Grüne Datenbanken: Analyse und Optimierung des Energieverbrauches von Datenbanken für sehr große Datenbestände

Dissertation

zur Erlangung des Doktorgrades

der Naturwissenschaften

vorgelegt beim Fachbereich 12 Informatik

der Johann Wolfgang Goethe -Universität

in Frankfurt am Main

von

Raik Niemann

aus Rostock

Frankfurt am Main 2016

(D 30)

vom Fachbereich 12 Informatik der

der Johann Wolfgang Goethe -Universität als Dissertation angenommen.

Dekan: Prof. Dr. Uwe Brinkschulte

Gutachter: Prof. Dr.-Ing. Roberto Zicari, Prof. Dr. Richard Göbel

Datum der Disputation: 29. Oktober 2016

Kurzzusammenfassung und Danksagungen

Die zunehmende Verbreitung des Internets als universelles Netzwerk zum Transport von Daten aller Art hat in den letzten zwei Dekaden dazu geführt, dass die anfallenden Datenmengen von traditionellen Datenbanksystemen kaum mehr effektiv zu verarbeiten sind. Das liegt zum einen darin, dass ein immer größerer Teil der Erdbevölkerung Zugang zum Internet hat, zum Beispiel via Internet-fähigen Smartphones, und dessen Dienste nutzen möchte. Zudem tragen immer höhere verfügbare Bandbreiten für den Internetzugang dazu bei, dass die weltweit erzeugten Informationen mittlerweile exponentiell steigen.

Das führte zur Entwicklung und Implementierung von Technologien, um diese immensen Datenmengen wirksam verarbeiten zu können. Diese Technologien können unter dem Sammelbegriff *Big Data* zusammengefasst werden und beschreiben dabei Verfahren, um strukturierte und unstrukturierte Informationen im Tera- und Exabyte-Bereich sogar in Echtzeit verarbeiten zu können. Als Basis dienen dabei Datenbanksysteme, da sie ein bewährtes und praktisches Mittel sind, um Informationen zu strukturieren, zu organisieren, zu manipulieren und effektiv abrufen zu können. Wie bereits erwähnt, hat sich herausgestellt, dass traditionelle Datenbanksysteme, die auf dem relationalen Datenmodell basieren, nun mit Datenmengen konfrontiert sind, mit denen sie nicht sehr gut hinsichtlich der Performance und dem Energieverbrauch skalieren. Dieser Umstand führte zu der Entwicklung von spezialisierten Datenbanksystemen, die andere Daten- und Speichermodelle implementieren und für diese eine deutlich höhere Performance bieten.

Zusätzlich erfordern Datenbanksysteme im Umfeld von *Big Data* wesentlich größere Investitionen in die Anzahl von Servern, was dazu geführt hat, dass immer mehr große und sehr große Datenverarbeitungszentren entstanden sind. In der Zwischenzeit sind die Aufwendungen für Energie zum Betrieb und Kühlen dieser Zentren ein signifikanter Kostenfaktor geworden. Dementsprechend sind bereits Anstrengungen unternommen worden, das Themenfeld Energieeffizienz (die Relation zwischen Performance und Energieverbrauch) von Datenbanksystemen eingehender zu untersuchen.

Mittlerweile sind über 150 Datenbanksysteme bekannt, die ihre eigenen Stärken und Schwächen im Bezug auf Performance, Energieverbrauch und schlussendlich Energieeffizienz haben. Die Endanwender von Datenbanksystemen sehen sich nun in der schwierigen Situation, für einen gegebenen Anwendungsfall das geeigneteste Datenbanksystem in Hinblick auf die genannten Faktoren zu ermitteln. Der Grund dafür ist, dass kaum objektive und unabhängige Vergleichszahlen zur Entscheidungsfindung existieren und dass die Ermittlung von Vergleichszahlen zumeist über die Ausführung von Benchmarks auf verschiedensten technischen Plattformen geschieht. Es ist offensichtlich, dass die mehrfache Ausführung eines Benchmarks mit unterschiedlichsten Parametern (unter anderem die Datenmenge, andere Kombinationen aus technischen Komponenten, Betriebssystem) große Investitionen in Zeit und Technik erfordern, um möglichst breit gefächerte Vergleichszahlen zu erhalten.

Eine Möglichkeit ist es, die Ausführung eines Benchmarks zu simulieren anstatt ihn real zu absolvieren, um die Investitionen in Technik und vor allem Zeit zu minimieren. Diese Simulationen haben auch den Vorteil, dass zum Beispiel die Entwickler von Datenbanksystemen die Auswirkungen auf Performance und Energieeffizienz bei der Änderungen an der Architektur simulieren können anstatt sie durch langwierige Regressionstests evaluieren zu müssen. Damit solche Simulationen eine praktische Relevanz erlangen können, muss natürlich die Differenz zwischen den simulierten und den real gewonnenen Vergleichsmetriken möglichst klein sein. Zudem muss eine geeignete Simulation eine möglichst große Anzahl an Datenbanksystemen und technischen Komponenten nachstellen können.

Die vorliegende Dissertation zeigt, dass eine solche Simulation realistisch ist. Dafür wurde in einem ersten Schritt die Einflussaktoren auf Performance, Energieverbrauch und Energieeffizienz eines Datenbanksystems ermittelt und deren Wirkung anhand von experimentellen Ergebnissen bestimmt. Zusätzlich wurden auch geeignete Metriken und generelle Eigenschaften von Datenbanksystemen und von Benchmarks evaluiert. In einem zweiten Schritt wurde dann ein geeignetes Simulationsmodell erarbeitet und sukzessiv weiterentwickelt. Bei jedem Entwicklungsschritt wurden dann reale Experimente in Form von Benchmarkausführungen für verschiedenste Datenbanksysteme und technische Plattformen durchgeführt. Diese Experimente wurden mittels des Simulationsmodell nachvollzogen, um die Differenz zwischen realen und simulierten Benchmarkergebnissen zu berechnen. Die Ergebnisse des letzten Entwicklungsschrittes zeigen, dass diese Differenz unter acht Prozent liegt. Die vorliegende Dissertation zeigt auch, dass das Simulationsmodell nicht nur dazu geeignet ist, anerkannte Benchmarks zu simulieren, sondern sich im allgemeinen auch dafür eignet, ein Datenbanksystem und die technische Plattform, auf der es ausgeführt wird, generell zu simulieren. Das ermöglicht auch die Simulation anderer Anwendungsfälle, zum Beispiel Regressionstests.

Es sei all jenen Menschen gedankt, dich mich bei der Ausarbeitung dieser Dissertation in allen erdenklichen Formen unterstützt haben. Dazu gehören insbesondere das *BigData Lab* der *Goethe-Universität Frankfurt/Main* und das *Institut für Informationssysteme* der *Hochschule Hof*, deren Leiter Herr Prof. Dr.-Ing. *Roberto Zicari* und Herr Prof. Dr. *Richard Göbel* mich in allen Belangen als Mentoren gefördert haben. Zudem sei auch allen anderen Mitarbeiter der beiden Institute gedankt, die sich im Laufe der Zusammenarbeit als äußerst kompetente und hilfsbereite Kollegen erwiesen haben. Schlussendlich möchte ich mich auch bei Freunden und besonders bei meiner Familie bedanken, dass sie nie das Vertrauen in mich und meine Ambitionen verloren und ausreichend Geduld und Hilfsbereitschaft entgegengebracht haben.

Frankfurt am Main, d. 16. März 2017 Raik Niemann

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Definitionsverzeichnis

Verzeichnis der Experimente

1 Einleitung und Motivation

Es ist ein allseits bekannter Fakt, dass der Energieverbrauch weltweit in den letzten Jahren immer mehr exponentiellen Charakter angenommen hat. Die Gründe dafür sind vielfältig: globales Bevölkerungswachstum, der Wunsch nach Wohlstand und die schrittweise Transformation in eine Informationsgesellschaft [Sta09].

All diese Gründe trugen dazu bei, das der Informations- und Telekommunikationssektor (IuT) mittlerweile über 4,7 Prozent der weltweit generierten elektrischen Energie verbraucht. Nach *Gelenbe und Caseau* sind das circa 920 TWh an elektrischer Arbeit (1 TWh entspricht 10^{12} Watt pro Stunde) [GC15]. Ingesamt wurden ungefähr 19.000 TWh an elektrischer Arbeit verbraucht, was einem Anteil von 15 Prozent der weltweit erzeugten Energie entspricht. Zudem muss bemerkt werden, dass die Studie von *Gelenbe und Caseau* die Hersteller von IuT-Geräten, zum Beispiel Mobiltelefone, Netzwerkrouter und ähnliches, nicht einbezogen haben. Deren Produktions- und Entsorgungskosten sind bezogen auf den Energieverbrauch also nicht betrachtet worden, da entsprechende Statistiken fehlen. Somit kann angenommen werden, dass der Energieverbrauch des IuT-Sektors noch höher ausfällt.

Zudem ist auch ein Effekt zu erkennen, der darauf schließen lässt, dass das Wachstum des Energieverbrauchs des IuK-Sektors auch in den kommenden Jahren ansteigt, da sich das Wachstum selbst verstärkt. Durch das weltweite Bevölkerungswachstum und steigenden Wohlstand ist ein immer größerer Teil der Menschheit in der Lage, die Dienstleistungen des IuK-Sektor zu nutzen, zum Beispiel Telefonkommunikation und die Teilnahme am Internet. Besonders letztgenanntes eröffnet eine schiere Menge an neuen Möglichkeiten. Dabei reicht die Bandbreite von (a)synchroner Kommunikation (Chat, E-Mail), das Recherchieren und Konsumieren von Informationen (*World Wide Web*, Nachrichtenkanäle) bis zum Teilen und Anschauen von Fotos und Videos. Diese Möglichkeiten trugen dazu bei, dass immer mehr Bandbreite in den Übertragungsnetzwerken gefordert und auch ausgebaut wird. Durch den Wunsch, die genannten Dienste auch mobil zu nutzen, werden auch Mobiltelefone immer leistungsfähiger. Auch im Mobilfunksektor werden dadurch die Übertragungswege stetig ausgebaut. Insgesamt führt dies wiederum dazu, dass durch die höheren Bandbreiten die Grundlagen geschaffen werden, um noch mehr Dienste anbieten zu können, die aufgrund der vormals geringeren Übertragungsbandbreite undenkbar gewesen sind. Das wiederum fördert den Verkauf von leistungsfähigeren Endgeräten, der Entwicklung neuer Übertragungstechniken und der Schaffung potenterer Übertragungskanälen. Diese Wechselwirkungen zeigen, dass sich das oben erwähnte Wachstum verstärkt.

All dies resultiert letztendlich darin, das die Menge an erzeugten und weltweit verteilten digitalen Informationen, also strukturierte und unstrukturierte Daten, mittlerweile im Zettabereich (1 ZByte entspricht 10^{21} Bytes) angelangt ist [RRB+15]. Die größte Last bei der Verarbeitung, Speicherung und Bereitstellung dieser Datenströme tragen nach einer Studie von *Roberts et al.*

die großen und sehr großen Datenverarbeitungszentren der namhaften IT-Firmen, zum Beispiel *Google*, *Microsoft*, *IBM*, *Facebook* und *Amazon* [RRB+15]. Sie machen zwar nur einen kleinen Anteil an den weltweit rund 150.000 existierenden Datenverarbeitungszentren aus, aber verbrauchen für ihren Betrieb in Relation dazu einen großen Teil an elektrischer Energie. Verlässliche Angaben zu diesem Energieverbrauch sind bisher von offizieller Stelle nicht erhoben worden. Es ist aber nach Recherchen der *New York Times* bekannt, dass alleine die Datenverarbeitungszentren von *Google* 2011 rund 260 MWh an elektrischer Arbeit verbraucht haben [Gla11]. Eine andere Studie aus dem Jahr 2014 zeigt, dass die US-amerikanischen Datenverarbeitungszentren 2013 rund 91 Milliarden kWh an elektrischer Arbeit verbraucht haben, was der Energieerzeugung von 34 Kohlekraftwerken mit je 500 MW elektrischer Leistung entspricht [WD14]. Für das Jahr 2020 geht die Studie von einem elektrischen Energieverbrauch dieser Datenverarbeitungszentren von 140 Milliarden kWh aus. Nach *Dorenburg* verbrauchten 2011 die Datenverarbeitungszentren in Deutschland alleine 9.7 TWh an elektrischer Arbeit, was circa 1.8 Prozent des Gesamtstromverbrauchs in Deutschland ausmacht beziehungsweise 90 Prozent der Jahresarbeit eines Kernkraftwerkes mit einer Leistung von 1.300 MW entspricht [Dor14, S.1].

Ein weiterer nennenswerter Aspekt des immens gestiegenen Informationsaufkommens ist die Entwicklung und Einführung neuer Technologien, die unter dem Oberbegriff *Big Data* zusammengefasst werden können. Allerdings ist dieser Begriff uneinheitlich definiert. Nach *Rieß* "*reichen die Datenmengen bis in den Peta- und Exabyte-Bereich und sind damit dermaßen gewaltig, dass sie über mehrere Rechner verteilt gespeichert und ausgewertet werden müssen*" [Rie13a]. Zudem "*sind die Daten so wenig strukturiert, dass man ihnen mit den Techniken relationaler Datenbanken und Data Warehouses nicht beikommt.*" Außerdem "*kommt oftmals hinzu, dass Daten aus einer Cloud-Umgebung –prädestiniert für BigData– durch Interaktion miteinander verbunden sind, was eine eindeutige Zuordnung und Analyse erschwert.*" Nach *De Mauro et al.* versteht man unter diesem Begriff die Nutzung von Technologien, die es ermöglichen, Datenmengen effektiver und in "tolerierbarer" Zeit zu verarbeiten als es mit konventionellen Methoden und Techniken möglich ist [DGG16]. Die Daten, die es zu verarbeiten gilt, werden durch fünf "V"s charakterisiert: *volume*, *variety*, *variability*, *velocity* und *veracity*. Die ersten drei Begriffe stimmen dabei mit der obigen Beschreibung von *Grieß* überein. Der vorletzte Begriff bezeichnet den Umstand, dass die Datenmengen nicht nur immens anwachsen, sondern dass das Wachstum in immer kürzeren Zeiträumen steigt. Der letzte Begriff bezeichnet, dass die Qualität der Daten, die erhoben und verarbeitet werden sollen, je nach Quelle und Methode durchaus erheblich schwanken kann. Nach [Rie13b] und [DGG16] sind prominente Beispiele für Technologien, die oft im Zusammenhang mit *Big Data* genannt werden, das *Map-Reduce*-Verfahrens und das Aufkommen der sogenannten *NoSQL*-Datenbanksysteme. Das *Map-Reduce*-Verfahren und dessen Implementierung *Hadoop* hat dazu geführt, dass selbst mit recht schwacher PC-Rechentechnik sehr große Datenmengen auf vielfältigster Weise verarbeitet werden können [DG04, Lin13, YDHP07]. Inzwischen hat sich ein großes Ökosystem rund um *Hadoop* gebildet und ermöglicht es Firmen, bisher ungenutzte Daten zu verarbeiten, um Geschäftsabläufe zu analysieren und zu optimieren.

Hinsichtlich der Datenbanksysteme hat sich gezeigt, dass die Leistung der traditionellen, meist kommerziellen Systeme nicht mit den zu verwaltenden Datenmengen skaliert. Das führte zu der Entwicklung von spezialisierten Datenbanksystemen, deren Architektur und Implementierung sich grundlegend unterscheidet. Am deutlichsten äußert sich dieser Umstand darin, dass eine standardisierte, universelle Abfragesprache wie *SQL* nicht mehr unterstützt wird. Daher stammt auch der Begriff *NoSQL* (engl. *Not Only SQL*). Dieser Schritt wird zumeist damit begründet, dass sich die *NoSQL*-Datenbanksysteme zur Steigerung der Leistung auf ein bestimmtes Datenmodell spezialisieren, zum Beispiel die ausschließliche Speicherung und Verwaltung von Schlüssel-Wert-Daten, Dokumenten oder Graphen. Zusätzlich werden auch spezielle Speichermodelle genutzt,

die diese Datenmodelle optimal unterstützen. Traditionell wurde das relationale beziehungsweise objekt-relationale Datenmodell genutzt. Mit Aufkommen der *NoSQL*-Datenbanksysteme kamen weitere Modelle hinzu, zum Beispiel das zeilen- oder Dokumenten-basierte Speichermodell. Auch andere Paradigmen, die fester Bestandteil der Implementierung traditioneller Datenbanksysteme waren, wurden entfernt oder durch abgeschwächte Äquivalente ersetzt. Dazu gehört zum Beispiel, dass die Konsistenz und Dauerhaftigkeit von gespeicherten Daten nicht mehr garantiert wird, um die Zugriffsgeschwindigkeit oder den Datendurchsatz zu erhöhen. Insgesamt kann festgestellt werden, dass die traditionellen Datenbanksysteme darauf ausgelegt waren, eine Vielzahl von Anwendungsfällen effektiv zu unterstützen, aber in speziellen Anwendungsfällen von *NoSQL*-Pendants verdrängt wurden, um den gestiegenen Anforderungen im Bezug auf Datenmenge und Zugriffsgeschwindigkeit gerecht zu werden.

Unumstritten ist, dass Datenbanksysteme eine große Rolle in Datenverarbeitungszentren spielen, da sie fundamentale Vorteile bieten. Dazu gehört die effektive Verwaltung und Speicherung von strukturierten und unstrukturierten Daten. Ebenso kann durch den Einsatz von Heuristiken die Suche nach bestimmten Daten anhand von Suchparametern effektiv unterstützt werden. Durch den Einsatz einer Datenabfrage- und -manipulationssprache können sehr schnell Anwendungen implementiert werden, die auf die gespeicherten Daten zugreifen können, ohne die dahinterliegende Speicherungsstruktur kennen zu müssen.

Analog zum Energieverbrauch der Datenverarbeitungszentren gibt es leider keine verlässlichen Aussagen darüber, wie groß der Anteil der Server gegenüber der Gesamtanzahl an Servern ist, die in nativer oder virtualisierter Form Datenbanksysteme ausführen. Ein Anhaltspunkt bietet der Rechenschaftsbericht aus dem Jahr 2014 der *Wikimedia Foundation*, die unter anderem auch die *Wikipedia*-Plattform bereitstellt. Mittlerweile betreibt sie weltweit vier Datenverarbeitungszentren mit insgesamt 350 Servern. Darunter befinden sich rund ein Viertel an Servern, die dediziert ein verteiltes Datenbanksystem[1] ausführen.

