Quorenbildung entsprechen (siehe Abschnitt 2.2.2). Schreibquoren bestehen aus dem Wurzelknoten, der Mehrheit seiner Kinder, der Mehrheit deren Kinder und so fort. Lesequoren bestehen aus dem Wurzelknoten, bei Ausfall des Wurzelknotens aus der Mehrheit seiner Kinder, bei Ausfall eines der Kinder, aus der Mehrheit dessen Kinder und so fort. Die generierten Voting-Strukturen sind in Abbildung 5.10 dargestellt. In Abbildung 5.9 sind die gemessenen Werte tabellarisch und graphisch dargestellt. Für z.B. sieben Knoten führt bereits der Ausfall eines Knotens zum Fehlschlagen der Schreiboperation. Das TQP-Verfahren entspricht hier dem ROWA-Verfahren. Für neun Knoten können zwei Knotenausfälle toleriert werden, jedoch nur von Knoten die sich in der untersten Ebene befinden. Beim TQP-Verfahren kommt es sehr stark auf die Position der ausgefallenen Knoten in der Voting-Struktur an. Besonders wichtige Positionen in der Voting-Struktur, wie z.B. der Wurzelknoten, müssen mit zuverlässigen Knoten besetzt werden. Da in der Simulation alle Knoten den gleichen Verfügbarkeitsgrad besitzen, spielt die Positionierung der Knoten in der Voting-Struktur keine Rolle. Die Optimierung der Schreiboperationsverfügbarkeit durch das Platzieren zuverlässiger Knoten an wichtige Stellen in der Voting-Struktur wird in der Simulation nicht berücksichtigt. Das erklärt z.B. die schlechte Schreiboperationsverfügbarkeit des TQP-Verfahrens im Vergleich zu anderen Strategien.



(a) Graph der Operationsverfügbarkeit in Abhängigkeit der Knotenverfügbarkeit des Tree Quorum Protocol-Verfahrens für Lese- und Schreiboperationen

pr_v	v_r	e_r	v_w	e_w
0.1	0.0882	1	0.0000	1
0.2	0.1450	1	0.0000	2
0.3	0.1779	1	0.0000	3
0.4	0.2880	1	0.0000	2
0.5	0.4253	1	0.0000	3
0.6	0.6427	1	0.0000	4
0.7	0.8233	1	0.0013	5
0.8	0.9673	1	0.0007	5
0.9	1.0000	1	0.0653	33

(b) Tabelle der Operationsverfügbarkeit in Abhängigkeit der Knotenverfügbarkeit des Tree Quorum Protocol-Verfahrens für Lese- und Schreiboperationen

Abbildung 5.9: Messergebnisse des Tree Quorum Protocol-Verfahrens mit neun Knoten

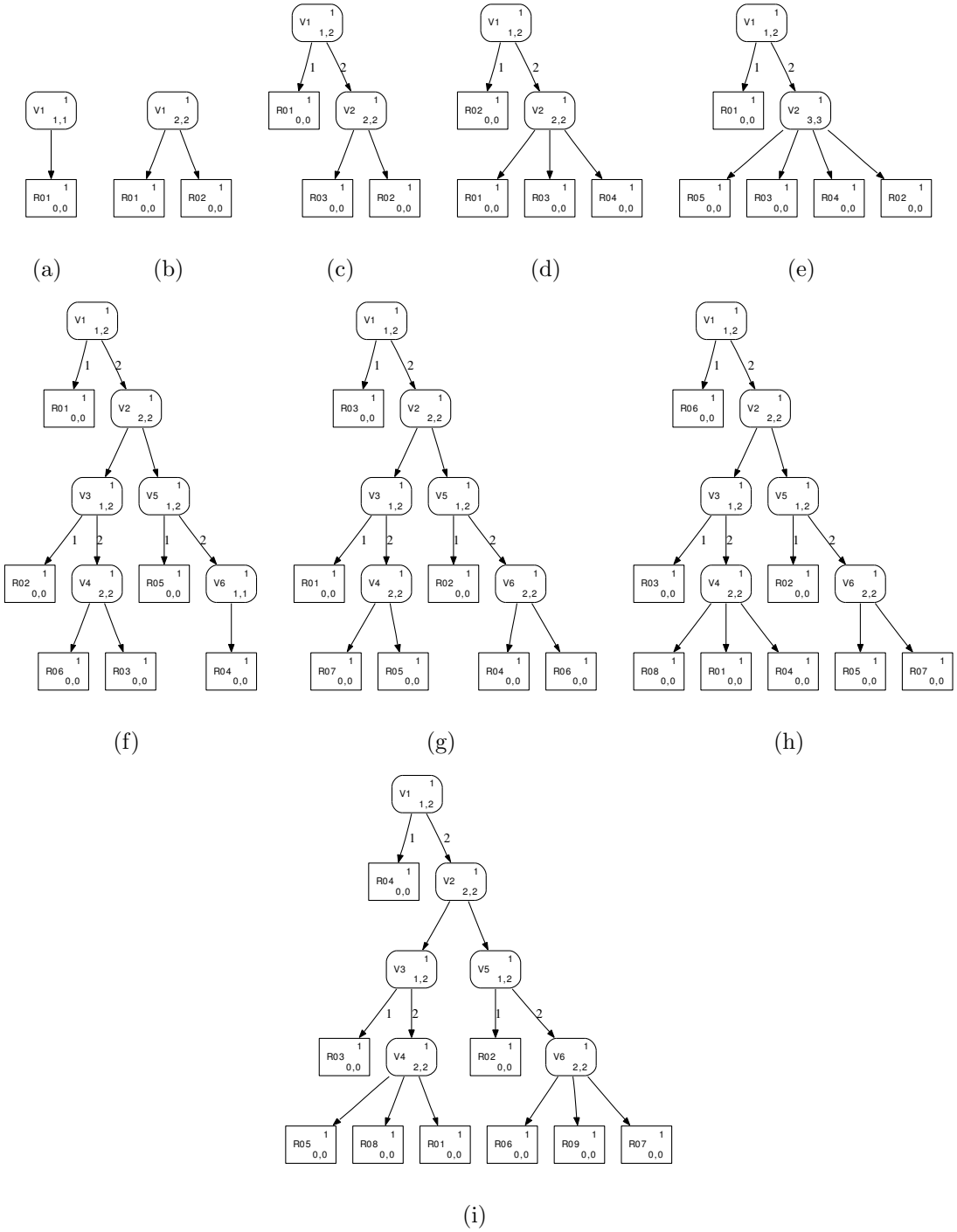
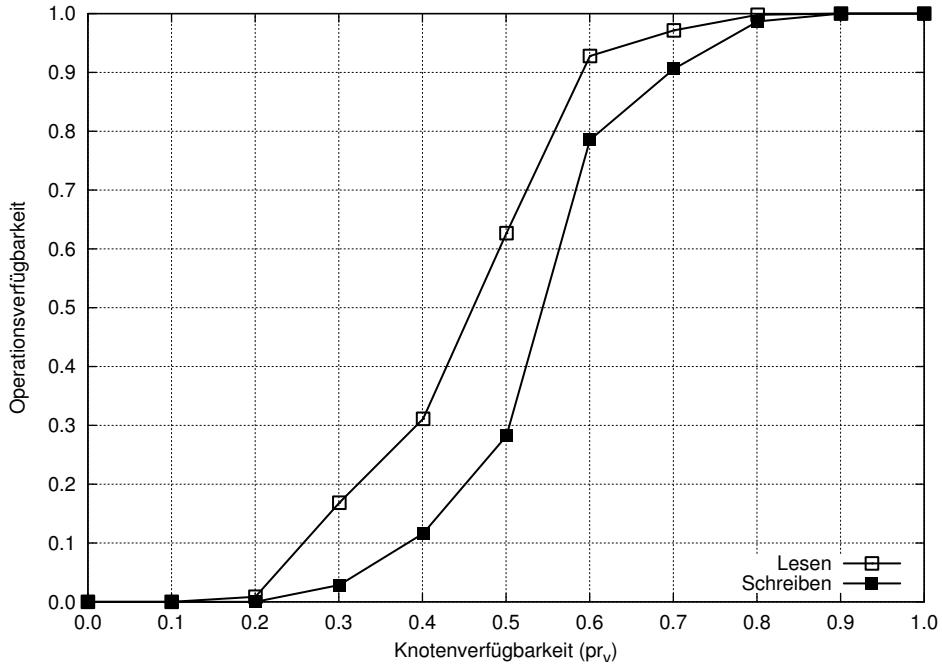