1.1 Motivation

Im vorhergehenden Textabschnitt ist die Bedeutung von Datenbanksystemen in der aktuellen IT-Landschaft gezeigt worden. Zudem dienen sie als fundamentale Grundlage zur Speicherung, Verarbeitung und Abfrage von allen erdenklichen digitalen Informationen. Somit kann davon ausgegangen werden, dass die Server in einem Datenverarbeitungszentrum, die ein Datenbanksystem dediziert ausführen, einen nicht unerheblichen Anteil an der Gesamtanzahl ausmachen. Zudem werden diese dedizierten Server im Gegensatz zu den übrigen Servern nicht abgeschaltet oder in einen Ruhezustand versetzt, da die Datengrundlage permanent zur Verfügung stehen muss [LP10b, LKP11, CII+10]. Dementsprechend kann auch davon ausgegangen werden, dass der Energieverbrauch[2] dieser dedizierten Server recht hoch ausfällt, wenn man sie zum Beispiel mit reinen Applikationsservern vergleicht.

Die beschriebenen Problematiken rund um die Thematik *Big Data* mit all seinen Konsequenzen sind bereits von Forschung und Wissenschaft aufgegriffen worden. So trafen sich 2008 renommierte Wissenschaftler in *Claremont*, um über den aktuellen Entwicklungsstand von Datenbanksystemen und zukünftige Forschungsfelder zu diskutieren. Die Ergebnisse sind im sogenannten *Claremont report* im Jahr 2009 veröffentlicht worden [AGMG+09]. Als eine der wesentlichsten

[1] Die *Wikimedia Foundation* nutzt das relationale Datenbanksystem *MariaDB* in einer Cluster-Konfiguration, sodass ein quasi-verteiltes Datenbanksystem entsteht (*MariaDB Galeria cluster*).

[2] Der Begriff Energieverbrauch wird im folgenden synonym zum Verbrauch elektrischer Arbeit verwendet.

Forschungsrichtungen wurde darin die Senkung des Energieverbrauches von Datenbanksystemen beziehungsweise die Optimierung deren Energieeffizienz für die folgenden Jahre genannt.

In der Praxis hat sich gezeigt, dass die Ressourcenplanung für die Betreiber von Datenbanksystemen eine sehr große Bedeutung gewonnen hat. Dieser Begriff beschreibt den Prozess, für einen gegebenen Anwendungsfall ein geeignetes Datenbanksystem auszuwählen oder für ein existierendes den optimalen Einsatz der Betriebsmittel zu planen. Um die Bedeutung zu illustrieren, können die folgenden drei Beispiele betrachtet werden:

1. Der Anwendungsfall sieht vor, dass ein großer Datenbestand eines bestimmten Datenmodells auf einer bereits existierenden technischen Plattform gespeichert und verwaltet werden soll. Solch ein Anwendungsfall kann zum Beispiel von einem Kunden an den Betreiber eines Datenverarbeitungszentrums gestellt werden, wobei auch zusätzliche Anforderungen wie zeitliche Schranken bei der Speicherung und dem Abruf vereinbart werden können. Der Betreiber wählt auf Grundlage des Anwendungsfalls dasjenige Datenbanksystem aus, dass am geeignetesten ist, um die Anforderungen zu erfüllen.

2. Ein weiterer Anwendungsfall ist das Ermitteln des Skalierungsverhalten eines Datenbanksystems. Dabei gilt es, die Grenzen eines existierenden Datenbanksystem auf einer existierenden technischen Plattform zu bestimmen. Dazu kann zum Beispiel die Größe des Datenbestandes sukzessive erhöht und die Speicherung beziehungsweise der Abruf von Daten getestet werden, bis festgelegte Voraussetzungen nicht mehr erfüllt sind. Das kann zum Beispiel die Überschreitung der Zugriffszeiten auf den Datenbestand oder die Überlastung der technischen Plattform sein.

3. Ein Anwendungsfall ergibt sich, wenn die Datenmenge und -modell sowie das Datenbanksystem bekannt sind. Er kann zum Beispiel dann auftreten, wenn ein Kunde den Betrieb dieser beiden Aspekte auf seiner eigenen technischen Plattform einstellt und stattdessen ein externes Datenverarbeitungszentrum mit dem Betrieb beauftragt. Insofern kann dieser Anwendungsfall als Kombination der beiden vorherigen angesehen werden, da der Betreiber des Datenverarbeitungszentrums nun die geeigneteste Kombination seiner technischen Plattformkomponenten ermittelt, um die Anforderungen des Kunden zu erfüllen.

Für die Hersteller von Datenbanksystemen ist noch ein vierter Anwendungsfall von Bedeutung: Regressionstests. Hierbei modifiziert der Hersteller die Implementierung, um Aspekte wie die Leistung (zum Beispiel die Zugriffszeit oder den Datendurchsatz) oder den Energieverbrauch zu optimieren. Sofern die technische Plattform sowie die Datenmenge und -modell nicht geändert werden, so können die Auswirkungen gegenüber der Vergängerversion ermittelt werden.

Generalisiert man diese vier Anwendungsfälle, so ist auffallend, dass es immer deren Ziel ist, eine optimale technische Plattform zu finden beziehungsweise einen optimalen Arbeitspunkt hinsichtlich der Leistung zu ermitteln. Optimal bedeutet in diesem Zusammenhang, dass die Leistung des Datenbanksystems und die technische Plattform den Anforderungen des Anwendungsfalles entsprechend genau dimensioniert ist. Dabei schließen sich beide Aspekte nicht aus. Im Bezug auf die Leistung eines Datenbankmanagementsystems muss angemerkt werden, dass der Leistungsbegriff nicht bestimmt ist. Im allgemeinen wird er aber mit der Zugriffsgeschwindigkeit oder mit dem Durchsatz auf die gespeicherten Daten gleichgesetzt. Dementgegen ist der optimale Arbeitspunkt bei der technischen Plattform bestimmt und bedeutet die Konfiguration an technischen Plattformkomponenten, die den geringsten Energieverbrauch zur Erfüllung der Anforderung aufweisen.

Zusätzlich fällt bei der Generalisierung auf, dass die Evaluation und Optimierung der Leistung beziehungsweise des Energieverbrauches eines Datenbanksystems auf die drei Parameter technische Plattform, Größe und Modell der Daten und das gewählte Datenbanksystem zurückgeführt werden können. Dabei werden zwei der drei Parameter als konstant angesehen und der verbleibende variiert. Fixiert man zum Beispiel die Größe sowie Modell der zu speichernden Daten und legt sich auf eine technische Plattform fest, so kann das geeigneteste Datenbanksystem ermittelt werden. Die so gewonnene Zusammenstellung der drei Parameter zeichnet sich dadurch aus, dass je nach Zielstellung die Leistung oder der Energieverbrauch optimal ist. In Tabelle 1.1 sind alle Parameterbelegungen aufgelistet und entsprechen den vier Anwendungsfällen.

Datengröße und -domäne	technische Plattform	Datenbankmanagementsystem	Zielstellung
fixiert	fixiert	variabel	geeignetestes Datenbankmanagementsystem
fixiert	fixiert	variabel	Regressionstest
fixiert	variabel	fixiert	geeigneteste technische Plattform
variabel	fixiert	fixiert	Skalierungsverhalten

Tabelle 1.1: Parameterbelegung und Zielstellung zur Evaluation und Optimierung der Leistung beziehungsweise des Energieverbrauches

Abstrakt betrachtet dienen die vier Anwendungsfallbeispiele dazu, eine *Vergleichbarkeit* zwischen zwei unterschiedlichen Versionen desselben Datenbanksystems oder zwei unterschiedlichen Konfigurationen herzustellen. Dabei werden die Auswirkungen auf die Leistung beziehungsweise Energieverbrauch bemessen. Eine Konfiguration besteht hierbei aus einer Zusammenstellung einer technischen Plattform, worauf ein Datenbanksystem ausgeführt wird, das einen Datenbestand verwaltet und Zugriffe auf diesen ermöglicht.

Es ist offensichtlich, dass gewisse Investitionen in Zeit und technische Komponenten getätigt werden müssen, um diese Vergleichbarkeit nachvollziehbar, überprüfbar und aussagekräftig zu gestalten. Um diese Bedingungen einzuhalten, werden im allgemeinen Fall anerkannte, standardisierte Benchmarks genutzt und ausgeführt, um Aussagen zum Leistungsverhalten eines Datenbanksystems zu erhalten. Das sind zum Beispiel Messwerte zu Zugriffszeiten von lesenden oder schreibenden Zugriffen auf den Datenbestand (Antwortzeiten) oder zum Datendurchsatz. Misst man zugleich den Energieverbrauch der technischen Plattform, auf dem dieser Benchmark ausgeführt wird, so können auch Aussagen zum elektrischen Energieverbrauch getroffen werden. Es ist auch offensichtlich, dass die Investitionen in Zeit (für die Ausführung des Benchmarks) und technischen Komponenten (um den Benchmark auf verschiedenen technischen Konfigurationen ablaufen zu lassen) mehr oder weniger stark ausgeprägt sind.

Bezogen auf die oben genannten vier Anwendungsfallbeispiele kommt erschwerend hinzu, dass eine *allumfassende* Evaluation der Leistung und des Energieverbrauches schlichtweg unpraktikabel ist. Möchte man zum Beispiel den Einfluss der technischen Plattform auf diese beiden Aspekte für ein gewähltes Datenbanksystem und fest definierter Datenmenge ermitteln, so müssen alle Kombinationen aller technischen Komponenten mittels eines geeigneten Benchmarks getestet werden, auf der das Datenbanksystem lauffähig ist. Das ist zwar nicht unmöglich, erfordert aber sehr großen zeitlichen Aufwand und beträchtliche monetäre Investitionen. Gleiches gilt, wenn zum Beispiel das Skalierungsverhalten von Datenbanksystemen verglichen werden soll. Bei circa 150 derzeit bekannten Datenbanksystemen[3] ist es äußerst schwierig, Benchmarks mit allen

[3] Stand Januar 2016. Detailliertere Informationen sind hierzu in Textabschnitt 2.4 verfasst.

denkbaren Größen der Datenmenge für jedes von ihnen auf derselben technischen Plattform auszuführen. In Anbetracht der Größen, die durch das Themenfeld *Big Data* vorgegeben werden, sind die zu testenden Größen fast illusorisch.

Dennoch kann festgestellt werden, dass Vergleichsstudien hinsichtlich der beiden Metriken Leistung und Energieverbrauch von Datenbanksystemen zu spezifischen, klar abgegrenzten Anwendungsfällen durchgeführt wurden und auch weiterhin werden. Die Gründe dafür sind zum Beispiel die Herausstellung von Produktmerkmalen zu Marketingzwecken, aber auch um bestehende Implementationen von Datenbanksystemen zu erweitern oder zu optimieren. Damit können neue Anwendungsfälle abgedeckt und existierende besser unterstützt werden.

Zugleich ermöglichen es Vergleichsstudien, die *Energieeffizienz* eines Datenbanksystems hinsichtlich eines Anwendungsfalles zu bestimmen. Die Energieeffizienz ist der Quotient aus Leistung und Energieverbrauch und kann als zusätzliches Optimierungsziel angesehen werden. In Anbetracht der bereits aktuellen Datenmengen und der durch *Big Data* noch zu erwartenden Mengen ist die Energieeffizienz bereits ins Blickfeld von Forschung und Praxis gerückt. So haben zum Beispiel *Leverich und Kozyrakis* in [LK10] gezeigt, dass der Großteil der untersuchten *Hadoop*-Cluster äußerst energieineffizient sind. Als eine Ursache wurde das verteilte Dateisystem *HDFS* ausgemacht, das in *Hadoop*-Clustern zum Einsatz kommt. Eine Studie von *Vasić et al.* zeigte ein großes Steigerungpotential der Energieeeffizienz, sofern Modifikationen an *HDFS* vorgenommen werden [VBSK09]. Eine solche Modifikation ist zum Beispiel *GreenHDFS*, dessen Grundlagen von *Kaushik* in [Kau10] publiziert wurden. Die Ergebnisse der Implementation von *GreenHDFS* wurden dann von *Kaushik et al.* in [KAEN11] vorgestellt. Dabei wurden auch Simulationen durchgeführt, welches Einsparpotential die Einführung von *GreenHDFS* haben würde. Die Simulationsergebnisse zeigen, dass der Energieverbrauch um circa 24 Prozent sinken könnte, womit die Energieeffizienz steigt.

Zusammengefasst bedeutet all dies, dass die *Vergleichbarkeit* die Basis zur Optimierung der Leistung, des Energieverbrauch und schließlich der Energieeffizienz eines Datenbanksystems darstellt. Erst wenn es möglich ist, zwei Versionen eines Datenbanksystems miteinander zu vergleichen, sind Regressionstests praktikabel. Wie bereits erwähnt, ist dieser Aspekt zum Beispiel für Anbieter von Datenbanksystemen äußerst wichtig. Wenn es zusätzlich möglich ist, zwei völlig unterschiedliche Datenbanksysteme hinsichtlich der oben genannten Metriken vergleichen zu können, so eröffnen sich weiterführende Aspekte: die Auswahl des geeignetesten Datenbanksystems und die Ressourcenplanung für einen definierten Anwendungsfall. Beide Aspekte sind für Endanwender von Datenbanksystemen interessant. Liegen Vergleichszahlen für die Metriken vor, können sie sich zum Beispiel für das leistungsfähigste oder energieeffizienteste Datenbanksystem entscheiden beziehungsweise aufgrund des Anwendungsfalles ihre Ressourcen planen und ändern.

1.2 Beitrag zu Forschung und Praxis

Um das Forschungsthema “Energieeffizienz von Datenbanken” in Hinblick auf sehr große Datenbestände eingehend zu betrachten, wurde auf Methoden der Verhaltensforschung sowie der Systemanalyse zurückgegriffen. Im allgemeinen bezieht sich die Verhaltensforschung auf das Beobachten und Studium des Verhaltens von Personengruppen und deren Mitglieder untereinander, aber die verwendeten Methoden und Vorgehensweisen können auf beliebige Systeme angewandt werden. Das bedeutet, dass dieselben drei Stufen durchlaufen werden, um ein Forschungsthema vollständig zu durchdringen. Die Stufen sind in Abbildung 1.1 auf Seite 7 dargestellt.

In der ersten Stufe werden alle potentiellen Einflussfaktoren, Metriken sowie deren Bestimmbarkeit zusammengetragen. Alle gefundenen Einflussfaktoren werden in der zweiten Stufe zunächst individuell and danach koopkurrent auf die Größe ihres Einflusses hin untersucht. Zudem werden erste praktische Ergebnisse sichtbar. In der dritten, finalen Stufe wird ein Modell gebildet, dass alle Ergebnisse der vorherigen Stufen abstrahiert. Dieses Modell wird dann genutzt, um neues Wissen zu deduzieren.

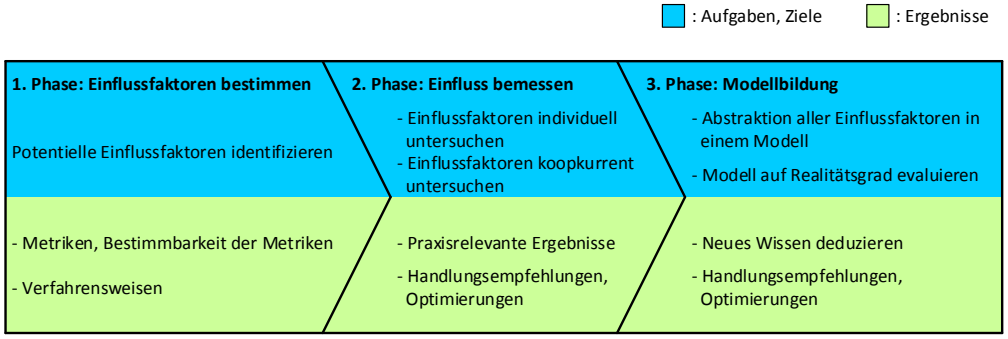


Abbildung 1.1: Vorgehensmodell zur Betrachtung des Forschungsthemas *Energieeffizienz von Datenbanken*

Dieses Vorgehen deckt sich mit den empirischen und semantischen Ergebnissen, die in der letzten Dekade publiziert wurden. So wurden erste Faktoren, die Einfluss auf die Energieeffizienz von Datenbanksystemen haben könnten, durch *Xu et al.*, *Wang et al.* und *Tsirogiannis et al.* bestimmt und evaluiert [XTW10, WFXS11, THS10]. Obwohl die Autoren dieser Publikationen nur relationale Datenbanksysteme untersuchten, wurden wichtige Einflussfaktoren zusammengetragen und erste Methodiken zur Bestimmung der Energieeffizienz vorgeschlagen. Zudem wurde auch eine erste These bestätigt, dass ein einzelner Faktor signifikanten Einfluss auf die Energieeffizienz haben kann. Zum Zeitpunkt des Erscheinens der oben genannten Publikationen wurden auch *NoSQL*-Datenbanksysteme populär, wobei auch das *Hadoop*-Ökosystem ein wesentlichen Anteil hatte. Beide waren Gegenstand zahlreicher Publikationen, die zusätzliche potentielle Einflussfaktoren auf die Energieeffizienz herausstellten, zum Beispiel von *Lang et al.*, *Abadi*, *Rabl et al.* und *Baru et al.* [LKP11, Aba12, RGVS+12, BBN+13].

Kombinationen der gefundenen Einflussfaktoren wurden dann auf ihre Einflussgröße hin untersucht, wobei mittlerweile etablierte Benchmarks genutzt wurden, zum Beispiel durch *Vasić et al.*, *Lang und Patel* und *Poess et al.* [VBSK09, LP09, PNV+10]. Die bis dato publizierten Ergebnisse wurden auch in der Praxis genutzt. So wurde versucht, die Energieeffizienz von etablierten Datenbanksystemen dadurch zu optimieren, dass entweder die Performance (Durchsatz oder Antwortzeit) gesteigert oder der Energieverbrauch gesenkt wurde. *Hindman et al.* [HKZ+11] zum Beispiel zeigten, dass es möglich ist, den Energieverbrauch bei existierende Datenbanksystemen zu senken, wenn man bei einer Datenbankabfrage den zu erwartenden Energieverbrauch einbezieht. Hierbei wurden existierende Heuristiken wie Datenbankindizes und Abfragestatistiken mit Energieverbrauchsmetriken kombiniert.

Um die Energieeffizienz weiter zu optimieren, wurde auch damit begonnen, bereits in der Projektierung und Implementierung von Datenbanksystemen die Performance und den Energieverbrauch effektiver zu gestalten. Ein prominentes Beispiel ist *F1*, ein relationales Datenbanksystem von *Google*, das diese beiden Aspekte bereits in den Hauptanforderungen definierte [SVSR13]. Auch in der Praxis und akademischen Lehre nahm das Thema Energieeffizienz einen immer

wichtigeren Stellenwert in Form des sogenannten *software performance engineering* ein. So zeigten zum Beispiel *Dolan und Moré*, dass es möglich ist, bereits existierende Software generell hinsichtlich der Energieeffizienz zu optimieren [DM02]. *Woodside et al.* sowie *Calero et al.* haben alle bisherigen Ansätze zusammengefasst und gezeigt, wie die Thematik Energieeffizienz in der Lehre angewandt werden kann [WFP07, CP15]. Insbesonders erstgenannte deuteten bereits darauf an, dass eine Modellierung der Energieeffizienz in Softwaresystemen möglich und praktikabel ist.

Der Hauptbeitrag dieser Dissertation zu Forschung und Praxis ist die Konkretisierung des Optimierungsproblems hinsichtlich der Energieeffizienz von Datenbanksystemen sowie ein neuer Ansatz zur Lösung dieses Problems. Dabei wurden Modellen erarbeitet und eingeführt, die es ermöglichen, die Performance und Energieverbrauch eines Datenbanksystems zu simulieren, sofern es einer gewissen Definition genügt. Um die wichtigsten Einflussfaktoren in den Modellen nachzubilden, wurde wichtige wissenschaftliche Vorarbeit geleistet. Diese Vorarbeiten können grob in zwei Abschnitte geteilt werden. Der erste Abschnitt befasste sich mit der Identifizierung und Evaluierung der Einflussfaktoren im Bezug auf Performance und Energieverbrauch von zentral betriebenen Datenbanksystemen (Einzelplatz-Datenbanksysteme). Die Ergebnisse wurden in einer Publikation zusammengefasst:

- In **Evaluating the Energy Efficiency of OLTP Operations** ([NKZG13]) wurden die Ergebnisse von drei Benchmarks für das relationale Datenbanksystem *Postgresql* in Hinblick auf Performance, Energieverbrauch und -effizienz vorgestellt. Dabei wurden speziell alle technischen Kernkomponenten (Netzteil, Mainboard, CPU und Massenspeicher) individuell vermessen, um die reellen Auswirkungen der Benchmarks zu erfassen. Zusätzlich wurden die Konsequenzen von Energiesparmechanismen und Kombinationen von wichtigen Datenbankabfragen evaluiert.

Der zweite Abschnitt untersuchte verteilte Datenbanksysteme, da deren Charakteristiken einen deutlichen Gewinn in puncto Performance, Ausfallsicherheit und Energieeffizienz versprechen.

- Da die vorherige Publikation nur die Auswirkungen bezüglich der Performance, des Energieverbrauchs und -effizienz für ein zentral betriebenes Datenbanksystem untersuchte, wurde in **Benchmarking DataStax Enterprise/Cassandra with HiBench** ([INI+15]) der deutlich komplexere Benchmark *HiBench* genutzt, um das verteilte *NoSQL*-Datenbanksystem *Cassandra* zu evaluieren. Hierbei wurden verschiedene *Cassandra*-Clustergrößen mit verschiedenen *HiBench*-Konfiguration kombiniert, um eine mehrdimensionale Auswertung vornehmen zu können.
- In **Benchmarking the Availability and Fault Tolerance of Cassandra** ([RNI+15]) wurde ein anderer Benchmark (*Yahoo Cloud Serving Benchmark*, *YCSB*) genutzt, um *Cassandra* hinsichtlich der Ausfallsicherheit mit besonderem Fokus auf die Auswirkungen auf die Performance und Energieverbrauch zu evaluieren. Auch hier wurden verschiedene *Cassandra*-Clustergrößen mit verschiedenen *YCSB*-Konfigurationen getestet, wobei verschiedene Ausfall-Szenarien forciert wurden.
- Da verteilte Datenbanksysteme gängige Transportprotokolle von Computernetzwerken nutzen, wurde in **Performance Evaluation of netfilter** ([NPG15]) untersucht, welchen Einfluss eine Firewall auf die Performance haben kann. Dabei wurden alle Transportprotokolle, die auf dem *Internet Protocol* (IP) zur Adressierung basieren, für verschiedene Szenarien evaluiert. Dazu zählen sowohl eine unterschiedliche Anzahl an Sender- und Empfängerendpunkten als auch eine starke Varianz in der Firewall-Konfiguration.

Während den wissenschaftlichen Vorarbeiten hat sich herausgestellt, das es möglich ist, die einflussreichsten Faktoren auf Performance und Energieeffizienz von Datenbanksystemen zu modellieren. Der Modellierungs- und Simulationsprozess wurde in den folgenden Publikationen verifiziert:

- In **Modelling the Performance, Energy Consumption and Efficiency of Data Management Systems** ([NI15b]) wurden die Gründe und Vorteile für eine Modellierung der Performance und Energieeffizienz von Datenbanksystemen dargelegt. Die grundlegenden Modelle und deren Eigenschaften wurden beschrieben. Zudem wurden wichtige Metriken und erste Simulationsergebnisse präsentiert, die experimentell belegt wurden.
- Die grundlegenden Modelle wurden in **Evaluating the Energy Efficiency of Data Management Systems** ([NI15a]) konkretisiert. Sie wurden dahingehend genutzt, um die experimentellen Ergebnisse aus der vorherigen Publikation [NKZG13] zu simulieren. Verglichen mit den experimentellen Messergebnissen zeigten die simulierten Messergebnisse eine Abweichung von durchschnittlich 24 Prozent.
- Da die grundlegenden Modelle nicht dafür genutzt werden können, verteilte Datenbanksysteme zu modellieren, wurden sie überarbeitet. Die entstandenen erweiterten Modelle sind in der Lage, Computernetzwerke und darauf ausgeführte verteilte Datenbanksysteme abzubilden. In **Evaluating the Performance and Energy Consumption of Distributed Data Management Systems** ([Nie15]) wurden die erweiterten Modelle eingeführt und erläutert. Um die Modelle zu validieren, wurden die experimentellen Ergebnisse aus [RNI+15] genutzt, in der das verteilte Datenbanksystem *Cassandra* genutzt wurde. Auch hier wurden diverse Kombinationen, unter anderem verschiedene *Cassandra*-Clustergrößen und unterschiedliche Konfigurationen des *YCSB*, genutzt, um Stresstests durchzuführen. Die erweiterten Modelle wurden dergestalt verwendet, dass die Experimente simuliert werden konnten. Ein Vergleich der experimentellen Messergebnisse mit denjenigen der Simulationen zeigt, dass sie eine mittlere prozentuale Abweichung von 20 Prozenz besitzen.
- Die erweiterten Modelle wurden auch genutzt, um die Größe des *Cassandra*-Clusters zu prognostizieren, in der die Energieeffizienz optimal ist. Die optimale Clustergröße konnte in [Nie15] nicht validiert werden, da der Versuchsaufbau nicht die erforderlichen Dimensionen bot. Um die Prognosen zu bestätigen, wurde ein zweiter Versuchsaufbau genutzt und die Experimente aus [Nie15] wiederholt. Die Ergebnisse wurden in **Towards the Prediction of the Performance and Energy Efficiency of Distributed Data Management Systems** ([Nie16]) publiziert. Durch weitere Optimierungen der erweiterten Modelle zeigten die experimentellen und simulierten Messergebnisse eine prozentuale mittlere Abweichung von 8 Prozent für die Performance sowie 16 Prozent für den Energieverbrauch.

Die Inhalt der genannten Publikationen wurden in dieser Dissertation verwendet. In Tabelle 1.2 wird dargestellt, welche Inhalte in welchen Textabschnitten genutzt wurden.

Publikation	Kapitel 3	Kapitel 4	Kapitel 5	Kapitel 6
[NKZG13]	3.1 bis 3.3	4.1		
[NPG15]			5.5 bis 5.7	
[NI15b]	3.1 bis 3.3		Kapiteleinleitung	Kapiteleinleitung
[NI15a]	3.1 bis 3.3	4.1	5.3, 5.4	6.2
[Nie15], [Nie16]	3.1 bis 3.3	4.2	5.5 bis 5.7	6.3

Tabelle 1.2: Verwendung der Publikationsinhalte in den einzelnen Kapiteln

1.3 Struktur dieser Dissertation

Die Inhalte der nachfolgenden Kapitel sind so konzipiert, dass sie aufeinander aufbauen. Dabei orientieren sich die Inhalte der Kapitel am Vorgehensmodell, wie es im vorherigen Textabschnitt 1.2 beschrieben ist. Zur Erläuterung der Struktur wurde das Vorgehensmodell von Abbildung 1.1 so annotiert, das erkenntlich wird, welcher Inhalt in welchem Kapitel zu erwarten ist. Das annotierte Vorgehensmodell ist in Abbildung 1.2 dargestellt.

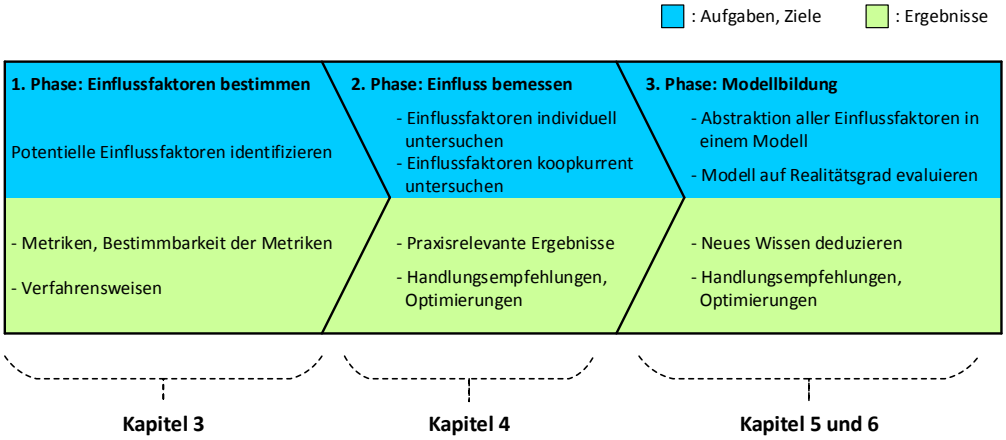


Abbildung 1.2: Struktur der Dissertation

So werden in **Kapitel 2** wichtige Grundbegriffe definiert und erklärt, die für das Verständnis der nachfolgenden Kapitel von Bedeutung sind. Dazu gehören zunächst Erläuterungen und Definitionen der wichtigsten Architekturelemente von Datenbanksystemen in den **Textabschnitten 2.1** bis **2.4**. Zudem wird darauf eingegangen, welche Fortschritte und Unterschiede hinsichtlich der Architektur moderne Datenbanksysteme in den letzten Jahren durchlaufen haben. In **Textabschnitt 2.5** werden *Petri*-Netze und dessen Derrivate erläutert, da diese die Grundlage für eine Modellierung und Simulation von Datenbanksystemen bilden.

In **Kapitel 3** werden diejenigen Metriken von Datenbanksystemen identifiziert und erklärt, die für diese Dissertation von Bedeutung sind. Dazu gehören die Performance, der Energieverbrauch und die Energieeeffizienz. In **Textabschnitt 3.1** wird erläutert, wie die "Leistung" (Performance) von Datenbanksystemen gemesssen und bewertet werden kann. Dabei werden wichtige Metriken und Aspekte eingehend betrachtet, die dann in der Begriffsbestimmung der Performance münden. Wichtige Faktoren, die einen signifikanten Einfluss auf die Performance von Datenbanksystemen haben, werden herausgestellt und deren Einflussgröße durch experimentelle Ergebnisse evaluiert. Analog dazu werden in **Textabschnitt 3.2** die Möglichkeiten und Verfahrensweisen beschrieben, wie der Energieverbrauch eines Datenbanksystem als zusätzliche Metrik zur Performance gemessen und beurteilt werden kann. Dafür werden allgemeine Messverfahren erläutert und deren Vor- und Nachteile diskutiert. Durch experimentell gewonnene Messergebnisse werden die technischen Komponenten identifiziert, die sowohl einen Einfluss auf den Energieverbrauch als auch auf die Performance haben. Beide Metriken –Performance und Energieverbrauch– werden genutzt, um in **Textabschnitt 3.3** den Begriff der Energieeffizienz zu definieren. Dies dient als Basis für die Beschreibung der Möglichkeiten, wie Datenbanksysteme hinsichtlich der drei Aspekte Performance, Energieverbrauch und schließlich Energieeffizienz

bewertet und optimiert werden können. In **Textabschnitt 3.4** werden alle gewonnenen Erkenntnisse für die drei Aspekte Performance, Energieverbrauch und Energieeffizienz von Datenbanksystemen zusammengefasst und die identifizierten Einflussfaktoren dargestellt.

Die in Kapitel 3 identifizierten Einflussfaktoren für die Performance, den Energieverbrauch und die Energieeffizienz von Datenbanksystemen wurden hinsichtlich ihrer Einflussgröße experimentell untersucht. Die Experimente inklusive des Versuchsaufbaus, der Versuchsdurchführung und der Analyse der gewonnenen Metrikwerte sind in **Kapitel 4** beschrieben. Dabei wurden Experimente mit einem zentral betriebenen Datenbanksystem durchgeführt, die in **Textabschnitt 4.1** erläutert sind. Analog dazu werden in **Textabschnitt 4.2** die Ergebnisse der Experimente mit einem verteilten Datenbanksystem dargestellt. **Textabschnitt 4.3** fasst die Ergebnisse und Aussagen von Kapitel 4 zusammen.

Auf Basis von Kapitel 2 bis 4 wird in **Kapitel 5** erläutert, wie die identifizierten Einflussfaktoren hinsichtlich Performance und Energieverbrauch abstrakt in einem Optimierungsproblem beschrieben werden können, wobei das Optimierungsziel die Maximierung der Energieeeffizienz von Datenbanksystemen ist. Dabei werden verschiedene Lösungsmöglichkeiten betrachtet und als Resultat dessen eine abstrakte Modellierung gegeben, um mit ihnen Simulationen durchführen zu können. In den **Textabschnitten 5.3** bis **5.7** werden alle notwendigen Modelle detailliert beschrieben und deren Funktionsweise erläutert. Die Vor- und Nachteile der eingeführten Modelle wird in **Textabschnitt 5.8** diskutiert.

Die in Kapitel 5 eingeführten und erläuterten Modelle wurden dafür genutzt, die Experimente aus Kapitel 4 nachzustellen und zu simulieren. Die Ergebnisse der Simulationen werden in **Kapitel 6** dargestellt. Dabei wird in den **Textabschnitten 6.3** und **6.3** auch darauf eingegangen, wie die Modelle zur Simulation angepasst und wie die Simulationen durchgeführt wurden. Der Vorteil der Simulation der Experimente aus Kapitel 4 ist, dass die Simulationsergebnisse mit denjenigen der Experimente verglichen werden können. Dadurch kann die Qualität der Modelle anhand ihrer Genauigkeit evaluiert werden, wie es in **Textabschnitt 6.4** detailliert beschrieben wird. **Textabschnitt 6.5** fasst alle Erkenntnisse von Kapitel 6 zusammen.

In **Kapitel 7** werden alle wissenschaftlichen Erkenntnisse dieser Dissertation nochmals zusammengefasst. Zusätzlich werden mögliche Richtungen beschrieben, wie die in dieser Dissertation begonnene Forschung weiter fortgeführt werden kann.

Hinweis zu den verwendeten Textformatierungen

Innerhalb dieser Dissertation werden zwei gesonderte Textformatierungen verwendet:

- *Kursive Texte* bezeichnen Eigennamen, Zitate und Definitionen sowie die Herausstellung bestimmter Wörter und Wortgruppen zum Zwecke der besonderen Betonung.
- `Monotype`-Texte bezeichnen Schlüsselwörter, etwa aus Programmier- und Abfragesprachen, sowie Quelltexte und Konstanten.

Hinweis zum Begriff Energieverbrauch

Der Begriff Energieverbrauch wird in der gesamten Dissertation synonym zum Verbrauch von elektrischer Arbeit gebraucht. Um Verwechslungen mit dem Begriff elektrische Leistung zu vermeiden, wird dieser Begriff klar in den Textstellen genutzt, die sich auf die elektrische Leistung beziehen.

2 Grundlegende Begriffe und Sachverhalte

Dieses Kapitel dient dazu, grundlegende Begriffe und Sachverhalte zu erläutern. Sie dienen als theoretische Grundlage für die nachfolgenden Kapitel. Dementsprechend stellt dieses Kapitel den nicht-originären Teil dieser Dissertation dar.

2.1 Datenbanken und deren Datenmodelle

Um eine logisch zusammenhänge Menge von Informationen und Fakten strukturell zu erfassen, wird ein sogenannten *Datenmodell* genutzt. Nach *Ferstl und Sinz* sind Datenmodelle das Ergebnis mehrere Entwicklungsstufen, die letztendlich zu lauffähigen Datenbeständen führen [FSV05]. Die dabei eingesetzten Verfahren der Datenmodellierung dienen dazu, die in einer Domäne relevanten Objekte, deren Attribute und die Beziehungen untereinander formal abzubilden. Datenmodelle geben somit eine *logische* Sicht auf die Datendomäne. Das bedeutet, dass sie nicht von einem technischen System abhängig sind, das das Datenmodell in letzter Instanz praktisch umsetzt.

In Abbildung 2.1 auf Seite 13 ist eine Auswahl an Datenmodellen sowie Datenbanken als Evolutionsgraph dargestellt. Das Jahr, in dem das Datenmodell formal beschrieben wurde beziehungsweise erste Implementationen einer Datenbank erschienen, kann an der linken Seite abgelesen werden.