Abbildung 5.10: Voting-Strukturen des Tree Quorum Protocol-Verfahrens für einen (a) bis neun Knoten (i)

5.2.4 adaptive dynamic General Structured Voting-Verfahren

Die hier verwendete inhomogenene Konfiguration des adGSV-Verfahrens wurde im Hinblick auf bessere Schreiboperationsverfügbarkeit im Vergleich zum GP-Verfahren konstruiert. Die verwendeten Voting-Strukturen für die jeweiligen Knotenanzahlen sind in Abbildung 5.12 dargestellt. Für neun Knoten wird das *Triangular Lattice Protocol (TLP)*-Verfahren [TPK95, PT97] verwendet, das im Rahmen dieser Arbeit nicht vorgestellt wird. Das GP-Verfahren wird für vier, sechs und acht Knoten verwendet. Das MCV-Verfahren wird für ungerade Knotenanzahlen, bis auf neun, verwendet. Für Knotenanzahlen kleiner als vier Knoten mit der Ausnahme von zwei Knoten, wo das ROWA-Verfahren zum Einsatz kommt, wird ebenfalls das MCV-Verfahren benutzt. Interessanterweise war die gemessene Operationsverfügbarkeit bei Verwendung des TLP-Verfahrens statt des GP-Verfahrens für vier, sechs und acht Knoten geringer. Als „Faustregel“ erwies sich ein unstrukturiertes Verfahren wie das MCV-Verfahren für ungerade, und ein strukturiertes Verfahren wie das GP-Verfahren oder das TLP-Verfahren für gerade Knotenanzahlen ab vier Knoten zu benutzen. Neun Knoten sind aufrgund der 3×3 Anordnung, die besonders für das GP- oder das TLP-Verfahren geeignet ist, eine Ausnahme.
In Abbildung 5.11 sind die gemessenen Werte tabellarisch und graphisch dargestellt. Die Leseoperationsverfügbarkeit ist für Knotenverfügbarkeiten $pv_r \leq 0.6$ im Vergleich zum dGP-Verfahren kleiner, während sie für die Knotenverfügbarkeiten $pv_r = 0.6$ und $pv_r = 0.7$ größer ist. Für die Knotenverfügbarkeiten $pv_r = 0.8$ und $pv_r = 0.9$ ist die Leseoperationsverfügbarkeit beider Verfahren gleich. Die Schreiboperationsverfügbarkeit ist im Vergleich zum dGP-Verfahren ab einer Knotenverfügbarkeit $pv_r \geq 0.4$ höher, während sie für kleinere Knotenverfügbarkeiten vergleichbar ist.

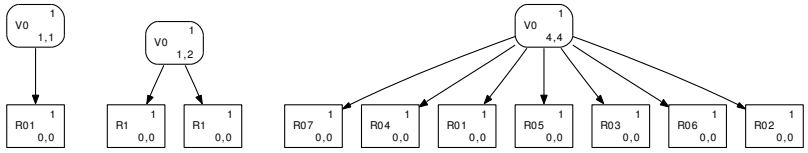


(a) Graph der Operationsverfügbarkeit in Abhängigkeit der Knotenverfügbarkeit des adaptive dynamic General Structured Voting-Verfahrens für Lese- und Schreiboperationen

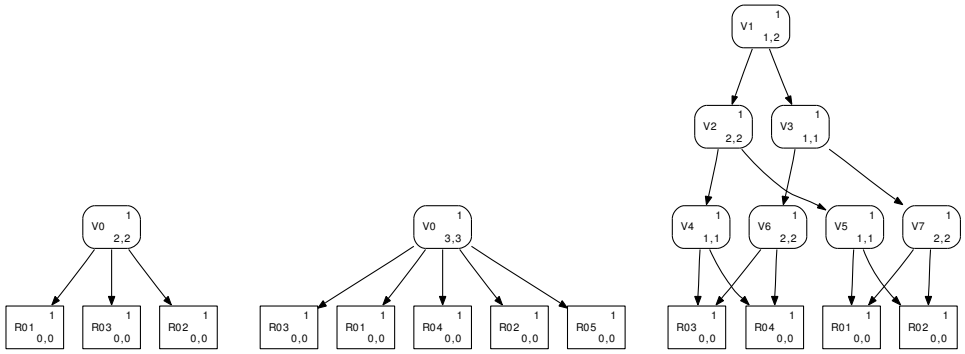
pr_v	v_r	e_r	v_w	e_w
0.1	0.0000	2	0.0000	2
0.2	0.0086	1	0.0000	2
0.3	0.1686	80	0.0287	63
0.4	0.3111	292	0.1160	226
0.5	0.6268	635	0.2822	650
0.6	0.9278	1591	0.7852	1603
0.7	0.9713	1280	0.9060	1415
0.8	0.9980	810	0.9867	1002
0.9	1.0000	328	1.0000	503

(b) Tabelle der Operationsverfügbarkeit in Abhängigkeit der Knotenverfügbarkeit des adaptive dynamic General Structured Voting-Verfahrens für Lese- und Schreiboperationen

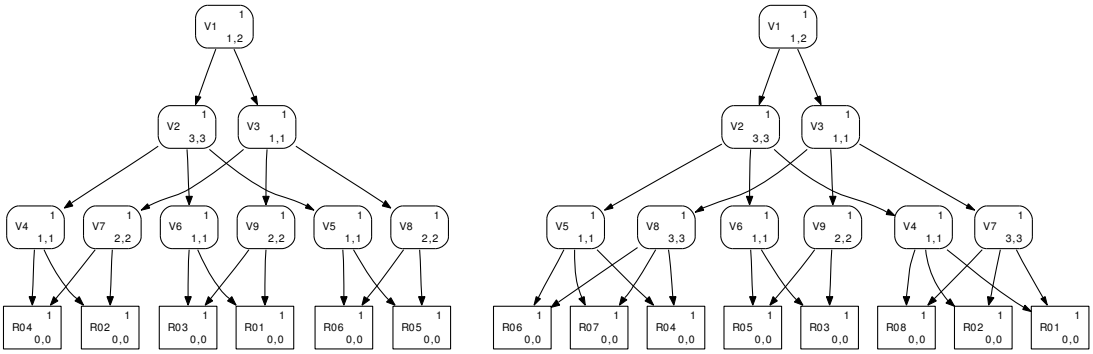
Abbildung 5.11: Messergebnisse des adaptive dynamic General Structured Voting-Verfahrens mit neun Knoten