Nach *Elmasri und Navathe* sowie *Geisler* bezeichnet man eine Menge von logisch zusammenhängenden Daten oder Informationen als *Datenbank* beziehungsweise *Datenbasis* [EN09, Gei14]. Sie folgt einem Datenmodell. Das prominenteste Datenmodell ist sicherlich das *Entity-relationship*-Modell, das 1976 durch *Peter Chen* beschrieben wurde [Che76]. Ein weit verbreiteter Irrtum ist, dass dieses Datenmodell die formale Grundlage von sogenannten *relationalen Datenbanken* ist. Sie wurden bereits 1970 von *Edgar F. Codd* vorgestellt [Cod70]. Relationale Datenbanken basieren auf einer von *Codd* entwickelten relationalen Algebra[1]. Relationale Datenbanken bestehen aus *Tabellen* (Relationen), in denen die Daten zeilenweise verwaltet werden (*Tupel*, *Datensatz*). Jedes Tupel setzt sich aus einer Reihe von Attributen zusammen (Spalten der Tabelle). Relationale Datenbanken lassen auch Verknüpfungen zwischen Tabellen zu. Sofern Attribute innerhalb einer Relation eindeutig sind, das heisst sie identifizieren ein Tupel eindeutig, können sie zum Bei-

[1] Eine rationale Algebra beschreibt, wie Daten in Relationen (im mathematischen Sinne) modelliert, gespeichert, abgefragt und modifiziert werden. *Codd* definiert in [Cod70] auch grundlegende Operationen für diese Relationen: Projektion, Selektion, Kreuzprodukt, Vereinigung und Differenz.

spiel für die referentielle Integrität herangezogen werden. Zusätzlich definiert *Codd* in [Cod70] auch vier Normalisierungsschritte, die das Ziel verfolgen, Redundanzen zu vermeiden und Inkonsistenzen in den Relationen aufzudecken. Im Bezug auf das *Entity-relationship*-Modell hat die Praxis aber gezeigt, dass es ein geeignetes Mittel ist, um eine Anwendungsdomäne hinsichtlich ihrer Objekte, deren Attribute und der Beziehungen der Objekte untereinander zu modellieren. Im Vergleich zur relationalen Algebra bezeichnen Relationen hier aber nur die Beziehungen zwischen den identifizierten Objekten. Allerdings können *Entity-relationship*-Modelle durch einfache formale Regeln in ein relationales Modell überführt werden. Dieser Umstand macht die Popularität beim Einsatz dieses Modells als Vorbereitung für eine relationale Datenbank aus.

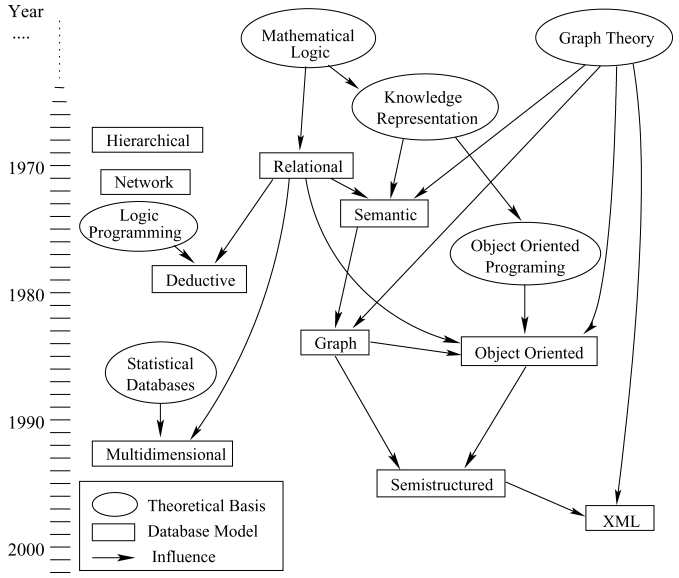


Abbildung 2.1: Evolution der Datenbanken und deren zugrunde liegendes Datenmodell sowie die gegenseitige Beeinflussung nach [AG08]

Mit Aufkommen der objekt-orientierten Programmiersprachen kamen auch *objekt-orientierte Datenbanken* auf [Dev01]. Sie basieren darauf, dass die Daten nicht mehr in Relationen verwaltet werden, sondern im Sinne des *objektorientierten Datenmodells* als Objekte angesehen werden [BPR88]. Ein Objekt bedeutet hierbei, dass logisch zusammenhängende Daten in Attribute mit denjenigen Methoden kombiniert werden, die für die Attribute definiert wurden. Hierdurch wird versucht, die Struktur und das Verhalten abstrakt als Objekt zusammenzufassen, was auch als abstrakter Datentyp aufgefasst werden kann (vgl. [Vos94, S.342],[KE11, S.385],[EN09, S.271]). Objekte mit gleicher Ausprägung, dass heisst mit gleicher Struktur und Verhalten, können zu Klassen zusammengefasst werden und zur Erzeugung weiterer Objekte (sogenannte Instanzen) herangezogen werden. Die Instanzen werden durch sogenannte *object identifier* oder OIDs innerhalb der Datenbank unterschieden. Das entspricht im Vergleich zu relationalen Datenbanken einem Primärattribut. Zudem können in einer objekt-orientierten Datenbank auch die Paradigmen objekt-orientierter Programmiersprachen angewandt werden. Das sind unter anderem die Kapselung von Struktur und Verhalten, die Änderung des internen Zustandes eines Objektes über den Aufruf dessen Methoden oder durch den Versand und Empfang von Nachrichten sowie die Spezialisierung von Ober- zu Unterklassen, Vererbung und Polymorphismus. Es sei bemerkt, dass

die formale Definition des objekt-orientierten Datenmodells im Bezug auf Datenbanken recht komplex ist. Eine ausführliche Darstellung kann bei *Vossen* nachverfolgt werden [Vos94, S.336 ff.].

Neben den genannten Datenmodellen treten auch Mischformen auf. So tragen *objekt-relationale Datenbanken* Eigenschaften von relationalen als auch objekt-orientierten Datenbanken beziehungsweise deren zugrunde liegenden Datenmodellen. Der Grund liegt darin, dass beide Modelle gewisse Analogien, aber auch Stärken und Schwächen haben. Objekt-relationale Datenbanken versuchen die Stärken der beiden zugrunde liegenden Datenmodellen zu kombinieren: die formale Abfrage von Daten durch das Relationkalkül[2] und die Flexibilität bei der Definition des Datenschemas. *Kemper und Eickler* sehen objekt-relationale Datenbanken pragmatischer. So sind sie "nur" relationale Datenbanken, wobei "*Konzepte der objekt-orientierten Datenmodellierung [...] integriert wurden*" [KE11, S.419]. Die Erweiterungen betreffen die folgenden Konzepte: große Objekte (*large objects*, LOBs), mengenwertige Attribute, geschachtelte Attribute, Typdeklarationen, Referenzen, Objektidentitäten (OIDs), Pfadausdrücke, Vererbung und Operationen (Objektmethoden) [KE11, S.417 ff.].

Die Gemeinsamkeit von relationalen, objekt-orientierten und -relationalen Datenbanken besteht darin, dass sie schemabehaftet ist. Das Schema beschreibt die Struktur einer Datenbank und ist zeitlich invariant. Das heisst, sie ändert sich im Gegensatz zum Inhalt der Datenbank nur sehr selten [Vos94, S.23]. Im Gegensatz dazu basieren schemalose Datenbanken auf anderen Datenmodellen, wobei die Struktur der Datenbank nicht fixiert ist.

So werden in *Schlüssel-Wert-Datenbanken* (engl. *key-value databases*) die Daten in einem assoziativem Feld beziehungsweise Wörterbuch organisiert. Ähnlich dem relationalen Datenmodell werden die Daten zeilenweise verwaltet, wobei die Datensätze nur aus zwei Elementen bestehen: dem Schlüssel, der den jeweiligen Datensatz eindeutig identifiziert und dem Wert, in denen ein oder mehrere Attribute zusammen gespeichert werden. Insgesamt kann man sich das als 2-spaltige Tabelle vorstellen, wobei die erste Spalte die Schlüssel und die zweite die jeweiligen Werte enthält. Der Wert besteht im einfachsten Fall aus nur einem Attribut, aber es können auch mehrere verwaltet werden. Durch die Möglichkeit, eine variable Anzahl an Attributen im Wert eines Schlüssel verwalten zu können, ergibt sich kein einheitliches Schema, dass für alle Datensätze gilt. In Tabelle 2.1(a) ist zur Illustration dieses Sachverhaltes der Aufbau und der Inhalt einer *Schlüssel-Wert*-Datenbank dargestellt.

Schlüssel	Wert
K1	abc,123
K2	abc
K3	1234,abc,xyz
K4	xyz,123

(a) Schlüssel-Wert-Datenbank

Schlüssel	Wert
K1	Name: Mustermann, Vorname: Max, Alter: 43
K2	Name: Mustermann, Vorname: Erika
K3	Name: Mustermann, Vorname: Ralf, Alter: 13
K4	Name: Normalverbraucher, Vorname: Otto, Anschrift: Musterstrasse 1

(b) Dokumenten-orientierte Datenbank

Tabelle 2.1: Beispiel des Aufbaus und Inhaltes einer *Schlüssel-Wert-* und *Dokumenten-orientierten* Datenbank

[2] Das Relationkalkül ist nach *Vossen* ist eine "*deskriptive Sprache, welche die Tupel einer Ergebnisrelation durch Eigenschaften beschreibt*". Sie ist mit der Sprache der Prädikatenlogik erster Stufe eng verwandt und dient der Ermittlung einer endlichen Tupelmenge aus Operationen auf Relationen. Die formale Definition ist bei *Vossen* nebst Beispielen beschrieben [Vos94, S.165 ff.].

Schlüssel-Wert-Datenbanken stellen auch die Grundlage für die sogenannten *Dokumenten-orientierte Datenbanken* dar. Im Gegensatz zu erstgenannten wird der Umstand genutzt, dass die Werte in den Schlüssel-Wert-Datensätzen eine gewisse Struktur aufweisen. Das bedeutet, sie sind semi-strukturiert. Betrachtet man das Beispiel einer Dokumenten-orientierten Datenbank in Tabelle 2.1(b), so erkennt man, dass die Werte der Schlüssel `K1` bis `K4` eine gemeinsame Struktur aufweisen. So besitzen alle Werte die Attribute Vorname und Name sowie die optionalen Attribute Alter und Anschrift. Diese Struktur wird *Dokument* genannt. Somit wird pro Schlüssel ein Dokument verwaltet, das aus obligatorischen und optionalen (zumeist benannten) Attributen besteht. Der Vorteil hierbei ist, dass die Struktur beziehungsweise Schema nicht fest ist. Wenn im Beispiel aus Tabelle 2.1(b) entschieden wurde, das Attribut Bundesland dem Schema hinzuzufügen, so können nachfolgend gespeicherte Datensätze dieses neue Attribut nutzen, ohne das die vorherigen von dieser Strukturänderung betroffen sind. Diese Eigenschaften sind auch bei Daten erkennbar, die per XML[3] formatiert und organisiert sind. Hier kann ein Schema genutzt werden, das die Struktur der Attribute von Entitäten beschreibt. Fasst man diese Entitäten als Datensätze einer Dokumenten-orientierten Datenbank auf und vergibt Schlüsselwerte für jede Entität, so können die Attribute der Entitäten in den Werten gespeichert werden. Insgesamt eignen sich Dokumenten-orientierte Datenbanken im besonderen dazu, XML-formatierte Daten zu importieren oder Datensätze XML-formatiert zu exportieren. Im weitesten Sinne können somit Dokumenten-orientierte Datenbanken als Schlüssel-Wert-Datenbanken aufgefasst werden, auf deren Werte am ehesten mit Methoden und Verfahren zugegriffen wird, wie sie aus der Verarbeitung von XML-formatierten Daten bekannt sind [KE11, S.588].

Sogenannte *Graph-Datenbanken* basieren auf dem *Graph-orientierte Datenmodell*. Nach *Robinson et al.* bilden die Daten der Datenbank einen Graphen beziehungsweise ein Netzwerk [RWE13]. Im genauerem bedeutet es, das die Daten aus Knoten und Kanten bestehen. Die Kanten können gerichtet sein und Attribute tragen. Ein prominentes Beispiel ist die Verwaltung eines Soziogrammes in einer Graph-Datenbank: die Knoten repräsentieren die Individuen einer menschlichen Gruppe und die Kanten ihre Beziehungen untereinander. Solche Beziehungen können zum Beispiel “Person X kennt Person Y” oder “Person X ist verwandt mit Person Y” sein. Diese Art Beziehungen werden als Attribut oder Attributkombinationen an den Kanten annotiert. Eine solche Graph-Datenbank ist zum Beispiel die Geschäftsgrundlage der Firma *Facebook*. Neben Soziogrammen können aber auch andere Anwendungsbereiche modelliert werden, zum Beispiel Urheberschaftsbeziehungen, Nahrungsketten oder Protein-Interaktionen [SPÖ$^+$10, S.40]. Graph-Datenbanken können nach *Robinson et al.* sowohl schemafrei als auch semi-strukturiert sein und können analog zu Schlüssel-Wert-Datenbanken betrachtet werden: hierbei werden die Knoten als Schlüssel und als Wert eine Zielknoten-Attribut-Kombination aufgefasst. Diese Kombination kann natürlich anstatt eines einzigen Attributes auch eine Attributliste beinhalten. Sofern die Attributliste eine gewisse Struktur aufweist, ist die gesamte Datenbank semi-strukturiert [AG08]. Interessant ist der Fakt, dass die dominierenden Implementationen von Graph-Datenbanken zumeist als Schlüssel-Wert-Datenbanken ausgeführt sind [SPÖ$^+$10, S.38-40]. Es muss auch angemerkt werden, dass im weitesten Sinne sogenannte *triple stores* mit Graph-Datenbanken in Verbindung gebracht werden können. Diese Datenbanken werden hauptsächlich in semantischen Netzwerken genutzt und verwalten Daten in der Form Subjekt-Prädikat-Objekt [Tol06, S.9 ff.][KE11, S.659].

[3] XML ist die Abkürzung für *Extensible Markup Language* und ist eine vom *World Wide Web Consortium* standardisierte Sprache zur Darstellung von semi-strukturierten Daten. XML-formatierte Daten können implementations- und plattformunabhängig ausgetauscht und verarbeitet werden und sind sowohl maschinen- als auch menschenlesbar. Die Daten können einem Schema folgen. Die Daten können verschachtelt in Elementen organisiert werden, wobei immer ein Wurzelelement existieren muss. Die formale Definition von XML ist in [BPSM$^+$08] beschrieben.

Dahingehend existieren bereits zahlreiche Standards, wobei der hauptgebräuchlichste *RDF*[4] ist. Eine *RDF*-Datenbank kann als gerichteter Graph dargestellt werden.

Neben den genannten Datenbanken und deren Datenmodellen existieren noch drei weitere (deduktive, hierarchische und Netzwerk-Datenbanken), auf die im Rahmen dieser Dissertation nicht eingegangen wird, da sie kaum noch Bedeutung haben. An dieser Stelle sei auf die Literatur verwiesen, zum Beispiel *Vossen* [Vos94, S.9 ff.]. Ebenso wird das *erweiterte relationale Datenmodell* hier nicht beschrieben (vgl. dazu [EN09, S.235 ff.] und [Vos94, S.605 ff.]). Zuletzt sei auch bemerkt, dass sowohl *Vossen*, *Elmasri und Navathe* als auch *Kemper und Eickler* auf die Historie und die Einflussnahme der hier beschriebenen Datenmodelle und Datenbanken beschreiben. Dennoch sei auf die sehr gute Zusammenfassung in der Studie zu Graph-Datenbanken von *Angles und Gutierrez* verwiesen [AG08].

2.2 Speichermodelle

Wie bereits erwähnt, stellen die beschriebenen Datenmodelle die Grundlage für Datenbanken dar und vermitteln eine logische Sicht auf die Datenmenge. Datenbanken werden zumeist von Informations(-verarbeitenden) Systemen[5] verwaltet. Das sind vorrangig PCs und Server. Dementsprechend definieren sogenannte *Speichermodelle*, wie die Datenbank physikalisch auf den Datenträgern und Betriebsmitteln gespeichert werden. Im Bezug auf Datenbanken hat sich in der Praxis gezeigt, dass zumeist Primär-, Sekundär- und Tertiärspeicher[6] genutzt wird [Vos94, S.395]. Als Primärspeicher kommt dabei der Arbeitsspeicher für den Zugriff auf häufig genutzte Daten in Betracht. Als Sekundärspeicher werden zumeist schnelle Massenspeicher genutzt, etwa Festplatten. Als Tertiärspeicher kommen zum Beispiel magnetische Bänder und magneto-optische Medien zur Langzeitarchivierung zum Einsatz.

Im allgemeinen kann davon ausgegangen werden, dass in modernen Informationssystemen hinsichtlich der Speicherkapazität mehr Massen- als Arbeitspeicher eingesetzt wird. Zur Beschleunigung des Zugriffes auf häufig genutzte Daten werden dabei Techniken genutzt, die Teile des Haupt- in den Massenspeicher verlagern und umgekehrt (sogenannte *Caches*, Puffer beziehungsweise Zwischenspeicher). Diese Technologien teilen beide Speicherarten in Segmente gleicher, fester Größe ein (sogenannte Blöcke, Segmente beziehungsweise Seiten). Das dient dazu, die Verwaltung von verlagerten Seiten zu vereinfachen. Zudem hat diese Segmentierung auch einen technischen Hintergrund: am Beispiel einer magnetischen Festplatte werden Daten mit fester Blockgröße gelesen und geschrieben. Dabei haben sich Blockgrößen von 64, 128, 512 und 4096 Byte etabliert. Um den Transfer von Blöcken effizient zu gestalten, orientieren sich die oben genannten Seitengrößen an den Blockgrößen. Das äußert sich zum Beispiel darin, wenn der Inhalt eines Blockes geändert werden soll. Dazu muss der entsprechende Block vom Massenspeicher

[4] *RDF* steht für *Resource Description Framework* und bezeichnet einen Satz an Standards des *World Wide Web Consortium*. Nach [Tol06, S.13] dient *RDF* zum Austausch und zur Beschreibung von Metadaten über Resourcen (Subjekt, Objekte), die Aussagen zueinander bilden (Prädikat).

[5] *Vossen* definiert in [Vos94, S.31] diesen Begriff in folgender Form: "*Ein Informationssystem ist ein Werkzeug zur Erfassung und Kommunikation von Informationen zum Zwecke der Erfüllung der Anforderung seiner Benutzer, der (Geschäfts-) Aktivitäten ihres Unternehmens und zur Erreichung der Unternehmensziele. Ein Informationssystem unterstützt die Unternehmensaktivitäten durch Bereitstellung der benötigten Informationen oder durch Automatisierung der mit den Aktivitäten zusammenhängenden Vorgänge. Es umfasst sämtliche, zu diesem Zweck im Unternehmen vorhandenen Resourcen, d.h. die Daten, die DBMS-Software, die nötige Rechner-Hardware, die Personen, die diese Daten benutzen und verwalten, die relevante Anwendungssoftware sowie die Programmierer, die diese entwickeln.*"

[6] Aus Sicht des Hauptprozessors werden die verfügbaren Speicher nach Zugriffszeit geordnet, wobei mit steigenden Maße die Zugriffszeit, der Datendurchsatz und der Preis sinkt, während die Speicherkapazität steigt.

gelesen und in den Arbeitsspeicher transferiert werden, um dort den Inhalt zu ändern. Danach kann er wieder auf den Massenspeicher zurückgeschrieben werden. Sofern sich die Seiten- und Blockgrößen unterscheiden, sind zusätzliche Schreib- und Leseoperationen notwendig, das den gesamten Prozess der Inhaltsänderung zeitlich verlängert.