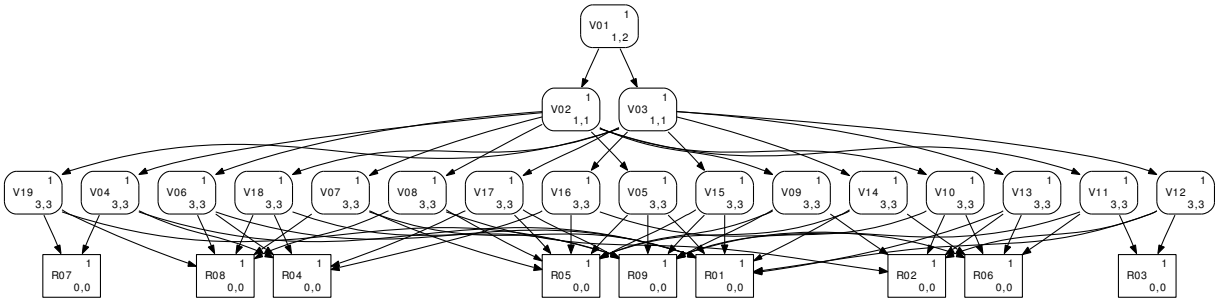
(a) ein Knoten (b) zwei Knoten: Read One, Write All-Verfahren (c) sieben Knoten: Majority Consensus Voting-Verfahren



(d) drei Knoten: Majority Consensus Voting-Verfahren (e) fünf Knoten: Majority Consensus Voting-Verfahren (f) vier Knoten: Grid Protocol-Verfahren



(g) sechs Knoten: Grid Protocol-Verfahren (h) acht Knoten: Grid Protocol-Verfahren



(i) neun Knoten: Triangular Lattice Protocol-Verfahren

Abbildung 5.12: Voting-Strukturen des adaptive dynamic General Structured Voting-Verfahrens für einen (a) bis neun Knoten (i)

5.2.5 direkter Vergleich der Verfahren

Einen direkten Vergleich der Operationsverfügbarkeiten der Verfahren geben Abbildung 5.13, Abbildung 5.14 und Abbildung 5.15. Die Messwerte sind den entsprechenden Tabellen zu entnehmen und hier nicht aufgeführt.

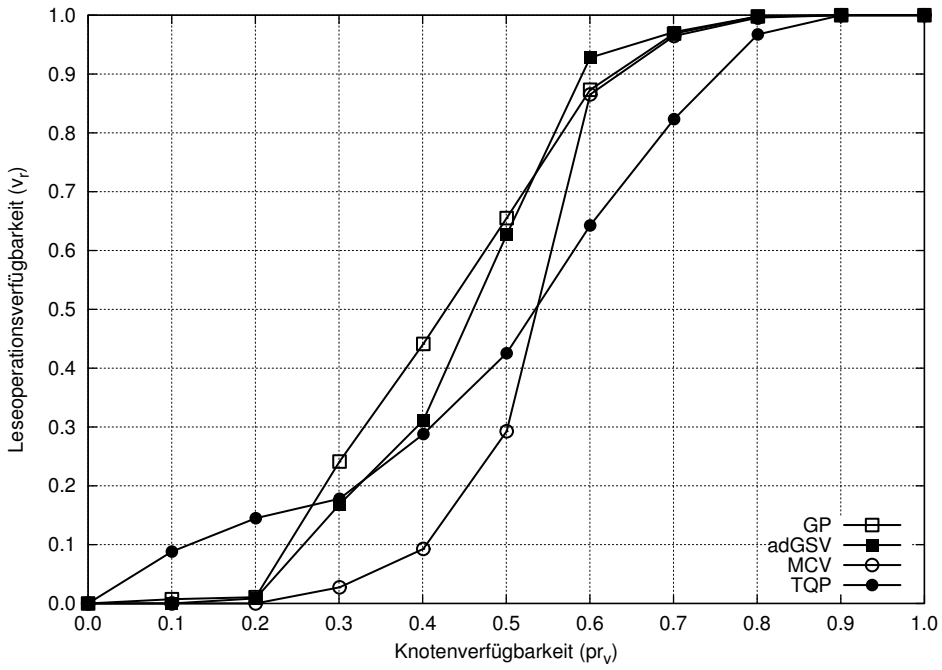


Abbildung 5.13: Vergleich der Leseoperationsverfügbarkeiten in Abhängigkeit der Knotenverfügbarkeit

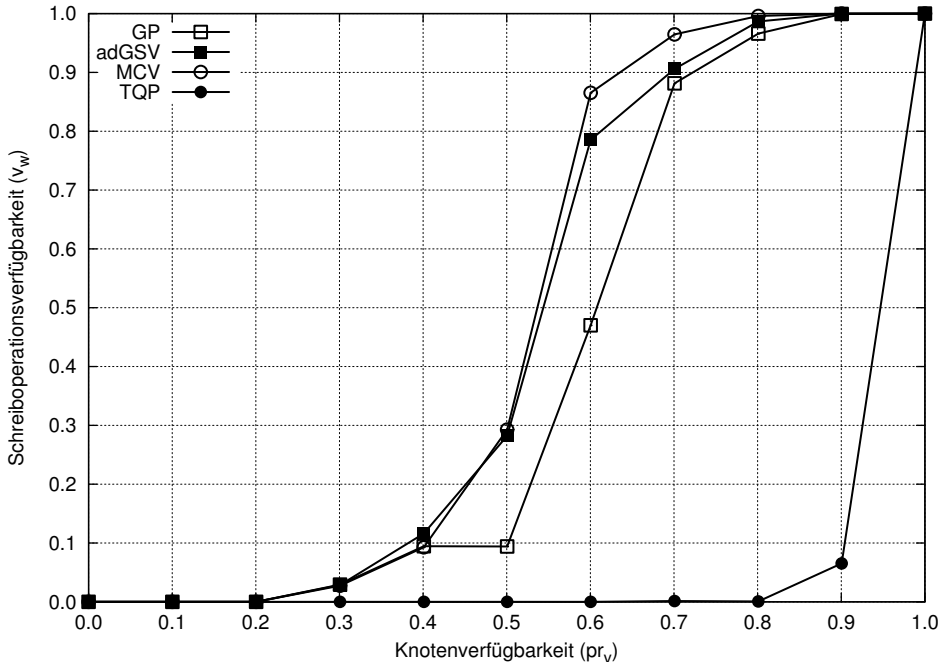


Abbildung 5.14: Vergleich der Schreiboperationsverfügbarkeiten in Abhängigkeit der Knotenverfügbarkeit

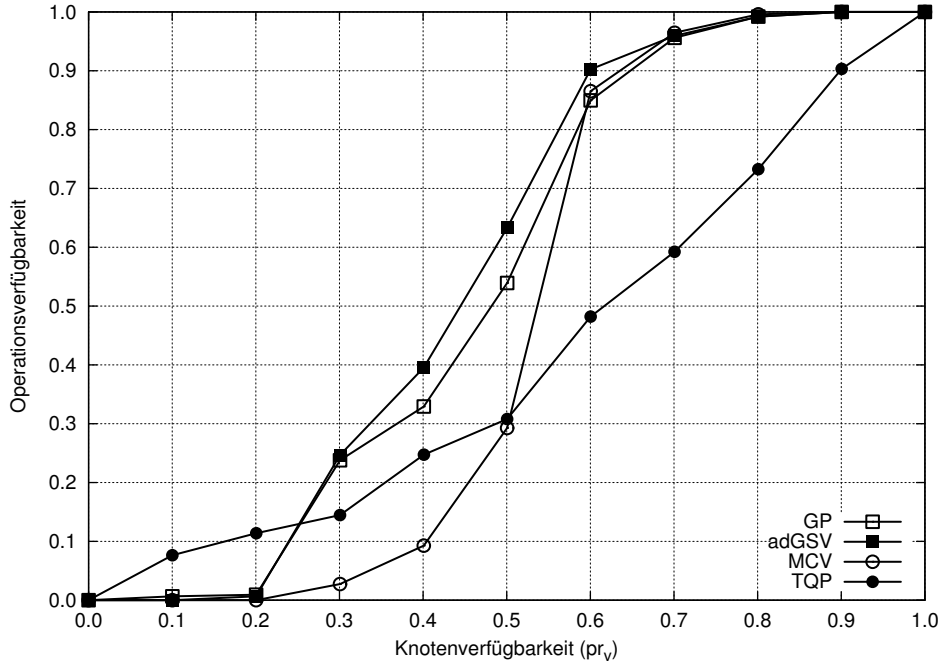


Abbildung 5.15: Vergleich der Operationsverfügbarkeit in Abhängigkeit der Knotenverfügbarkeit bei einem Operationsverhältnis von 80% Leseoperationen und 20% Schreiboperationen