Zusätzlich kann im Falle von PCs und Servern immer davon ausgegangen werden, dass Daten in Form von Dateien zum Zwecke der Persistierung gespeichert werden und in baumartigen Verzeichnisstrukturen logisch organisiert werden [Vos94, S.395]. Dazu werden sogenannte Dateisysteme genutzt, die eine Mittelschicht zwischem dem reinen Zugriff auf die Daten einer Datei und den Primär-, Sekundär- und Tertiärspeicher darstellen. Somit kann man in der überwiegenden Anzahl von Fällen davon ausgehen, dass auch die Speichermodelle vorhandene Dateisysteme nutzen, um die Daten einer Datenbank zu speichern und zu verwalten.

Zur Illustration der existierenden Speichermodelle sei im folgendem eine relationale Datenbank angenommen, die aus einer einzigen Relation und drei Entitäten (Tupeln) besteht. Die Tupel bestehen aus drei Attributen, die in den Abbildungen 2.2 und 2.3 durch die Farben grün, orange und blau gekennzeichnet sind. Zudem wird angenommen, dass die Datenbank in Dateien organisiert ist.

Es haben sich vier Speichermodelle herausgestellt:

1. **Zeilen-basiertes Speichermodell** Die Attribute einer Entität werden zeilenweise gespeichert. Die Speichermodell wird in erster Linie bei relationalen Datenbanken eingesetzt. Hier wird jede Relation in einer eigenen Datei verwaltet, in der die Tupel der Relation nacheinander gespeichert sind. Der Hintergrund dieses Speichermodells ist es, dass die Attribute einer Entität möglichst "nahe" aneinander gespeichert werden. Die Entität soll somit möglichst im ganzen gelesen, modifiziert und zurückgespeichert werden. Dieses Vorgehen wird auch *Cluster*-Bildung genannt [Vos94, S.413]. Der Vorteil dieses Speichermodells ist es, Zugriffe auf Datenbanken zu unterstützen, deren Ziel es ist, vollständige Entitäten zu verarbeiten. Das ist im allgemeinen bei sogenannten *OLTP*[7]-Datenbankanwendungen der Fall.

   Dieses Speichermodell ist bei fast allen schema-behafteten und bei vielen semi-strukturierten Datenbanken die Grundlage der Datenspeicherung. So wird zum Beispiel bei relationalen Datenbanken das Datenbankschema genutzt, um den Zugriff auf Entitäten (Tupeln) effizient zu gestalten. Damit haben die Tupel sowie die darin enthaltenen Attribute eine feste Länge, da auch die verwendeten Datentypen eine feste Länge haben[8]. Mittels wahlfreiem Dateizugriff[9] und dem Lesen beziehungsweise Schreiben von Vielfachen der Tupellänge ist der Zugriff auf die Tupel unaufwändig möglich.

   Nach *Vossen* wird dieses Speichermodell auch als *normalisiertes Speichermodell mit horizontaler Partitionierung* bezeichnet [Vos94, S.415]. Dabei wird versucht, anhand des Schemas eine Normalisierung der Entitäten vorzunehmen. Bei relationaler Datenbanken entspricht das der ersten Normalform.

---

7 *OLTP* ist eine Abkürzung für *Online Transaction Processing*. Dieser Begriff wird in Textabschnitt 3.1 näher erläutert.

8 Es existieren auch Datentypen, die eine variable Länge besitzen. Im *SQL*-Standard sind zum Beispiel die Datentypen `VARCHAR`, `TEXT` und `BLOB` definiert. In relationalen Datenbanksystemen wird der Inhalt solcher Datentypen von den Datenbankdateien getrennt gespeichert. Innerhalb der Datenbankdateien wird dann auf diese externen Dateien referenziert. Hier ergibt sich auch der Vorteil, dass zum Beispiel für den Datentyp `TEXT` mittels verlustfreier Kompression Speicherplatz auf den Massenspeicher gespart werden kann.

9 Der wahlfreie Zugriff auf eine Datei besagt, dass zu jeder Datei ein Zeiger existiert, der die Position innerhalb der Datei für den nächsten Zugriff (lesend oder schreibend) bestimmt. Dieser Zeiger kann wahlfrei vor- oder rückwärts verschoben werden.

In Abbildung 2.2 ist das zeilen-basierte Speichermodell am Beispiel von *Postgresql* dargestellt. Für das oben beschriebene Datenbankbeispiel (1 Relation, 3 Entitäten, 3 Attribute) wird eine Datei genutzt, die in feste Größen eingeteilt ist (Seite mit 8 kByte Speicherplatz). Jede Seite enthält am Anfang einen festen Abschnitt mit Metainformationen über die Seite: die Seitennummer, die Anzahl der gespeicherten Tupel in der Seite und der noch freie Speicherplatz. In Abbildung 2.2 sind diese Metadaten als roter Block gekennzeichnet.

Möchte man auf ein Tupel zugreifen, wird der sogenannte *Tupel-Identifikator* (TID) genutzt. Es ist eine Kombination aus der Nummer der Seite und dem Tupel, das auf dieser Seite adressiert werden soll. Dazu sei bemerkt, dass die Tupel bei *Postgresql* innerhalb der Seite in reverser Reihenfolge an das Seitenende ausgerichtet werden. Zusätzlich werden Tupel-Metainformationen mit fester Größe als Tupel-Deskriptoren gespeichert, die am Seitenanfang in natürlicher Reihenfolge ausgerichtet werden. Diese Metainformationen speichern unter anderem den Startpunkt und die Länge des jeweiligen Tupels innerhalb der Seite. In Abbildung 2.2 sind die Metainformationsblöcke der drei gespeicherten Tupel in gelb dargestellt. Sie verweisen auf die Tupel, deren Attribute jeweils mit den Farben grün, gelb und blau dargestellt sind. Anhand der Verweise erkennt man auch die reverse Speicherung der Tupel.

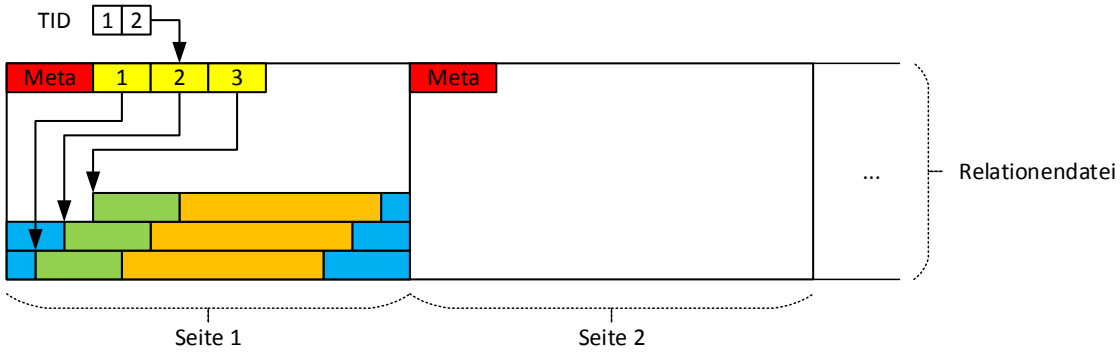


Abbildung 2.2: Zeilen-basiertes Speichermodell am Beispiel des Dateilayouts von *Postgresql*. Die Relationendatei wird in Segmente fester Größe geteilt (Seiten). Jedes Segment beginnt mit beschreibenden Metadaten (rot) und Tupel-Deskriptoren (gelb), die auf die Startposition der Tupel verweisen. Die Tupel bestehen aus Attributen (grün, orange und blau).

Mittels der TID ist somit der Zugriff auf gespeicherte Tupel sehr leicht möglich, indem die referenzierte Seite des TIDs in den Arbeitsspeicher gelesen wird. Anhand der Tupel-Deskriptoren kann dann der Startpunkt und die Gesamtlänge des adressierten Tupel ermittelt werden. Durch das Schema ist dann den Zugriff auf die Attribute des Tupels möglich. Nach dem Zugriff kann die gesamte Seite wieder auf den Primär- oder Sekundärspeicher zurückgesichert werden.

2. **Spalten-basiertes Speichermodell** Die Attribute einer Entität werden spaltenweise gespeichert. Für relationale Datenbanken bedeutet es, dass pro Attribut einer Relation eine Datei genutzt wird. Die Attributwerte werden dann aufeinanderfolgend in die jeweiligen Attributdateien geschrieben. Auch hier ergeben sich die Vorteile des wahlfreiem Zugriff, um auf ein oder mehrere aufeinanderfolgende Attributwerte lesend oder schreibend zuzugreifen. Dieses Speichermodell kann unter anderem immer dann genutzt werden, wenn

vor allem analytisch auf die Attribute zugegriffen werden soll, was zumeist bei sogenannten *OLAP*-Datenbankzugriffen[10] das Ziel ist. Das ist zum Beispiel der Fall, wenn die Summe oder der Durchschnitt der Werte eines Attributes ermittelt werden soll. Beim zeilenbasierten Speichermodell kann erst auf den Attributwert zugegriffen werden, nachdem das gesamte Tupel gelesen wurde. Für das Endergebnis stellt das eine gewisse Redundanz und Mehraufwand dar. Beim Spalten-basierten Speichermodell treten diese Aspekte nicht auf, da der gesamte Inhalt der Datei redundanzfrei zum Endergebnis beiträgt.

Dieses Speichermodell kann prinzipiell für alle Arten von Datenmodellen genutzt werden. Bei schema-behafteten und semi-strukturierten Datenmodellen ergibt sich der Vorteil, dass das Schema genutzt werden kann, um die Attribute innerhalb einer Datei zu adressieren, sofern die Attributwerte eine feste Länge besitzen. In diesem Fall entspricht dieses Speichermodell einer sequentiellen Speicherung in aufsteigender Tupelreihenfolge. *Vossen* bezeichnet dieses Speichermodell auch als *normalisiertes Speichermodell mit vertikaler Partitionierung* [Vos94, S.415].

In Abbildung 2.3 ist für das oben genannte Beispiel das spalten-basierte Speichermodell abgebildet. Die einzige Relation der relationalen Datenbank besitzt drei Entitäten (Tupel), die jeweils drei Attribute besitzen. Dementsprechend wird pro Attribut eine Datei zur Speicherung verwendet. Auch hier werden die Attributdateien in Seiten fester Größe eingeteilt, wobei Metainformationen pro Seite am Seitenanfang gespeichert werden (in Abbildung 2.3 in rot gekennzeichnet).

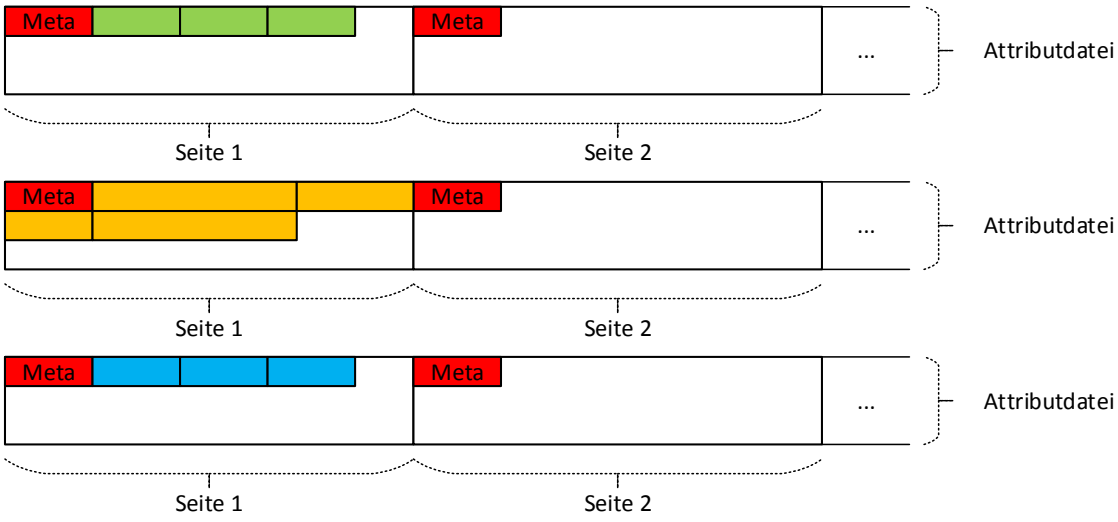


Abbildung 2.3: Spalten-basiertes Speichermodell: eine Relation nutzt pro Tupelattribut eine Datei zur sequentiellen Speicherung der Attributwerte (grün, gelb und blau gekennzeichnet), wobei die Attributdateien in Seiten fester Größe eingeteilt werden. Am Anfang jeder Seite werden beschreibende Metainformationen über die Seite gespeichert (rot).

Auf Abbildung 2.3 erkennt man auch die sequentielle Speicherung der Attributwerte in den einzelnen Attributdateien. Hier ergeben sich die beschriebenen Vorteile des Zugriffes

[10] Abkürzung für *Online Analytical Processing*. Dieser Begriff wird in Textabschnitt 3.1 näher erläutert.

auf nur ein Attribut einer Relation. Wenn zum Beispiel angenommen wird, dass das zweite Attribut der Relation numerisch ist und nun dafür der Mittelwert gebildet werden soll, so muss nur die entsprechende Attributdatei seitenweise in den Arbeitsspeicher gelesen werden. Die Metainformationen am Anfang der Seite können übersprungen werden. Das die Attributwerte eine konstante Speicherlänge aufweisen, kann einfach über den Seiteninhalt iteriert werden, wobei der Mittelwert aktualisiert wird. Dieses Vorgehen ist insoweit gegenüber dem zeilen-basierten Speichermodell vom Vorteil, da zum Beispiel keine Tupel-Identifikatoren nötig sind, da sich die Adressierung implizit durch die Größe der Attributdatei, der Seitengröße, der Länge der Seiten-Metainformationen und der konstanten Länge des Attributes ergibt.

Ein weiterer Vorteil ist, dass bei einer Datenbankabfrage nur diejenigen Seiten in den Arbeitsspeicher geladen werden, die auch hinsichtlich des Abfrageergebnisses relevant ist. Sei zum Beispiel eine Datenbankabfrage gegeben, die das zweite Attribut aller Tupel der Beispielrelation abfragt. Das Abfrageergebnis soll also eine Liste mit den Attributwerten des zweiten Attributes sein. Beim zeilen-basierten Speichermodell müssen dazu alle Tupel gelesen werden, um auf den Inhalt des zweiten Attributes zuzugreifen. Da damit auch das erste und dritte Attribut gelesen werden, stellt das eine Redundanz dar. Beim spalten-orientierten Speichermodell muss nur die Attributdatei in den Arbeitsspeicher gelesen werden wie es oben beschrieben ist.

3. **Baumstrukturen** In diesem Speichermodell werden die Entitäten in einer Baumstruktur verwaltet, wobei die Attribute in den Blättern gespeichert werden. Hier wird das Ziel verfolgt, die Entitäten in einer gewissen Ordnung zu speichern.

   In der Praxis werden zumeist B- oder B^+-Bäume als Baumstruktur genutzt [KE11, S.220]. Am Beispiel von relationalen Datenbanken wird für die Persistierung der Tupel je eine Datei pro Relation genutzt. Außerdem kann in den meisten Fällen angenommen werden, dass sich die Baumstruktur anhand der natürlichen Ordnung des Primärattributes oder der Attributkombination ergibt. Ist das Primärattribut zum Beispiel numerisch, so ergibt sich im Falle der Nutzung eines B-Baumes eine Anordnung der Tupel in auf- beziehungsweise absteigender Reihenfolge gemäß des Primärattributwertes. Auf Blattebene sind die Tupel sozusagen geordnet. Der Vorteil der Nutzung von Baumstrukturen ist, dass die Entitäten bereits anhand der Ordnungsfunktion ihres Primärattributes geordnet sind, womit spezielle Zugriffsoperationen auf die Tupel möglich werden. Um auf das oben genannte Beispiel des B-Baumes mit numerischen Primärattribut einzugehen, so kann zum Beispiel die Suche nach Tupeln durch die Baumstruktur effektiv unterstützt werden, deren Primärattributwerte in einem bestimmten numerischen Bereich liegen [Vos94, S.407]. Dazu wird die Baumstruktur traversiert, bis der entsprechende Startpunkt auf Blattebene in der Datei erreicht ist, in der die Relation verwaltet wird. Danach können nachfolgende Tupel sequentiell eingelesen werden, bis das Ende des Suchbereiches erreicht wird.

   Das wird dadurch erreicht, dass die genutzte Datei zur leichteren Verwaltung in Seiten fester Größe eingeteilt werden, aber die Entitäten werden gemäß der Baumstruktur auf die Seiten verteilt, sodass sie laut ihres Primärattributes geordnet sind. Dabei wird auch versucht, dass die Seiten auch innerhalb der Datei aneinander grenzen. Wie bereits erwähnt, kann auch eine Attributkombination der Entitäten zur Anordnung herangezogen werden, sofern sich eine Ordnungsfunktion herausstellt. Zudem ist die Ordnungsfunktion nicht nur auf numerische Wertebereiche eingeschränkt. So ist es möglich, Primärattribute zu nutzen, die Zeichenketten speichern.

Insgesamt kann dieses Speichermodell auch als eine Art flexibleres Clustering angesehen werden. Somit werden Entitäten möglichst "nahe" aneinander gespeichert, wobei die "Nähe" durch eine Ordnungsfunktion hergestellt wird. Nachteilig an diesem Speichermodell ist, dass die notwendigen inneren Knoten der Baumstruktur in die Datei gespeichert werden müssen, was eine Redundanz darstellt. Zudem muss die Baumstruktur zusätzlich zu den Daten gepflegt werden, was einen gewissen Mehraufwand bedeutet, zum Beispiel CPU-Rechenzeit und erhöhter Arbeitsspeicheraufwand. Das trifft zum Beispiel zu, wenn neue Entitäten gespeichert, existierende gelöscht oder der Wert des Primärattributes einer Entität geändert wird. Auf Blattebene werden dadurch neue Knoten erzeugt, bestehende gelöscht oder aufgrund der Ordnungsfunktion und des Clusterings verschoben. Das hat zur Folge, dass auch die innere Knoten der Baumstruktur aktualisiert werden müssen, wobei auch hier unter Umständen neue innere Knoten erzeugt oder bestehende gelöscht werden müssen. Diese Aktualisierung kann sich bis zur Wurzel der Baumstruktur fortsetzen.