5.3 Beispielszenario und Messergebnisse des Adaptivitätsaspektes

Als Beispielszenario wird ein Unternehmensnetzwerk angenommen, das in Europa, Nordamerika und Australien jeweils ein Subnetzwerk mit fünf Knoten besitzt, wobei die drei Subnetzwerke untereinander verbunden sind. Das Netzwerk besteht also aus fünfzehn Knoten $R_1, \ldots, R15$, die gleichmäßig auf die drei Subnetzwerke C_1, C_2 und C_3 verteilt sind. Das Subnetzwerk C_1 besteht aus der Knotenmenge $\{R1, R2, R3, R10, R11\}$, das Subnetzwerk C_2 besteht aus der Knotenmenge $\{R7, R8, R9, R14, R15\}$ und das Subnetzwerk C_2 besteht aus der Knotenmenge $\{R4, R5, R6, R12, R13\}$. Das eingesetzte Replikationsverfahren soll hohe Lese- und Schreiboperationsverfügbarkeit bei gleichzeitig geringen Kommunikationskosten bieten. Die Kommunikationkosten zweier Knoten innerhalb eines Subnetzwerkes sind ebenso wie für rechnerlokale Kommunikation eins. Für die Kommunikation eines Knotens aus dem Subnetzwerk C_1 (C_2) zu einem Knoten aus C_2 (C_1) wird drei, und von einem Knoten aus dem Subnetzwerk C_1 (C_3) zu einem Knoten aus C_3 (C_1) wird fünf berechnet. Die Kommunikationskosten eines Knotens aus dem Subnetzwerk C_2 (C_3) zu einem Knoten aus C_3 (C_2) betragen vier. Die Kosten einer Operation entsprechen der Summe der Kommunikationskosten zu den jeweiligen Knoten des Quorums. Wenn z.B. der Knoten $R1$ aus dem Subnetzwerk C_1 eine Leseoperation ausführt und das Lesequorum aus den Knoten $R7, R5, R9$ und $R10$ besteht, dann ergeben sich die Gesamtkosten zu $3 + 5 + 3 + 1 = 12$.

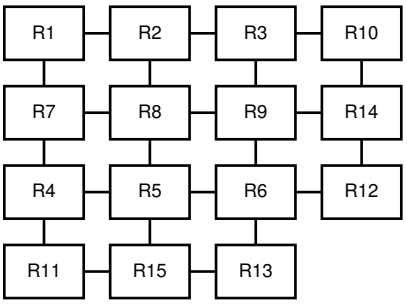


Abbildung 5.16: Knotenanordnung des dGP-Verfahrens mit fünfzehn Knoten

Das dGP-Verfahren bietet eine gute Balance zwischen Operationsverfügbarkeiten und Operationskosten und ermöglicht mit der in Abbildung 5.16 dargestellten Knotenanordnung für fünfzehn Knoten Leseoperationen subnetzwerklokal auszuführen. Schreiboperationen benötigen die Zustimmung eines Knotens aus jeder Spalte (C-Cover) und die Zustimmung von allen Knoten aus einer Spalte (CC-Cover), während Leseoperationen die Zustimmung eines C-Covers oder die Zustimmung eines CC-Covers benötigen. Wenn der Knoten $R1$ aus dem Subnetzwerk C_1 eine Leseoperation ausführt, dann wird das Lesequorum nach Möglichkeit aus den Knoten $R1, R2, R3$ und $R10$ bestehen, da die Gesamtkosten nur $1 + 1 + 1 + 1 = 4$ betragen, denn die Knoten $R1, R2, R3$ und $R10$ sind alle im gleichen Subnetzwerk C_1 und die Kommunikationskosten für subnetzwerklokale sowie für rechnerlokale Kommunikation betragen eins.
Die Simulation des dGP-Verfahrens wurde mit einer homogenen Konfiguration des adGSV-Verfahrens durchgeführt, die für alle Knotenanzahlen das GP-Verfahren benutzt. Der GP-Strukturgenerator ordnet möglichst viele Knoten eines Subnetzwerkes in jeweils einer Zeile des Gitters an. Eine Zeile entspricht einem C-Cover und ermöglicht bei Verwendung dieses C-Covers subnetzwerklokale Leseoperationen durchzuführen. Die dann noch nicht in das Gitter eingefügten Knoten werden der Reihe nach auf noch freie Stellen im Gitter verteilt.

In Abbildung 5.16 bestehen die ersten drei Zeilen jeweils aus Knoten eines Subnetzwerkes und die letzte Zeile ist mit jeweils einem Knoten aus einem Subnetzwerk besetzt.
Die adaptive Konfiguration des adGSV-Verfahrens benutzt ebenfalls das GP-Verfahren und den gleichen GP-Strukturgenerator, allerdings werden von den vorhandenen fünf Knoten in jedem Subnetzwerk nur maximal drei Knoten gleichzeitig benutzt. Die neun aktiven Knoten sind in einem 3×3 Gitter angeordnet, in dem jede Zeile aus den drei aktiven Knoten eines Subnetzwerkes besteht. Die restlichen zwei Knoten des Subnetzwerkes dienen als Ersatzknoten, die ebenso wie aktive Knoten ausfallen können. Wenn aktive Knoten eines Subnetzwerkes ausfallen, dann werden diese Knoten durch Ersatzknoten desselben Subnetzwerkes ersetzt, falls noch Ersatzknoten vorhanden sind. Ein Ersatzknoten wird mittels der `create_replica()`-Funktion in das Replikationsverfahren integriert. Wenn keine Ersatzknoten mehr vorhanden sind, dann nehmen weniger als drei Knoten eines Subnetzwerkes am Verfahren teil und die jeweilige Zeile im Gitter ist dann mit entsprechend weniger Knoten besetzt. Wenn ein ausgefallener Knoten wieder verfügbar ist und weniger als drei Knoten in dessen Subnetzwerk aktiv und in der Voting-Struktur der aktuellen Epoche sind, dann wird dieser Knoten in das Verfahren integriert und die Anzahl aktiver Knoten des Subnetzwerkes erhöht sich um eins. Sind bereits drei Knoten aktiv und in der Voting-Struktur der aktuellen Epoche und ein ausgefallener Knoten wird wieder verfügbar, dann wird der Knoten als Ersatzknoten gekennzeichnet. Wenn ein Ersatzknoten ausfällt, dann wird die Anzahl der verfügbaren Ersatzknoten entsprechend um eins verringert.
Die Operationsverfügbarkeiten und Operationskosten des dGP-Verfahrens mit neun Knoten in einem 3×3 Gitter, das wie für das dGP-Verfahren mit fünfzehn Knoten befüllt wird, werden in den Messergebnissen als Anhaltspunkt angegeben, da das adGSV-Verfahren auch auf der 3×3 Gitteranordnung basiert. Die Messergebnisse sind jedoch nicht direkt vergleichbar, da das adGSV-Verfahren zum einen fünfzehn Knoten verwalten muss und zum anderen in jeder Zeile des Gitters nur Knoten aus einem Subnetzwerk enthalten sein dürfen.