4. **Hashing** Dieses Speichermodell verteilt die Entitäten über mehrere Speicherorte. Das vorhergehende Speichermodell, in der Baumstrukturen genutzt wurden, hat einen vergleichbaren Ansatz. Allerdings muss vor dem Zugriff auf die gewünschten Entitäten zuerst die Baumstruktur traversiert werden. Um diesen Mehraufwand zu vermeiden, können die Entitäten mittels Hash-Techniken verteilt werden. Im Gegensatz zu Baumstrukturen wird die Adressierung einer Entität durch Berechnung aus einem Primärattribut oder einer Attributkombination erreicht [Vos94, S.409].

   Formal ist eine Hash-Funktion h eine Streuwert-Funktion der Form

   $$h : D \rightarrow Z$$

   wobei D die Definitionsmenge und Z die Zielmenge sei. Im allgemeinen gilt, das D deutlich größer ausfällt als Z ($|D| \gg |Z|$). Nach *Vossen* sowie *Kemper und Eickler* ist h surjektiv und zumeist nicht injektiv ([Vos94, S.410],[KE11, S.223]). Das bedeutet dass für zwei Elemente $a, b \in D$ unter der Bedingung von $h(a) = h(b)$ nicht notwendigerweise $a = b$ gilt. Dennoch kann es vorkommen, dass der Fall $h(a) = h(b)$ für die beiden Elemente $a, b \in D$ eintritt. Er wird als Kollision bezeichnet.

   Hash-Funktionen werden als Speichermodell für zwei Aufgabenbereiche eingesetzt: zur Adressierung einer Entität innerhalb einer oder mehreren lokalen Dateien oder zur deren dezentralen Adressierung. Zur Erläuterung der erstgenannten Art kann wieder das oben beschriebene Beispiel der relationalen Datenbank herangezogen werden (eine Relation mit drei Entitäten (Tupeln) und jeweils drei Attributen). Dabei wird angenommen, dass das erste Attribut das Primärattribut A sein soll, dessen numerischer Wertebereich sich von 0 bis 1000 erstreckt ($dom(A) = \{0, 1, ..., 1000\}$). Weiterhin wird angenommen, dass die Tupel der Relation in nur einer Relationendatei verwaltet werden, wobei das zeilen-basierte Speichermodell zum Einsatz kommt. Es kann nun eine Hash-Funktion h_1 genutzt werden, die aus A direkt die entsprechende Seite innerhalb der verwendeten Relationendatei errechnet. Damit kann zum Beispiel zielgerichtet die errechnete Seite in den Arbeitsspeicher gelesen werden, um dann Zugriff auf das gewünschte Tupel zu nehmen. Die Hash-Funktion h_1 ist demnach nicht kollisionsfrei. Generell kann man davon ausgehen, dass die Relationendatei nicht unendlich viele Seiten besitzt. Wenn also S die Menge aller Seiten ist, dann ist $h_1 : A \rightarrow S$ beziehungsweise $h_1 : dom(A) \rightarrow dom(S)$. Problematisch hieran ist, dass auch die Seiten nicht unendlich Speicherkapazität haben, um alle Tupel zu speichern.

   Idealerweise kann auch eine Hash-Funktion h_2 eingesetzt werden, die zusätzlich zur Seitennummer noch den genauen Speicherplatz eines Tupels errechnen kann. Mit anderen

Worten: aus dem Primärattributwert eines Tupel kann der Tupel-Identifikator direkt durch h_2 errechnet werden. h_2 muss in diesem Fall kollisionsfrei sein. Das ist in der Praxis kaum möglich. Bezogen auf die obige Definition von h_1 kann h_2 als Funktion $h_2 : \mathit{dom}(A) \rightarrow T$ definiert werden. T sei hierbei die Menge aller möglichen Tupel-Identifikatoren. Das heisst, T ist die Menge aller möglichen Kombinationen aus S und einem endlichen $n \in \mathbb{N}$, das die Maximalanzahl an Tupeln pro Seite angibt.

Wie bereits erwähnt, ist der zweite Aufgabenbereich von Hash-Funktionen die dezentrale Adressierung. Dabei werden mehrere Dateien zur Speicherung von Tupeln einer Relation genutzt, die durchaus dezentral verteilt sind. Dementsprechend wird eine Hash-Funktion h_3 definiert, die den Wertebereich des Primärattributes A in mehrere gleich große Segmente aufteilt. Pro Segment wird eine Datei zur Verwaltung der Tupel genutzt. Sei R die Menge dieser Dateien. h_3 erweitert die Hash-Funktion h_1 insofern, dass die Zielmenge eine Kombination aus R und S ist, also $h_3 : A \rightarrow R \times S$ beziehungsweise $h_3 : \mathit{dom}(A) \rightarrow \mathit{dom}(R \times S)$. So sind mehrere Szenarien möglich: einzelne Dateien werden durch die Hash-Funktion so genutzt, dass die meistgenutzten Tupel in einer Datei verwaltet werden, die auf einem Speicherort liegt, der besonders performant hinsichtlich des Datendurchsatzes ist. Die restlichen Dateien können sich auf weniger performanten Speicherorten befinden. Zudem ist auch eine dezentrale Dateiorganisation nicht ausgeschlossen. Dabei befinden sich die Dateien auf verschiedenen Informationssystemen und bilden dabei eine *verteilte Datenbank*.

Dieses Verfahren kann insoweit ausgebaut werden, dass zum Beispiel die Anzahl der beteiligten Dateien variabel gehalten werden kann. Es kann also der Fall auftreten, dass neue Dateien zur Verwaltung der Tupel hinzukommen oder bestehende Dateien wegfallen. Wenn ein solches Verhalten berücksichtigt werden muss, dann müssen unter Umständen Tupel umverteilt werden, da sich die Werte der Hash-Funktion ändern und somit auch der Speicherplatz der Tupel angepasst werden muss. Um die notwendigen Umverteilungen von Tupeln auf ein Minimum zu beschränken, werden sogenannte konsistente Hash-Funktionen eingesetzt [KK03]. Sie bilden die Grundlage für die effiziente, dezentrale Speicherung von Daten in Datenbanksystemen, zum Beispiel bei *Amazon Dynamo* und *Cassandra* [DHJ+07, LP10a].

Verteilte Hash-Tabellen hingegen kommen zum Einsatz, wenn das Hinzukommen und der Wegfall von Speicherorten sehr dynamisch ist. Nach *Karger et al.* wird zusätzlich nicht nur von einer globalen Hashtabelle ausgegangen, sondern von multiplen [KLL+97]. Bezogen auf die obige formale Definition einer Hash-Funktion entsteht eine Hashtabelle dadurch, dass für alle möglichen Elemente $d \in D$ deren Hashwerte, also die Funktionswerte von $h(d) \in Z$ berechnet werden. Sofern Z konstant verbleibt, so ist auch die Hashtabelle konstant. Ist Z dagegen dynamisch, so ist die Zuordnung einer Entität gemäß ihres Primärattributes zum Speicherort auch dynamisch. Um Ausfällen entgegenzuwirken, werden Entitäten zumeist über mehrere Speicherorte verteilt, dass heisst, sie werden repliziert (vgl. dazu [NW03]). Bei verteilten Hashtabellen pflegt jeder Speicherort seine eigene Hashtabelle. Über geeignete Synchronisations- und Beitrittsprotokolle wird sichergestellt, dass die verteilten Hashtabellen aktualisiert werden, sobald ein Speicherort hinzukommt oder wegfällt (vgl. dazu [KK03] und insbesondere *Kemper und Eickler* in [KE11, S.509 ff.]).

Neben diesen vier Speichermodellen kann noch ein weiteres hinzugefügt werden, wobei die Literatur noch darüber uneinig ist. Beim sogenannten *In-Memory*-Speichermodell wird die gesamte Datenbank im Hauptspeicher gespeichert und verwaltet. *Vossen* argumentiert zum Beispiel, dass dies keine neue Form eines Speichermodells darstellt, sondern nur eine andere Art der Organisation [Vos94, S.624]. Dementgegen argumentieren *Kemper und Eickler*, dass sehr viele neue

Konzepte und Technologien eingeführt wurden, sodass man von einem neuen Speichermodell ausgehen kann, das von anderen klar abgegrenzt ist [KE11, S.668 ff.].

Dabei wird der Fakt genutzt, dass der Zugriff auf den Arbeitsspeicher wesentlich schneller im Vergleich zu Massenspeichern wie Festplatten ist. Zudem müssen bestimmte Einschränkungen nicht mehr berücksichtigt werden. Dazu gehört zum Beispiel der Zugriffsschutz von Dateien im Mehrbenutzerbetrieb oder die Cache-Kohärenz von Dateiinhalten [SMA+07]. Eine Studie von *Harizopoulos et al.* [HAMS08] zufolge erzeugen Datenbanken bis zu 93 Prozent Mehraufwand für diese Einschränkungen. Die einzelnen Anteile sind in Abbildung 2.4 dargestellt.

Diese Studie maß, wieviele CPU-Instruktionen ausgeführt werden mussten, um eine Transaktion auf der Datenbank auszuführen. Dabei zeigte sich, dass klassische Datenbanksysteme, die Primär- und Sekundärspeicher für die Verwaltung der Datenbank verwandten, einen sehr signifikaten Anteil an Instruktionen aufwandten, um zum Beispiel die Konsistenz der Datenbank im Mehrbenutzerbetrieb (circa 30 Prozent) oder die Cache-Kohärenz sicherzustellen (circa 35 Prozent). Dieser Mehraufwand tritt bei *In-Memory*-Datenbanksystemen nicht auf.

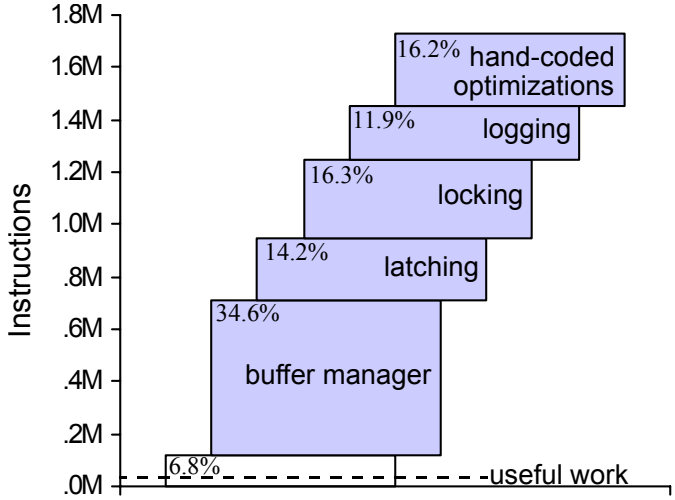


Abbildung 2.4: Mehraufwand eines klassischen Datenbanksystems mit Primär- und Sekundärspeicher für die Verarbeitung einer Transaktion gegenüber eines In-Memory-Datenbanksystems nach [HAMS08]

In der Regel werden dennoch Zeilen- oder Spalten-basierte Speichermodelle im Arbeitsspeicher genutzt, um die Entitäten einer Datenbank zu organisieren. Man kann annehmen, dass durch den vergleichsweise großen Datendurchsatz und die sehr schnelle Zugriffsgeschwindigkeit des Arbeitsspeichers das Speichermodell irrelevant wird. Nach einer Studie von *Wöltche* sind dementgegen durchaus Unterschiede messbar [Wöl13]. Sie basieren darauf, in welcher Weise die Betriebssysteme Segmente des Arbeitsspeichers allokieren und in welcher Form Bereiche dieser Segmente iteriert werden. Dennoch ist nach *Kemper und Eickler* der Datendurchsatz und die Zugriffsgeschwindigkeit des Arbeitsspeichers so groß, dass die Vorteile der beiden genannten Speichermodelle relativiert werden [KE11, S.669].

Das hat zur Folge, dass sowohl *OLTP*- als auch *OLAP*-zentrischen Datenbankanwendungen auf die Datenbank zuzugreifen, ohne Geschwindigkeitsnachteile befürchten zu müssen.

Der Nachteil des Einsatzes von Arbeitsspeicher ist, dass er im Bezug auf die Speicherkapazität ungleich kleiner als Massenspeicher ist. Daher kommen auch spezialisierte Zeilen- oder Spalten-basierte Speichermodelle zum Einsatz, die versuchen, den Verbrauch an Arbeitsspeicher so klein wie möglich zu halten, zum Beispiel durch effiziente Kompressionsalgorithmen [Col15, S.8][CGK10]. Ein weiterer Nachteil ist, dass Arbeitsspeicher im Gegensatz zu Massenspeicher eine weitaus höhere Preis-zu-Speicherkapazität-Relation aufweist.

Am Beispiel einer relationale Datenbank kann man sich durchaus einen Kompromiss vorstellen: die Relationen der Datenbank werden Zeilen-basiert in regulären Dateien verwaltet, aber bestimmte Attribute werden zusätzlich Spalten-basiert im Hauptspeicher gespeichert, um zum Beispiel den analytischen Zugriff auf diese Attribute zu beschleunigen.

2.3 Datenbankmanagementsysteme

Der Begriff *Datenbankmanagementsystem* ist in der Literatur nicht eindeutig definiert. Nach *Vossen* [Vos94, S.8]

> *stellt ein Datenbankmanagementsystem eine Kontrollinstanz (in Form eines Software-Paketes) dar, über welche Anwendungsprogramme ("Applikationen") sowie Dialogbenutzer auf eine große Datensammlung, die Datenbank, zugreifen können; es ist im allgemeinen in der Lage, mehrere Datenbanken gleichzeitig zu verwalten.*

Nach *Elmasri und Navathe* ist ein Datenbankmanagementsystem [EN09, S.19]

> *ein Softwaresystem (Sammlung von Programmen), das dem Benutzer das Erstellen und die Pflege einer Datenbank ermöglicht. Das Datenbankmanagementsystem ist folglich ein generelles Softwaresystem, das die Prozesse der Definition, Konstruktion und Manipulation von Datenbanken für verschiedene Anwendungen vereinfacht.*

Beide Definitionen zeigen Gemeinsamkeiten. So greifen Datenbankanwendungen zumeist in Form von Datenbankabfragen auf ein oder mehrere Datenbanken zu. Der Zugriff beschränkt sich dabei nicht nur auf die reine Abfrage von Daten, sondern auch auf deren Manipulation sowie beispielsweise auf die Manipulation des Schemas von Datenbanken. Wie in Abbildung 2.5 auf Seite 25 dargestellt, definieren *Elmasri und Navathe* zwei wesentliche Komponenten in einem Datenbankmanagementsystem. Die erste Komponente nimmt die Datenbankabfragen der Datenbankanwendungen entgegen. Sie werden validiert, optimiert und resultieren in einer Serie von Datenbankoperationen, die von der zweiten Komponente verarbeitet und ausgeführt werden. Diese Datenbankoperationen greifen dabei auf die eigentlichen Datenbanken zu.

Abstrahiert man die Ablauf eines Zugriffs innerhalb eines Datenbankmanagementsystems, so zeigt sich, dass sich die grundlegenden Komponenten kaum geändert haben. Nach *Elmasri und Navathe*, *Geisler* sowie *Vossen* ([EN09, Gei14, Vos94]) existieren vier Verarbeitungskomponenten: die Zugangskontrolle, die Erstellung des Zugriffsplanes, die Transaktionskontrolle und die Ausführung des Zugriffsplanes. In Abbildung 2.6 auf Seite 26 ist die Verarbeitung einer Datenbankabfrage anhand dieser vier Komponenten als Programmablaufplan dargestellt. Sie sind in dieser Abbildung vertikal angeordnet und sind auf der linken Seite erkennbar.

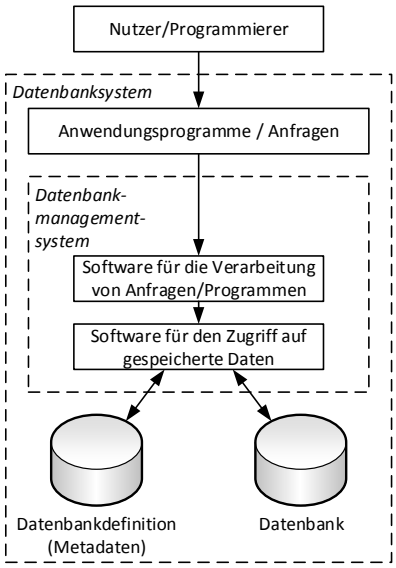


Abbildung 2.5: Zusammenhang zwischen Datenbanken, Datenbankmanagementsysteme und Datenbanksystemen nach [EN09]

Dabei verläuft die Verarbeitung einer Datenbankabfrage wie folgt ab:

1. **Zugangskontrolle** Der Anfragende wird authentifiziert und authorisiert. Die Authentifizierung bedeutet, dass der Anfragende identifiziert wird und seinen Zugriff legalisiert. Dies wird zumeist durch die Angabe eines Benutzernamen-Passwort-Paares, einem eindeutigen kryptografischen Schlüssel oder durch eine eindeutige IP-Addresse beziehungsweise Subnetzmaske erreicht. In Abbildung 2.6 ist zur Authentifizierung eine Benutzerkonten-Datenbank dargestellt, die die Benutzernamen-Passwort-Paare enthält. Nach der Authentifizierung wird durch den Vorgang der Authorisierung geprüft, ob der Anfragegende berechtigt ist, auf die Objekte innerhalb der Anfrage zuzugreifen. Dies wird im allgemeinen durch eine Zugriffsliste erreicht, in der tabellarisch den Benutzerkonten die Zugriffsrechte auf die Datenbankobjekte verzeichnet sind. In Abbildung 2.6 ist die Zugriffsliste als Datenfeld “ACL” erkennbar (engl. *Access Control List*, Zugriffsliste).