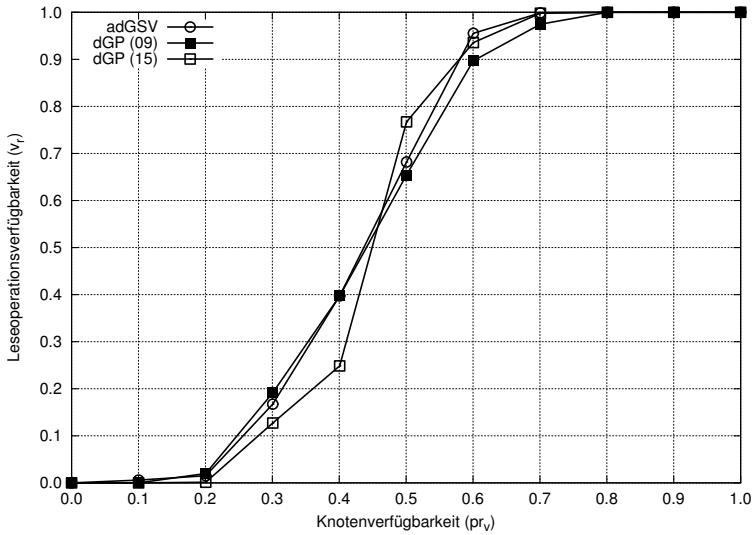


Abbildung 5.17: Graph der Leseoperationsverfügbarkeiten v_r des dGP(15)-Verfahrens, des dGP(09)-Verfahrens und des adGSV-Verfahrens in Abhängigkeit der Knotenverfügbarkeit

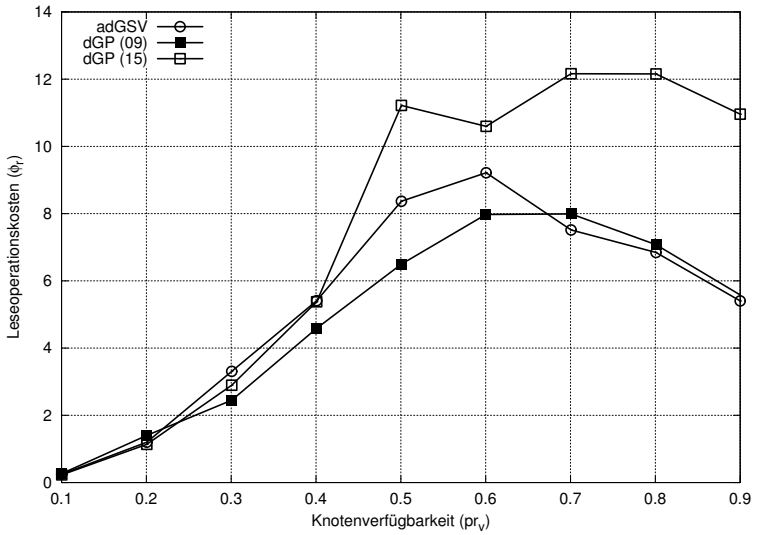


Abbildung 5.18: Graph der durchschnittlichen Leseoperationskosten ϕ_r des dGP(15)-Verfahrens, des dGP(09)-Verfahrens und des adGSV-Verfahrens in Abhängigkeit der Knotenverfügbarkeit

pr_v	adGSV v_r	adGSV ϕ_r	dGP(09) v_r	dGP(09) ϕ_r	dGP(15) v_r	dGP(15) ϕ_r
0.1	0.0060	0.25	0.0000	0.27	0.0000	0.23
0.2	0.0153	1.20	0.0200	1.40	0.0013	1.14
0.3	0.1673	3.31	0.1913	2.45	0.1273	2.89
0.4	0.3972	5.41	0.3979	4.59	0.2485	5.38
0.5	0.6819	8.37	0.6539	6.50	0.7672	11.22
0.6	0.9553	9.22	0.8968	7.97	0.9357	10.60
0.7	0.9987	7.51	0.9746	7.99	0.9980	12.16
0.8	1.0000	6.85	1.0000	7.08	1.0000	12.15
0.9	1.0000	5.40	1.0000	5.57	1.0000	10.96

Abbildung 5.19: Tabelle der Leseoperationsverfügbarkeiten v_r und der durchschnittlichen Leseoperationskosten ϕ_r des dGP(15)-Verfahrens, des dGP(09)-Verfahrens und des adGSV-Verfahrens

In Abbildung 5.17 sind die Leseoperationsverfügbarkeiten v_r des dGP-Verfahrens mit neun (dGP(09)) bzw. fünfzehn Knoten ((dGP(15)) und des adGSV-Verfahrens dargestellt. Die Operationsverfügbarkeit des adGSV-Verfahrens ist für alle Knotenverfügbarkeiten durchweg besser als die des dGP(15)-Verfahrens. Die Operationskosten des adGSV-Verfahrens sind ab einer Knotenverfügbarkeit von $pv_r \geq 0.5$ geringer als die des dGP(15)-Verfahrens, während die Kosten beider Verfahren für kleinere Knotenverfügbarkeiten vergleichbar sind. In Abbildung 5.18 sind die Leseoperationskosten ϕ_r dargestellt. Das dGP(09)-Verfahren ist für Knotenverfügbarkeiten $pr_v \geq 0.3$ mit der Ausnahme von $pr_v = 0.9$ kostengünstiger als das adGSV-Verfahren. Ab einer Knotenverfügbarkeit von $pr_v = 0.5$ ist die Leseoperationsverfügbarkeit des adGSV-Verfahrens größer als die des dGP(09)-Verfahrens. Beides ist durch den Mechanismus des Austauschens ausgefallener Knoten durch Ersatzknoten im adGSV-Verfahren zu erklären. Während das dGP(09)-Verfahren Knotenausfällen zu einer Voting-Struktur mit entsprechend weniger Knoten wechselt, werden ausgefallenene Kno-

ten im adGSV-Verfahren durch Ersatzknoten, falls vorhanden, ersetzt. Die Anzahl aktiver Knoten ist im adGSV-Verfahren generell höher, was zu höheren Operationskosten führt, da mehr Knoten in einem Quorum enthalten sind und kontaktiert werden müssen. Gleichzeitig wird die Anzahl möglicher Lesequoren erhöht, was zu einer höheren Operationsverfügbarkeit des adGSV-Verfahrens ab $pr_v = 0.5$ führt. Für $pr_v \leq 0.4$ treten zu viele Knotenausfälle auf, die vom dGP(09)-Verfahren besser kompensiert werden können, weil die Zeilen des Gitters nicht ausschließlich mit Knoten aus einem Subnetzwerk belegt sein müssen. In Abbildung 5.19 sind die gemessenenen Werte tabellarisch dargestellt.

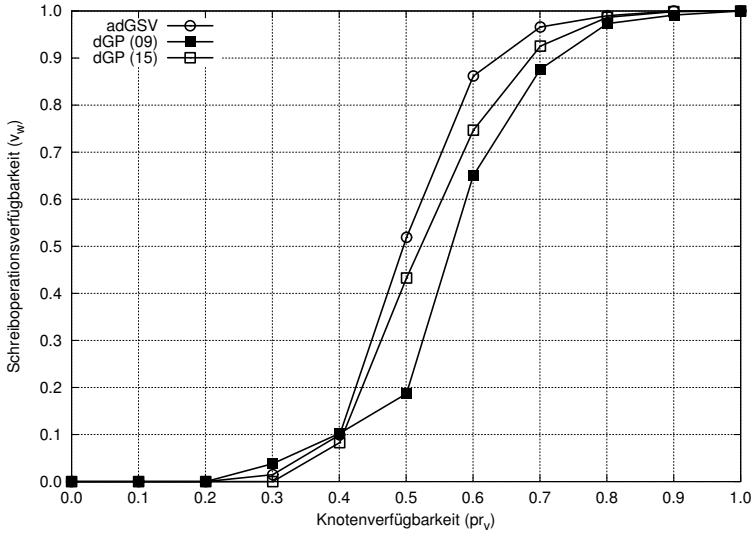


Abbildung 5.20: Graph der Schreiboperationsverfügbarkeiten v_w des dGP(15)-Verfahrens, des dGP(09)-Verfahrens und des adGSV-Verfahrens in Abhängigkeit der Knotenverfügbarkeit

In Abbildung 5.20 sind die Schreiboperationsverfügbarkeiten v_w des dGP(09)-Verfahrens, des dGP(15)-Verfahrens und des adGSV-Verfahrens dargestellt. Die Schreiboperationsverfügbarkeit des adGSV-Verfahrens ist für alle Knotenverfügbarkeiten durchweg besser als die des dGP(15)-Verfahrens. Für Knotenverfügbarkeiten $pr_v \leq 0.4$ ist die Operationsverfügbarkeit des dGP(09)-Verfahrens besser als die des adGSV-Verfahrens, was wiederum durch den Mechanismus des Austauschens ausgefallener Knoten durch Ersatzknoten im adGSV-Verfahren zu erklären ist. In Abbildung 5.21 sind die Schreiboperationskosten ϕ_w der drei Verfahren dargestellt. Sowohl für das dGP(09)-Verfahren als auch für das dGP(15)-Verfahren steigen die Operationskosten stetig an. Die Operationskosten des adGSV-Verfahrens stabilisieren sich hingegen ab einer Knotenverfügbarkeit von $pr_v \geq 0.6$ auf $\backsim 19.1175$. Ab dieser Knotenverfügbarkeit sind genug verfügbare Knoten pro Subnetzwerk vorhanden, um die Operationskosten relativ konstant zu halten. In Abbildung 5.22 sind die gemessenenen Werte tabellarisch dargestellt.

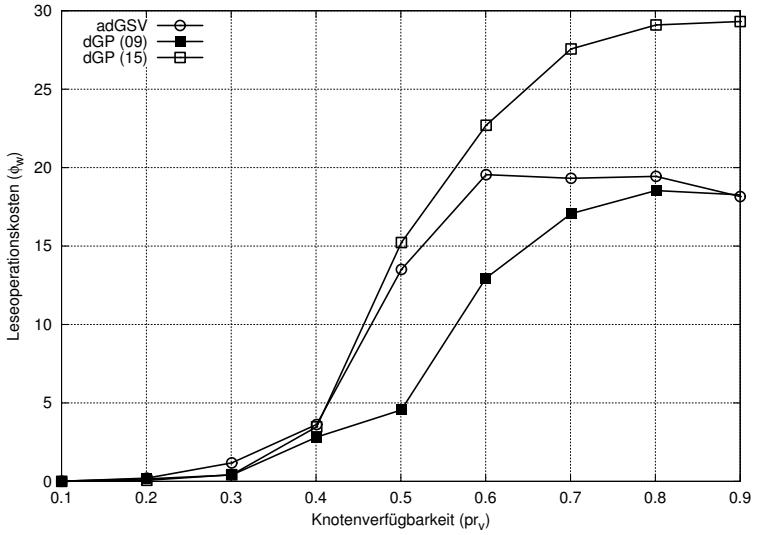
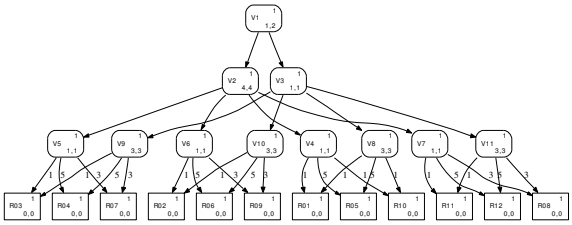


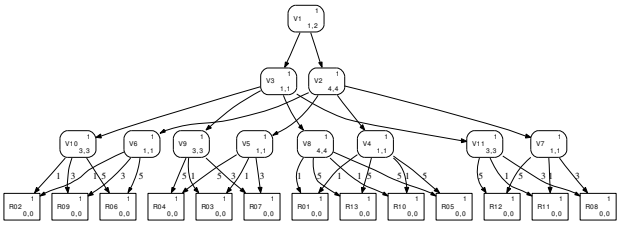
Abbildung 5.21: Graph der durchschnittlichen Schreiboperationskosten ϕ_w des dGP(15)-Verfahrens, des dGP(09)-Verfahrens und des adGSV-Verfahrens in Abhängigkeit der Knotenverfügbarkeit

pr_v	adGSV v_w	adGSV ϕ_w	dGP(09) v_w	dGP(09) ϕ_w	dGP(15) v_w	dGP(15) ϕ_w
0.1	0.0000	0.01	0.0000	0.01	0.0000	0.00
0.2	0.0000	0.21	0.0000	0.16	0.0000	0.07
0.3	0.0147	1.17	0.0380	0.41	0.0000	0.42
0.4	0.0993	3.62	0.1027	2.83	0.0827	3.48
0.5	0.5190	13.51	0.1868	4.55	0.4326	15.23
0.6	0.8620	19.55	0.6502	12.96	0.7467	22.70
0.7	0.9660	19.32	0.8764	17.07	0.9253	27.56
0.8	0.9900	19.44	0.9733	18.54	0.9867	29.09
0.9	1.0000	18.16	0.9913	18.26	0.9987	29.31

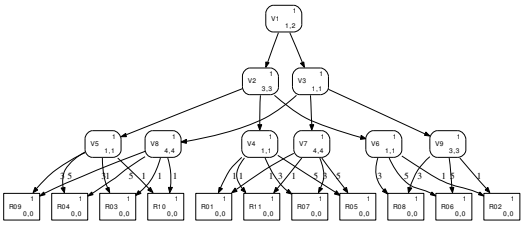
Abbildung 5.22: Tabelle der Schreiboperationsverfügbarkeiten v_w und durchschnittlichen Schreiboperationskosten ϕ_w des dGP(15)-Verfahrens, des dGP(09)-Verfahrens und des adGSV-Verfahrens