2. **Erstellung des Zugriffssplanes** Die Anfrage wird syntaktisch geprüft und ein Zugriffsplan erstellt. Als erster Schritt werden dabei die Elemente der Anfrage gegenüber dem Syntax der Zugriffssprache geprüft, zum Beispiel *SQL*. Im Fehlerfall wird die Verarbeitung der Anfrage in dieser Komponente bereits abgebrochen und ein Fehler an den Anfragenden übermittelt. Danach wird der Zugriffsplan erstellt. Der Zugriffsplan ist eine Serie von Lese- und Schreiboperationen auf die Datenbankdateien, um auf Basis der Anfrage die gewünschten Daten zu ermitteln oder zu modifizieren. Diese Serie wird als *Transaktion* bezeichnet, da das *ACID*-Paradigma eingehalten werden muss. Das bedeutet, dass die Lese- und Schreiboperationen atomar, also ganz oder gar nicht, ausgeführt werden. Im Falle von Schreiboperationen muss sichergestellt werden, dass die Konsistenz der gesamten Datenmenge nicht beeinträchtigt wird und die Schreiboperationen dauerhaft, also nicht-flüchtig, gespeichert werden.

Die Transaktion kann dadurch optimiert werden, dass Technologien genutzt werden, die die Anzahl der Schreib- und Leseoperationen vermindern. Am bekanntesten sind zum Beispiel die Datenbankindizes. Das sind Strukturen, die extern zu den Datenbankdateien gespeichert werden. Sie speichern Zeiger auf die eigentlichen Daten innerhalb der Datenbankdateien und genügen dabei einer gewissen Ordnung von einer oder mehreren Spalten.

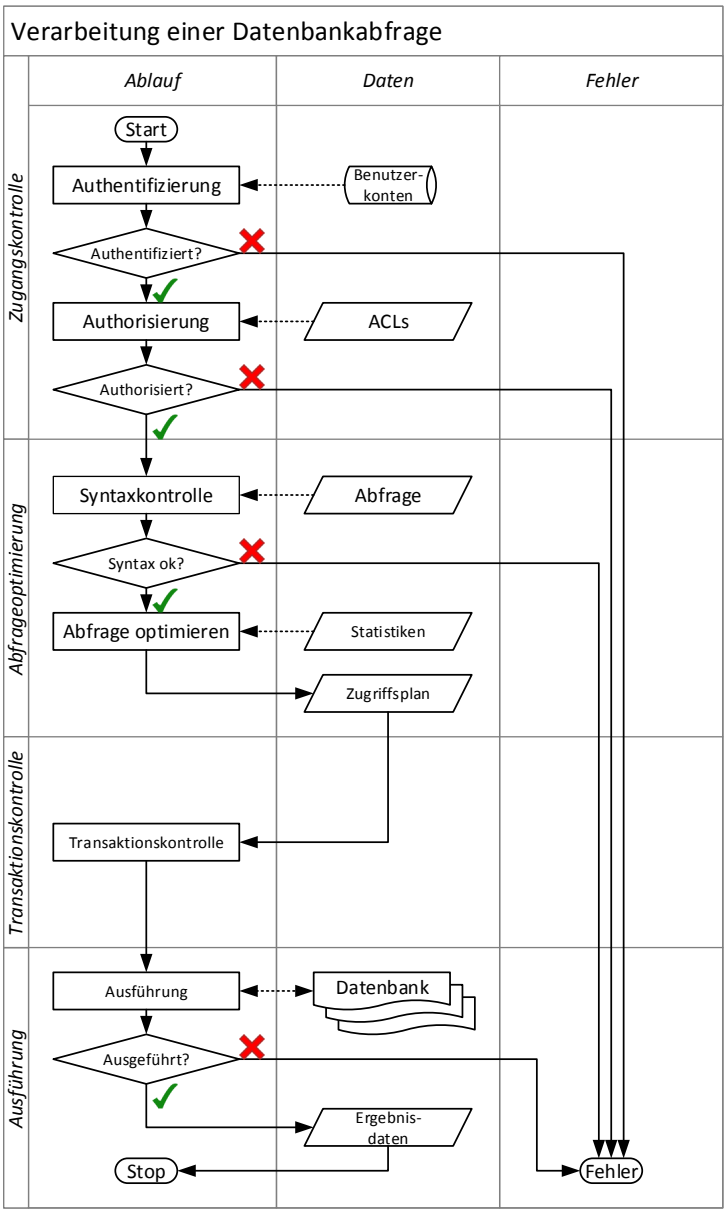


Abbildung 2.6: Ausführung einer Datenbankabfrage

Ein abstraktes Beispiel für einen Index wäre das Inhaltsverzeichnis dieser Dissertation. Wenn jede Seite ein Datensatz wäre, der durch die Seitenzahl eindeutig auffindbar ist, dann

beschleunigt das Inhaltsverzeichnis die Suche nach bestimmten Überschriften eines Textabschnittes. Ohne das Inhaltsverzeichnis muss jede Seite nach der gesuchten Überschrift durchsucht werden, was einer *sequentiellen Suche* entspricht. Die zeitliche Komplexität wäre hierbei nach der *Landau*-Notation $O(n)$, wobei n die Anzahl der Datensätze angibt und damit linear mit n ansteigt. Indizes bieten hier weitaus bessere zeitliche Komplexitäten. Bekannte Indexarten sind zum Beispiel die B- und R-Bäume sowie invertierte Indizes. Indizes können durchaus als eine Datenredundanz angesehen werden, aber sie beschleunigen die Suche und das Sortieren von Datensätzen.

Je nachdem, wie komplex die Abfrage ist, kann es bei dessen Auswertung durchaus zur Erstellung verschiedener Zugriffspläne kommen. Die Aufgabe der Abfrageoptimierung ist es, den effizientesten daraus auszuwählen. Effizient bedeutet, dass eine Abschätzung zwischen dem Zeitaufwand und den Kosten der Erfüllung der Abfrage gemacht werden muss. Das Ziel ist es aber immer, die Antwortzeit zu minimieren. In der Praxis kommen hierfür Kosten- und Regel-basierte Abschätzungen zum Einsatz. Bei ersterem werden zum Beispiel für jede Nutzung und Pflege eines Indizes, dem Lesen oder Schreiben eines Datensatzes oder dem Sortieren der Ergebnismenge gewisse Kosten zugeordnet. Diese Kosten können zum Beispiel Rechenzeit oder Speicherbedarf sein. Somit stellt dieses Verfahren eine heuristische Aufwandsabschätzung dar und nutzt dafür Statistiken, die für die Datenbankobjekte erhoben und periodisch aktualisiert werden. In Abbildung 2.6 sind diese Statistiken als Datenfeld “Statistiken” gekennzeichnet. Die Transaktion mit den geringsten Kosten wird zur Ausführung ausgewählt. Das führt manchmal dazu, dass zum Beispiel das sequentielle Suchen nach Datensätzen effizienter ist als die Nutzung eines Index. Dies ist zum Beispiel der Fall, wenn die Datenbank recht klein ist und die Kosten der Nutzung eines Index (Einlesen des Index in den Hauptspeicher, zusätzlicher Rechenaufwand für den Indexalgorithmus) größer sind als das simple, sequentielle Lesen der Datenbankdateien.

Bei der Regel-basierten Abschätzung werden indessen keinerlei Heuristiken und nur zum Teil Statistiken genutzt. Sie entscheiden nach einer Liste mit festgelegten Regeln. Eine Regel kann zum Beispiel lauten, dass ein Zugriffsplan effizienter als ein anderer ist, wenn ein Index existiert und genutzt werden kann. Eine zweite Regel kann besagen, dass die Reihenfolge der Datenbanktabellen in einem Kreuzprodukt die Effizienz festlegt. Der Anfragende hat also einen direkten Einfluss auf die Wahl des Zugriffsplanes. Der Vorteil von Regel-basierten Abschätzungen ist, dass zumeist nur sehr wenige Statistiken geführt und periodische aktualisiert werden müssen. Der Nachteil ist, dass die Auswahl des Zugriffsplanes zum Teil statisch ist und nicht auf der tatsächlichen Datenbankgröße, also der Datenmenge, basiert.

Eine weitere Aufgabe der Anfrageoptimierung ist auch, redundante Abfragekriterien bereits bei der Analyse zu verwerfen. Ein Beispiel ist die Suche nach Datensätzen, die eine ganzzahligen Inhalt x aufweisen. Wenn die Suche beispielsweise aus zwei Suchkriterien $x < 4 \wedge x < 7$ besteht, so ist das erste Suchkriterium redundant, da das zweite das erste bereits beinhaltet. Solche redundanten Suchkriterien bedeuten zumeist, dass der Zugriffsplan dieselben Lese- und Schreiboperationen aufweist. Das ist unnötig, um die mögliche Ergebnismenge zu bestimmen.

3. **Transaktionskontrolle** Die konkurrierenden, gleichzeitig ausführbaren Zugriffspläne werden vor ihrer Ausführung so angeordnet, dass sie sich möglichst nicht überschneiden. So können Zugriffspläne, die nur lesend auf den Datenbestand zugreifen, durchaus auch parallel ausgeführt werden, da sie ihn nicht verändern. Zugriffsplänen, die auch Schreiboperationen beinhalten, müssen unter Umständen sequentiell angeordnet und ausgeführt werden, um das *ACID*-Paradigma nicht zu verletzen [Vos94, S.453]. Insgesamt zeigt die

Transaktionskontrolle einige Analogien zum sogenannten *process scheduling* der Betriebssysteme, also das Planen, Verwalten und Ausführen von Prozessen. So wurden einige Algorithmen publiziert, die zum Beispiel die Schreib- und Leseoperationen der Zugriffspläne analysieren und sie dergestalt anordnen, dass diejenigen priorisiert werden, deren "Verfallsdatum" am nächsten ist. Dieser *Earliest-deadline-first*-Algorithmus wird zum Beispiel in Echtzeit-Datenbankmanagementsystemen eingesetzt [SDM+89, UB93]. Es sind aber auch andere Algorithmen möglich, zum Beispiel durch die Vergabe von Prioritäten, sodass die Ablaufpläne von bestimmten Benutzern eines Datenbankmanagementsystems bevorzugt vor anderen ausgeführt werden. Allen Algorithmen ist jedoch gemein, dass sie die Zugriffspläne *serialisieren* müssen [Vos94, S.458 ff.].

Nach *Vossen* kann die Transaktionskontrolle agressiv oder konservativ vorgehen [Vos94, S.474]. Das erstgenannte Vorgehen nutzt dabei Sperrmechanismen, um den Zugriff auf die Datenbankobjekte zu synchronisieren, etwa durch sogenannte *locks* (engl. für Sperre). Das meistgenutzte Protokoll hierbei ist nach *Vossen* das *2-Phasen-Sperrprotokoll* und dessen Varianten [Vos94, S.477]. Das letztgenannte Vorgehen vermeidet Sperren, um den Zugriff auf die Datenbankobjekte zu synchronisieren. Dazu gehören nach *Vossen* das sogenannte *Zeitmarken-Verfahren* und das sogenannte *Serialisationsgraphen-Tester*-Verfahren [Vos94, S.489]. Ersteres benutzt Zeitstempel, um die Ausführung von Zugriffsplänen zu ordnen und insofern anzuordnen, dass sie konfliktfrei ausgeführt werden können. Letzteres nutzt Methoden aus der Graphentheorie, um konkurrierende Zugriffspläne in einer Weise anzuordnen, dass sie konfliktfrei sind.

Es sei bemerkt, dass die eingesetzten Verfahren bei der Transaktionskontrolle nicht trivial sind. Deren Erläuterung würde den Rahmen dieser Dissertation sprengen. Dementsprechend sei insbesondere auf *Vossen* verwiesen, der detailliert alle Verfahren und Methoden zu den Themen Serialisierbarkeit und Synchronisation von Zugriffsplänen erläutert und formal beweist [Vos94, S.443 ff.].

4. **Ausführung des Zugriffsplanes** Der nächste Zugriffsplan, der von der Transaktionskontrolle zur Ausführung bestimmt wurde, wird ausgeführt. Das bedeutet, dass die Schreib- und Leseoperationen im Zugriffsplan ausgeführt werden. Bei Schreiboperationen werden die Datenbankdateien je nach Konfiguration sofort auf den Massenspeicher persistiert oder zunächst gepuffert, um sie nachfolgend periodisch zu persistieren. In Abbildung 2.6 werden die Datenbankdateien als Dateienstapel mit der Bezeichnung "Datenbank" gekennzeichnet. Sofern nur Schreiboperationen ausgeführt werden, wird auch keine Ergebnismenge gebildet. Sind Leseoperationen beteiligt, beinhaltet die Ergebnismenge diejenige Daten, die von der Anfrage selektiert wurden. Tritt ein Fehler bei den Schreib- und Leseoperationen auf, zum Beispiel aufgrund eines vollen Massenspeichers, bricht der komplette Vorgang ab und ein Fehler wird dem Aufrufenden zurückgegeben. Im Erfolgsfall erhält der Aufrufende die eventuelle Ergebnismenge.

2.4 Datenbanksysteme

Analog zur Definition eines Datenbankmanagementsystems ist auch der Begriff eines Datenbanksystems in der Literatur uneinheitlich definiert. Nach *Vossen* ist ein Datenbanksystem [Vos94, S.30]

> *stets die Kombination eines DBMS mit einer (oder sogar mehreren, unterscheidbaren) Datenbanken [...], d.h. kurz*
>
> $$DBS = n \times DB + DBMS, n \geq 1$$

Eine fast gleichlautende Definition nutzen auch *Elmasri und Navathe* [EN09, S.19]. Deren grafische Darstellung des Zusammenhanges zwischen Datenbanken, Datenbankmanagementsystem und Datenbanksystem ist in Abbildung 2.5 auf Seite 25 dargestellt.

Im Gegensatz dazu sind beide Begriffe Datenbankmanagementsystem und Datenbanksystem nach *Kemper und Eickler* gleichbedeutend [KE11, S.28]. Im Rahmen dieser Dissertation wird aber den Definitionen von *Vossen* sowie *Elmasri und Navathe* gefolgt.

Demnach ist ein Datenbanksystem eine Computersoftware, die eine Schnittstelle für Anwendungsprogramme und Anfragen bereitstellt, um die Datenbasis in Form einer Datenbank zu manipulieren und abzufragen. Das enthaltene Datenbankmanagementsystem speichert und verarbeitet dabei Informationen beziehungsweise Fakten einer Anwendungsdomäne und stellt sie untereinander in Verbindung. Dabei wird zumeist das sogenannte *ACID*-Paradigma verfolgt [Vos94, S.449]:

- Abgeschlossenheit (*Atomacity*), dass heisst die Datenbankoperationen werden als Transaktion entweder ganz oder gar nicht ausgeführt. Wird eine Transaktion erfolgreich ausgeführt, sind deren Auswirkungen auf die Datenbank auch für parallel agierende andere Datenbankanwendungen sichtbar. Bei Nichterfolg erscheint die Datenbank so, als wenn die Transaktion nie ausgeführt worden.
- Konsistenz (*Consistency*), dass heisst nach jeder erfolgreich ausgeführten Transaktion sind alle gespeicherten Informationen in einem konsistenten Zustand. Das betrifft vor allem die referentielle Integrität, wenn die Informationen untereinander verbunden sind. Bei Nichterfolg einer Transaktion wird die Datenbank wieder in einen konsistenten Zustand gebracht.
- Isolation, dass heisst nebenläufige Transaktionen werden voneinander isoliert und beeinflussen sich somit nicht. Das bedeutet ferner, dass Datenbankanwendungen voneinander unabhängig agieren können und ihnen durch die beiden vorherigen Punkte nur konsistente Daten zur Verfügung gestellt werden.
- Dauerhaftigkeit beziehungsweise Persistenz (*Durability*), dass heisst die Informationen einer Transaktion müssen nach dessen erfolgreichen Abschluss dauerhaft gespeichert werden, um sie im Falle eines Systemfehlers wiederherstellen zu können. Genauer gesagt, sofern ein Datenbankmanagementsystem einer Datenbankanwendung meldet, dass eine Transaktion erfolgreich ausgeführt wurde und danach ein Hard- oder Softwarefehler auftritt, so werden alle Effekte der Transaktion wiederhergestellt. Das gilt auch für alle etwaigen Transaktionsdaten, die sich vor dem Auftreten des Fehlers in diversen Puffern befanden.

Wie bereits in der Einleitung in Kapitel 1 dargestellt, hat sich in den letzten Jahren gezeigt, dass die traditionellen Datenbanksysteme nicht mehr den Ansprüchen an Datendurchsatz und Zugriffsgeschwindigkeit genügen. Das bedeutet nicht, dass ein komplettes Versagen vorliegt, sondern dass bedeutender Aufwand betrieben werden muss, um den Anforderungen gerecht zu werden, zum Beispiel Investitionen für den Ausbau der zugrunde liegenden Hardware. In Relation zum Nutzen sind diese Kosten nicht verhältnismäßig [SMA+07, AGMG+09].

Die etablierten, traditionellen Datenbanksysteme implementieren das relationale beziehungsweise das objekt-relationale Datenmodell, wobei in den meisten Fällen das zeilen-basierte Speichermodell zum Einsatz kommt. Der Hintergrund ist, dass mit *SQL* eine standardisierte Datenabfrage- und -manipulationssprache zur Verfügung steht, mit der auf relationale Datenbanken bis auf wenige Ausnahmen herstellerunabhängig zugegriffen werden konnte. Das kam insbesondere Datenbankanwendungen zugute, da bestehende Implementationen mit vergleichsweise geringem Aufwand von einem Datenbanksystem zu einem anderen migriert werden konnten. Zudem hat sich herausgestellt, dass sich das relationale Datenmodell als recht flexibel im Bezug auf das Spektrum der möglichen Anwendungsdomänen zeigt. Das bedeutet, dass die traditionellen Datenbanksysteme eine Vielzahl an möglichen Anwendungen abdecken können. Das ist insbesondere für Endanwender interessant, da zum Beispiel ein im Unternehmen zentral bereitgestelltes Datenbanksystem eine Vielzahl von verschiedenen Datenbankanwendungen unterstützen kann.

Dennoch hat sich auch gezeigt, dass die mittlerweile anfallenden, sehr großen Datenmengen selbst von traditionellen Datenbanksystemen kaum verarbeitbar waren, die enorm performante Hardware nutzten. Solche Datenmengen fallen zum Beispiel an, wenn die Daten eines Sensornetzes in einer Datenbank gespeichert werden sollen. Selbst der Einsatz von verteilten[11] Datenbanksystemen vermögen zwar solche Datenmengen persistent im Tera- und Petabyte-Bereich zu speichern, aber die permanente Aktualisierung der Datenbank durch neue Sensorwerte sowie die gegebenfalls gleichzeitige Abfrage der Datenbank zu Analysezwecken bringen sie an die Grenzen der noch zumutbaren Anforderungen hinsichtlich der Antwortzeiten. Zudem muss bedacht werden, dass die Implementation traditioneller Datenbanksystemen einen signifikaten Aufwand betreibt, um das *ACID*-Paradigma zu erfüllen (vgl. dazu Abbildung 2.4). Dieser Aufwand potenziert sich, wenn ein verteiltes Datenbanksystem zum Einsatz kommt, zum Beispiel durch die Durchführung einer Transaktion.