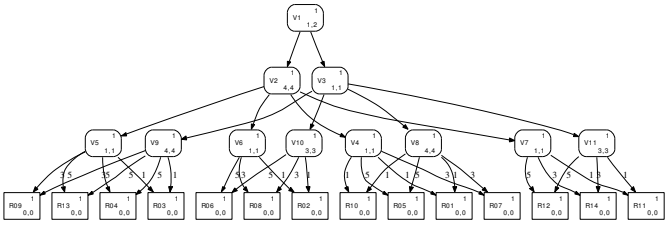
(a) zwölf Knoten



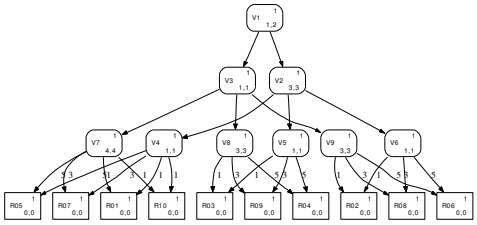
(b) dreizehn Knoten



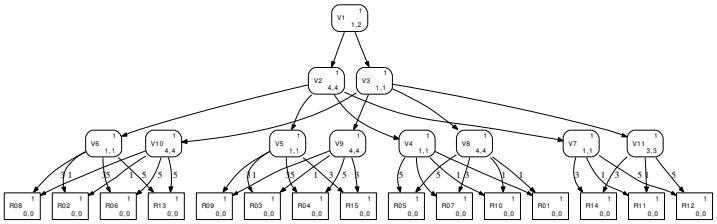
(c) elf Knoten



(d) vierzehn Knoten



(e) zehn Knoten



(f) fünfzehn Knoten

Abbildung 5.23: Voting-Strukturen des Grid Protocol-Verfahrens für zehn (a) bis fünfzehn Knoten (f)

5.4 Zusammenfassung

Die in Abschnitt 5.2 vorgestellten Messergebnisse stellen die spezifischen Vor- und Nachteile der jeweiligen Verfahren heraus. Die Nachteile homogener Verfahren für bestimmte Knotenanzahlen können durch die gezielte Verwendung einer besseren Strategie für die jeweilige Knotenanzahl kompensiert werden, wie in Abschnitt 5.2.4 durch die Verwendung einer inhomogenen Konfiguration des adGSV-Verfahrens exemplarisch gezeigt wurde.
Der adaptive Aspekt des adGSV-Verfahrens, der in Abschnitt 5.3 anhand eines Beispielszenarios mit einer homogenen Konfiguration des adGSV-Verfahrens im Vergleich zum dGSV-Verfahren dargestellt wurde, ermöglicht geringere Operationskosten bei gleichzeitig höherer Operationsverfügbarkeit.
Im Rahmen dieser Arbeit konnte eine Vergleichsmessung verschiedener Commit-Protokolle nicht durchgeführt werden und muss Gegenstand weiterer Arbeiten bleiben. Dieser Aspekt ist jedoch insofern interessant, als dass die Operationsverfügbarkeit auch durch das verwendete Commit-Protokoll beeinflusst werden kann. Wenn z.B. das 2PC-Protokoll benutzt wird und der Koordinator ausfällt, dann sind bis zur Reparatur des Koordinators keine Operationen mehr durchführbar.

6 Zusammenfassung und Ausblick

Replikationsverfahren dienen zur Erhöhung der Verfügbarkeit von kritischen Daten.
Statische Replikationsverfahren basieren auf einer fixen Anzahl Replikate, die zu Beginn des Verfahrens festgelegt wird. Wenn mehr Rechner ausgefallen sind als die vom Replikationsverfahren verwendete Strategie zur Durchführung von Lese- oder Schreiboperationen tolerieren kann, dann ist bei Rechnerausfällen der Zugriff auf die Daten nicht mehr möglich. Die Verfügbarkeit aller statischen Verfahren ist durch die eine symetrische Abhängigkeit des Verfügbarkeitsgrades von Lese- und Schreiboperationen beschränkt. Dynamische Verfahren heben diese Beschränkung durch die Fähigkeit der Adaption an die jeweilige Anzahl Knoten auf und erhöhen somit den Verfügbarkeitsgrad. Sowohl statischen als auch dynamischen Verfahren ist die Beschränkung auf eine vorher festgelegte maximale Anzahl verwaltbarer Knoten gemein. Dynamische Verfahren entfalten ihre Dynamik nur im Bereich von einem bis zur maximalen Anzahl Knoten, wohingegen statische Verfahren eine solche Dynamik nicht besitzen und nur mit der maximalen Anzahl Knoten arbeiten. Einige dynamische Verfahren sind durch spezifische Zusatsprotokolle zwar in der Lage zur Laufzeit neue Knoten in das Replikationsverfahren zu integrieren oder daraus zu entfernen, jedoch ist ein solches Protokoll für jedes einzelne Verfahren zu entwickeln und keineswegs allgemeingültig.
Zahlreiche statische und dynamische Verfahren mit spezifischen Vor- und Nachteilen sind bisher entwickelt worden. Mangels eines Verfahrens, das für alle Anwendungsfälle gleichermaßen gut geeignet ist, muss für jedes Anwendungsszenario das passende Verfahren gesucht und implementiert werden. Ein Wechsel des Verfahrens aufgrund geänderter Anforderungen ist nur durch die manuelle Anpassung auf ein neues Verfahren möglich, was kostenintensiv und fehleranfällig ist. Generalisierende Verfahren, wie z.B. das General Structured Voting-Verfahren, bei denen der Austausch einer Datenstruktur zur Adaption an die geänderten Anforderungen genügt, sind ein konsequenter Schritt in die richtige Richtung.
Alle bisherigen dynamischen Strategien sind homogen in dem Sinne, dass die jeweilige Strategie für alle Knotenanzahlen benutzt wird. Wenn z.B. das Grid Protocol-Verfahren für neun Knoten benutzt wird, dann wird es bei Knotenausfällen auch für alle Knotenanzahlennzahlen kleiner als neun benutzt.
Das in dieser Arbeit vorgestellte und prototypisch implementierte adaptive dynamic General Structured Voting-Verfahren, das auf auf dem dGSV-Verfahren aufbaut, erlaubt die Benutzung einer inhomogenen Strategie bei der für verschiedene Knotenanzahlen verschiedene Strategien verwendet werden. Die Schwächen homogener Verfahren für bestimmte Knotenanzahlen lassen sich zugunsten höherer Verfügbarkeit gezielt beheben, wie in Abschnitt 5.2.4 gezeigt wurde. Auf diese Weise lassen sich Verfahren mit höherer Verfügbarkeit im Vergleich zu bisherigen Verfahren entwickeln. Das Konzept der Strukturgeneratoren erlaubt zudem z.B. durch einen dedizierten Überwachungs- und Verwaltungsknoten ohne manuelles Eingreifen Knoten dem Replikationsverfahren hinzuzufügen oder vorhandene Knoten daraus zu entfernen.
Die automatische Erzeugung von Strukturgeneratoren und deren Attributierung, weitergehende Analysen von inhomogenen Strategien sowie die Realisierung der vorgestellten Erweiterungen bleibt Gegenstand weiterer Arbeiten.

Literaturverzeichnis

[AA90] Agrawal, Divyakant ; Abbadi, Amr E.: The Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data. In: *Proceedings of the 16th Very Large Data Bases Conference (VLDB'90)*, 1990, S. 243–254

[AAB96] Abdelsalam, Helal A. ; Abdelsalam, Heddaya A. ; Bharat, Bhargava B.: *Replication Techniques in Distributed Sytems.* Kluwer Academic Publishers, 1996

[AAC90] Ammar, Mostafa H. ; Ahamad, Mustaque ; Cheung, Shun Y.: The Grid Protocol: A High Performance Scheme for Maintaining Replicated Data. In: *Proceedings of the 6th International Conference on Data Engineering (ICDE'90)*, 1990, S. 438–445

[AAL94] Abbadi, A. E. ; Agrawal, D. ; Liu, M. L.: An Efficient Implementation of the Quorum Consensus Protocol / Department of Computer Science, University of California. 1994 (TR–1994–13). – Forschungsbericht

[AE92] Agrawal, D. ; El Abbadi, A.: The Generalized Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data. In: *ACM Transactions on Database Systems* 17 (1992), Nr. 4, S. 689–717

[BHG87] Bernstein, Philip A. ; Hadzilacos, Vassos ; Goodman, Nathan: *Concurrency Control and Recovery in Database Systems.* Addison Wesley, 1987

[Boc00] Bock, Michael: *Erweiterung eines CORBA-basierten Rahmenwerks zur Datenreplikation um dynamische Replikationsstrategien*, Technische Universität Darmstadt, Diplomarbeit, 2000

[Dav89] Davcev, Danco: A Dynamic Voting Scheme in Distributed Systems. In: *IEEE Transactions on Software Engineering* 15 (1989), Nr. 1, S. 93–98

[DGMS85] Davidson, Susan B. ; Garcia-Molina, Hector ; Skeen, Dale: Consistency in Partitioned Networks. In: *ACM Computing Surveys* 17 (1985), Nr. 3, S. 341–370

[FKT91] Freisleben, Bernd ; Koch, Hans-Henning ; Theel, Oliver: Designing Multi-Level Quorum Schemes for Highly Replicated Data. In: *Proceedings of the Pacific Rim International Symposium on Fault Tolerant Systems*, 1991, S. 154–159

[GHJV94] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley, 1994

[Gif79] Gifford, D. K.: Weighted Voting for Replicated Data. In: *Proceedings of the 7th Symposium on Operating Systems Principles (SOSP'79)* Bd. 13, 1979, S. 150–161

[GMB85] Garcia-Molina, Hector ; Barbara, Daniel: How to Assign Votes in a Distributed System. In: *Journal of the ACM* 32 (1985), Nr. 4, S. 841–860

[Her90] Herlihy, Maurice: Apologizing Versus Asking Permission: Optimistic Concurrency Control for Abstract Data Types. In: *ACM Transactions on Database Systems* 15 (1990), Nr. 1, S. 96–124

[JM87a] Jajodia, Sushil ; Mutchler, David: Dynamic Voting. In: *Proceedings of ACM SIGMOD annual conference*, 1987, S. 227–238

[JM87b] Jajodia, Sushil ; Mutchler, David: Enhancements to the Voting Algorithm. In: *Proceedings of the 13th International Conference on Very Large Data Bases (VLDB'87)*, 399–406

[JM90] Jajodia, Sushil ; Mutchler, David: Dynamic Voting Algorithms for Maintaining the Consistency of a Replicated Database. In: *ACM Transactions on Database Systems* 15 (1990), Nr. 2, S. 230–280

[KA01] Kemme, B. ; Alonso, G.: How to Select a Replication Protocol According to Scalability, Availability and Communication Overhead. In: *Proceedings of the IEEE International Conference on Reliable Distributed Systems (SRDS'01)*, 2001

[Koc94] Koch, Hans-Henning: *Entwurf und Bewertung von Replikationsverfahren*, Fachbereich Informatik, Technische Hochschule Darmstadt, Diss., 1994

[LMG95] Long, Darrell D. E. ; Muir, Andrew ; Golding, Richard A.: A Longitudinal Survey of Internet Host Reliability. In: *Proceedings of the Symposium on Reliable Distributed Systems*, 1995, S. 2–9

[LR93] Lazowska, Edward D. ; Rabinovich, Michael: Asynchronous Epoch Management in Replicated Databases. In: *Proceedings of the International Workshop on Distributed Algorithms (WDAG'93)*, 1993, S. 115–128

[MMM04] Mamun, Quazi Ehsanul K. ; Masum, Salahuddin M. ; Mustafa, Mohammad Abdur R.: Modified Bully Algorithm for Electing Coordinator in Distributed Systems. In: *WSEAS Transactions on Computers 2004* Bd. 3

[MN92] Mizuno, Masaaki ; Neilsen, Mitchell L.: *Measures of Node Importance for Quorum Structures.* 1992

[PT97] Pagnia, Henning ; Theel, Oliver: Improving Replication Protocols through Priorities / Department of Computer Science, Institute for System Architecture, University of Darmstadt. 1997 (THD–BS–1997–03). – Forschungsbericht. – 342–343 S

[RL92] Rabinovich, Michael ; Lazowska, Edward: Improving Fault Tolerance and Supporting Partial Writes in Structured Coterie Protocols for Replicated Objects. In: *Proceedings of the ACM SIGMOD Conference*, 1992, S. 226–235

[Sav97] Savikko, Vespe: Design Patterns in Python. In: *Proceedings of the 6th International Python Conference*, 1997

[Ske81] Skeen, Dale: Non-Blocking Commit Protocols. In: *Proceedings of the ACM SIGMOD Conference*, 1981

[SS83] Schlichting, R. D. ; Schneider, F. B.: Fail-Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems. In: *ACM Transactions on Computer Systems* (1983), S. 222–238

[SS03] Saito, Yasushi ; Shapiro, Marc: Optimistic Replication / Microsoft Research (MSR). 2003 (MSR-TR-2003-60). – Forschungsbericht

[Tan92] Tanenbaum, Andrew. S.: *Modern Operating Systems*. Prentice-Hall, 1992

[The93a] Theel, Oliver: General Structured Voting: A Flexible Framework for Modelling Cooperations. In: *Proceedings of the 13th International Conference on Distributed Computing Systems (ICDCS'93)*, 1993, S. 227–236

[The93b] Theel, Oliver: *Ein vereinheitlichendes Konzept zur Konstruktion hochverfügbarer Dienste*, Fachbereich Informatik, Technische Hochschule Darmstadt, Diss., 1993

[TP98] Theel, Oliver ; Pagnia, Henning: Optimal Replica Control Protocols Exhibit Symmetric Operation Availabilities. In: *Proceedings of the 28th International Symposium on Fault-Tolerant Computing (FTCS-28), München, Germany*, 1998, S. 252–261

[TPK95] Theel, Oliver ; Pagnia-Koch, Henning: General Design of Grid-Based Data Replication Schemes Using Graphs and a Few Rules. In: *Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS'95)*, 1995, S. 395–403

[TS98] Theel, Oliver ; Strauss, Thomas: Automatic Generation of Dynamic Coterie-based Replication Schemes / Department of Computer Science, Institute for System Architecture, University of Darmstadt. 1998 (TUD–BS–1998–01). – Forschungsbericht

[TS99] Theel, Oliver ; Strauss, Thomas: An Excursion to the Zoo of Dynamic Coterie-based Replication Schemes. In: *Proceedings of the 28th International Conference on Parallel Processing (ICPP'99)*, 1999, S. 344–352

Glossar

Kürzel	Beschreibung
1SR	1-Kopien-Serialisierbarkeit
2PC	2 Phase Commit
3PC	3 Phase Commit
adGSV	adaptive dynamic General Structured Voting
CORBA	Common Object Request Broker Architecture
CVS	Current Voting-Struktur
dGP	dynamic Grid Protocol
dGSV	dynamic General Structured Voting
DV	Dynamic Voting
GP	Grid Protocol
GSV	General Structured Voting
MCV	Majority Consensus Voting
MP	Majority Partition
MTTF	mean time to failure
MTTR	mean time to repair
MVS	Master Voting-Struktur
PCROWA	Primary Copy ROWA
QC	Quorum Consensus
RAWO	Read All, Write One
ROWA	Read One, Write All
ROWAA	Read One, Write All Available
TLP	Triangular Lattice Protocol
TQP	Tree Quorum Protocol
usc	update site cardinality
WV	Weighted Voting