Aus diesen Gründen kamen sogenannte *NoSQL*-Datenbanksysteme auf. Sie sind dadurch charakterisiert, dass sie ein anderes als das relationale Datenmodell implementieren, wobei auch darauf spezialisierte Speichermodelle zum Einsatz kommen. *NoSQL* bedeutet hierbei *Not only SQL*, das ausdrückt, dass zur Datenmanipulation- und -abfrage kein *SQL* als Schnittstelle eingesetzt wird. Das ist auch insofern nachvollziehbar, da *SQL* primär für relationale Datenbanken ausgerichtet ist. Zudem sind *NoSQL*-Datenbanksysteme im Vergleich zu traditionellen, relationalen Datenbanksystemen auf eine Anwendungsdomäne und zum Teil auf nur einen Anwendungsfall spezialisiert. Das bedeutet, sie sind eingeschränkt, was die Spanne an potentiellen Datenbankanwendungen betrifft. Der Vorteil dieser Spezialisierung ist aber, dass die Implementierung im Bezug auf das Datenmodell zielgerichtet ist und Funktionalitäten beinhaltet, die effektiv das Datenmodell unterstützen. Bei Graph-basierten Datenbanksystemen können solche Funktionalitäten zum Beispiel sein, den kürzesten Pfad eines Graphknotens zu einem anderen zu berechnen.

[11] Verteilte Datenbanksysteme und Replikas werden im weiteren Verlauf des Textabschnittes genauer erläutert.

Zudem verzichten *NoSQL*-Datenbanksysteme zwar nicht völlig auf die Einhaltung des *ACID*-Paradigmas, aber sie gehen von völlig anderen Voraussetzungen aus, um den geänderten Anforderungen Rechnung zu tragen. Nach *Robinson et al.* werden sie als *BASE*-Paradigma[12] bezeichnet und bedeuten im einzelnen:

- Eingeschränkte Erreichbarkeit[13]: Der Ausfall eines Teilsystems führt nicht automatisch zum Ausfall des gesamten Anwendungsfalles. Nach *Pritchett* bedeutet es, dass solche Ausfälle nicht die Ausnahme, sondern die Regel darstellen und müssen nicht unbedingt eine negative Bedeutung haben [Pri08]. So können zum Beispiel in einem verteilten Datenbanksystem ein oder mehrere beteiligte Server heruntergefahren werden, um Wartungsarbeiten durchzuführen. Dabei wird auch toleriert, dass es hinsichtlich der Erreichbarkeit und der Verarbeitungsgeschwindigkeit der Nutzernanfragen an das verteilte Datenbanksystem Einbußen gibt. Ein prominentes Beispiel ist die Verteilung von Nachrichten oder Neuigkeiten an eine Gruppe von Benutzern, wie es etwa *Facebook* praktiziert: fällt ein Server aus, kann es dazu kommen, dass ein Teil der Empfänger die verbreitete Nachricht nicht sofort beziehungsweise im Extremfall überhaupt nicht erhält.
- Scheinbare nicht-deterministische Zustände[14]: es wird toleriert, dass in einer Datenbank eine zeitliche Varianz zwischen zwei Zuständen auftreten können. Dieser Sachverhalt wird von *Pritchett* anhand eines Beispieles erklärt: man nehme eine Datenbank an, in der eine bestimmte Sache von Personen gespeichert werden. Das kann zum Beispiel Geld oder virtuelle Gegenstände sein. Durch die Datenbank wird die Übergabe dieser Dinge repräsentiert, dass heisst, die Übergabe gelingt dadurch, dass die Sache von einem Personendatensatz zu einem anderem transferiert wird. Dieser Transfer kann durch zwei Operationen ausgedrückt werden: 1) nehme (subtrahiere) das zu transferierende Ding von Person A und 2) gebe (addiere) es zu Person B. Es macht Sinn, beide Operation zu einer atomaren Transaktion zu bündeln, was in einem streng konsistenten Datenbanksystem der Standard ist. Die Datenbank geht also atomar von einem Zustand in den nächsten über, wobei dieser Übergang deterministisch ermittelt werden kann [Pri08].
  Toleriert man aber, dass beide Operationen nicht als atomare Transaktion ausgeführt werden, so kann es passieren, dass der Zielzustand nach dem Transfer erst mit einem (durchaus sehr geringem) Zeitversatz erreicht wird. Damit ist das Datenbanksystem nur scheinbar nicht-deterministisch.
- Schlussendliche Konsistenz[15]: dieser Punkt hängt stark mit dem vorhergehenden Punkt zusammen und betrifft verteilte Datenbankesysteme im besonderen Maße. Auf das im vorhergehenden Punkt beschriebene Beispiel bezogen heisst es, dass die beiden Operationen getrennt voneinander ausgeführt werden können. Dabei können zwei unterschiedliche Teilsysteme des verteilten Datenbanksystems betroffen sein. Dass heisst, auf einem Teilsystem wird die erste Operation ausgeführt und die zweite auf einem anderem. *Pritchett* führt hier nun an, dass nun die erste Operation erfolgreich durchgeführt wurde, aber es bei der zweiten nicht direkt darauffolgend dazu kam, sie durchzuführen. Die Gründe dafür können beispielsweise sein, dass das zweite Teilsystem ausgefallen ist oder das Teilsy-

12 *BASE* ist ein Akronym der englischen Wörter *Basic Availability*, *Soft state* und *Eventually consistent*.

13 Im Original wird dieser Punkt als *Basic Availability* umschrieben. Nach *Robinson et al.* ist der genaue Wortlaut: "*The store appears to work most of the time.*" [RWE13]

14 Im Original wird dieser Punkt als *Soft state* bezeichnet. Die originale Erklärung dieses Punktes nach *Robinson et al.* lautet: "*Stores don't have to be write-consistent, nor do different replicas have to be mutually consistent all the time.*" [RWE13]

15 Eine genaue Übersetzung dieses Punktes des Originals *eventually consistency* nach *Robinson et al.* ist nicht trivial. Das Original ist: "*Stores exhibit consistency at some later point, e.g. lazy writing at read time.*" [RWE13]

stem die Ausführung der Operation verschoben hat. Das Zurückstellen kann dabei mehrere mögliche Gründe haben, etwa die Bündelung von Schreiboperationen zu einer großen oder die Priorisierung von anderen parallel abzuarbeitenden Operationen (vgl. hierzu *Kemper und Eickler* [KE11, S.672 ff.]).

Ingesamt besteht also der Hauptunterschied zwischen traditionellen Datenbanksystemen, die das *ACID*-Paradigma durchsetzen und *NoSQL*-Datenbanksystemen, die auf dem *BASE*-Paradigma basieren, in einer differenzierten Auffassung der Konsistenz der Datenbank. Sowohl nach *Robinson et al.*, *Kemper und Eickler*, *Elmasri und Navathe* als auch *Pritchett* wird bei erstgenannten Datenbanksystemen die Konsistenz der Datenbank zu jeder Zeit sichergestellt. Daher werden Datenmanipulations- und -abfrageoperationen als einzelne Transaktionen durchgeführt, deren Ausführung als *pessimistisch* bezeichnet werden. Es wird also ein sehr großer Aufwand (etwa durch Sperr- und Synchronisationsprotokolle) betrieben, um die Konsistenz der Datenbank zu gewährleisten. Dieser Aufwand potenziert sich bei verteilten Datenbanksystemen, da zusätzlich die Konsistenz in allen Teilsystemen hergestellt werden muss. Es ist offensichtlich, dass dieser Mehraufwand die Leistungsfähigkeit hinsichtlich der Verarbeitung von Datenmanipulations- und -abfrageoperationen beeinträchtigt. Dies soll keine Aussage sein, dass diese Datenbanksysteme grundsätzlich nicht leistungsfähig sind, aber durch den Mehraufwand skalieren sie nicht mit der Datenmenge, wie sie zum Beispiel durch *BigData*-Anwendungen anfallen.

Im Gegensatz dazu steht bei Datenbanksystemen, die dem *BASE*-Paradigma folgen, die Konsistenz der Datenbank nicht im Vordergrund. Hier werden Einschränkungen toleriert, um die Leistungsfähigkeit im Bezug auf die Anzahl der verarbeiteten Datenmanipulations- und -abfrageoperationen bedeutend zu erhöhen. Zudem sind diese Datenbanksysteme aufgrund der geänderten Voraussetzungen darauf ausgelegt, als verteiltes Datenbanksystem zu agieren, um besser mit den gestiegenen Datenmengen zu skalieren. Es existiert zwar auch das Transaktionskonzept, aber deren Ausführung wird als *optimistisch* bezeichnet. *Kemper und Eickler* haben den Unterschied in [KE11, S.343] so beschrieben: *"[Man] geht davon aus, dass Konflikte selten auftreten und man Transaktionen einfach mal ausführen sollte und im Nachhinein (à posteriori) entscheidet, ob ein Konflikt aufgetreten ist oder nicht."* Pessimistische Transaktionsausführungen gehen von der Prämisse aus, dass *"Konflikte auftreten werden. Deshalb werden Vorkehrungen getroffen, um potentielle Konflikte zu verhindern – in manchen Fällen auf Kosten der Parallelität."*

Die existierenden Datenbanksysteme lassen sich nach mehreren Kriterien klassifizieren. Nach *Elmasri und Navathe* liegen unter anderem die folgenden Unterscheidungskriterien vor [EN09, S.52 ff.]:

1. die unterstützten Datenmodelle: existierende Datenbanksysteme implementieren mindestens ein Datenmodell, wie sie in Textabschnitt 2.1 bereits beschrieben sind. Einige Beispiele für reale Datenbanksysteme und ihr implementiertes Datenmodell sind in Tabelle 2.2 auf Seite 33 dargestellt.

2. die unterstützten Speichermodelle: ein Datenbanksystem kann zur Persistierung der Datenbanken ein oder mehrere Speichermodelle einsetzen, wie sie in Textabschnitt 2.2 näher beschrieben sind. Grundsätzlich ergibt sich aus der Kombination der Kriterien Datenmodell und Speichermodell der Charakter eines Datenbanksystems. Das bedeutet, der primäre Anwendungsbereich ist allgemein oder aufgaben-spezifisch. Im erstgenannten Bereich kann durch das Datenbanksystem eine Vielzahl an Anwendungen unterstützt werden, das beim letztgenannten Bereich nicht ohne weiteres möglich. Das ist zum Beispiel der Fall, wenn ein Datenbanksystem für eine bestimmte Anwendung entwickelt wurde.

3. die Anzahl der Nutzer, die gleichzeitig auf das Datenbanksystem zugreifen können: sofern nur ein Nutzer auf das Datenbanksystem zugreifen kann, so wird es als *Einbenutzersystem* bezeichnet. Solche Datenbanksysteme werden häufig in Form von Softwarebibliotheken angeboten und können in Softwareapplikationen eingebettet werden. Sofern mehr als ein Nutzer zugreifen können, so wird es als *Mehrbenutzersystem* klassifiziert. Solche Systeme sind zum überwiegenden Teil eine eigenständige Computersoftware und stellen die Mehrheit aller Datenbanksysteme dar.

4. die Anzahl der Systeme, auf die sich das Datenbanksystem verteilt: wird das Datenbanksystem auf nur einem Informationssystem (zum Beispiel PCs und Server) betrieben, so bezeichnet man es als *zentral*. Wird es hingegen auf mehreren Informationssystemen betrieben, die untereinander vernetzt sind und Daten austauschen können, so bezeichnet man es als *dezentral* beziehungsweise *verteilt*.

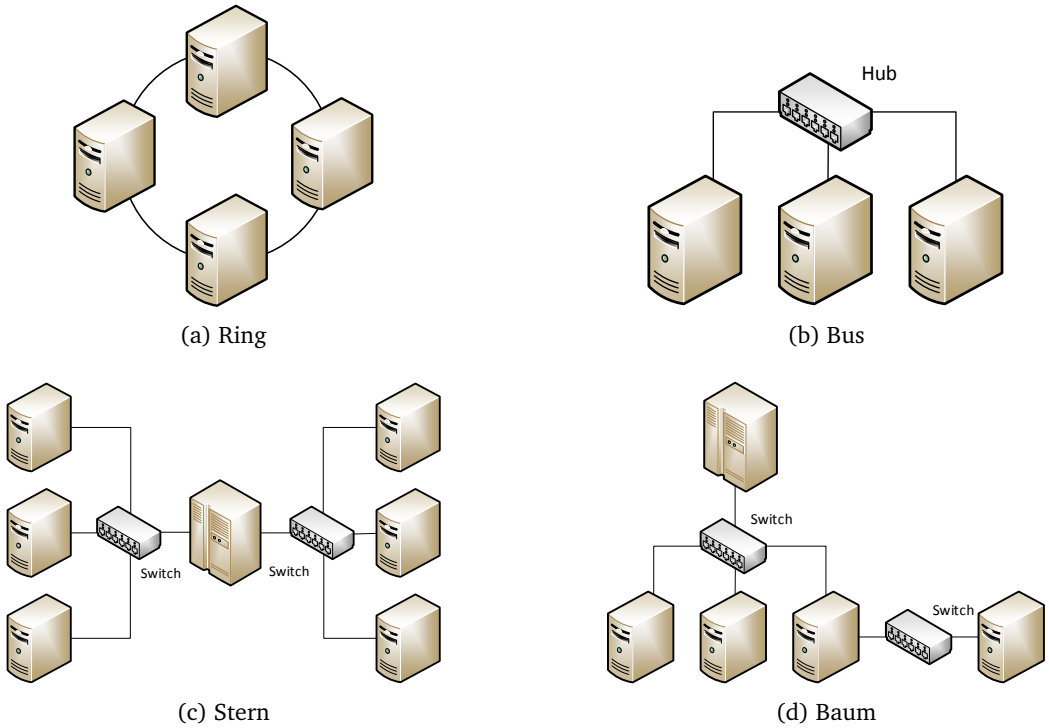
   Hierbei wird auch unterschieden, ob auf allen beteiligten Informationssystemen dasselbe Datenbanksystem zum Einsatz kommt oder verschiedene über eine gemeinsame Schnittstelle, zum Beispiel eine gemeinsame Datenabfrage- und Manipulationssprache wie *SQL*, kommunizieren. Im ersten Fall gilt das verteilte Datenbanksystem als *homogen* und im zweiten Fall als *heterogen* beziehungsweise *föderativ*. Ein föderatives Datenbanksystem kann als ein Datenbanksystem angesehen werden, *"bei dem die beteiligten Datenbankmanagementsysteme lose gekoppelt sind ein gewisses Ausmaß an Autonomie aufweisen."* [EN09, S.53].

Datenmodell	**Beispiele**
SQL- und OQL-Schnittstelle	
relational	*Oracle*, *Postgresql*, *MySQL*
objekt-orientiert	*Gemstone*, *ObjectStore*, *POET*
objekt-relational	*Oracle*, *DB2*, *Caché*
Baumstruktur	*Microsoft SQLServer*, *TokuDB*
keine einheitliche, standardisierte Schnittstelle	
Spalten-orientiert	*Cassandra*, *HBase*, *Accumulo*
Dokument-orientiert	*CouchDB*, *MongoDB*
Schlüssel-Wert-orientiert	*Dynamo*, *Riak*, *Terracotta*
Graph-orientiert	*Neo4J*, *OrientDB*, *Virtuoso*
SQL-ähnliche Schnittstelle (NewSQL)	
In-Memory	*SAP HANA*, *EXASolution*, *VoltDB*

Tabelle 2.2: Beispiele für Datenbanksysteme nach ihrem Datenmodell

Insbesondere das zuletzt genannte Kriterium muss tiefergehend betrachtet werden, da sich nachfolgende Textabschnitte darauf beziehen. So werden die beteiligten Informationssysteme, etwa PCs oder Server, in Anlehnung an die Graphentheorie als *Knoten* des verteilten Datenbanksystems bezeichnet. Das Netzwerk, dass die Knoten bilden, wird im allgemeinen als *Cluster* bezeichnet[16].

Verteilte Datenbanksysteme werden im allgemeinen in einem sogenannten *shared-nothing*-Netzwerk betrieben. Das bedeutet, dass die Netzwerkknoten ihre Betriebsmittel nicht teilen. Somit hat jeder Netzwerkknoten seine eigenen Prozessoren sowie eigenen Arbeits- und Massenspeicher, auf die der Netzwerkknoten exklusiven Zugriff hat. Die Vernetzung der Netzwerkknoten folgt einer Topologie, wobei sich vier Topologien herausgestellt haben: Ring, Bus, Stern und Baum. In Abbildung 2.7 sind alle vier Topologien dargestellt. Jede dieser vier Topologien hat Vor- und Nachteile hinsichtlich des Datendurchsatzes, der Datenübertragungsbandbreite, des Energieverbrauches und des Preises. Mittlerweile werden Computernetzwerke, die zum Beispiel der Bus- oder Ring-Topologie folgen, nicht mehr eingesetzt, da sie im Bezug auf den Datendurchsatz prinzipbedingt den anderen Topologien im Nachteil sind.



(a) Ring (b) Bus

(c) Stern (d) Baum

Abbildung 2.7: Netzwerktopologien bei verteilten Datenbanksystemen

In Abbildung 2.7 sind auch die vorherrschenden Serverarten und die Kommunikationstechniken dargestellt. Während in der Ring-Topologie alle Knoten direkt miteinander verbunden sind, werden bei allen anderen Topologien sogenannte *Hubs* und *Switche* genutzt. Die Hubs sind mittlerweile fast komplett von Switchen verdrängt worden, da sich die interne Bandbreite des Hubs

[16] Ein Cluster grenzt sich insofern von einer *Cloud* oder einem *Grid* ab, dass die von den teilnehmenden Knoten bereitgestellten Ressourcen lokal zusammengefasst und auch lokal verwendet werden. Extern betrachtet fasst ein *Grid* solche lokalen Cluster zu einer logischen Einheit zusammen und stellt die Ressourcen allen Anwendungen abstrahiert und global zur Verfügung. Es ist sozusagen ein Meta-Cluster.

von allen angeschlossenen Knoten geteilt werden muss. Bei Switchen ist dies nicht der Fall; hier kann jeder Teilnehmer mit jedem anderen in der maximal möglichen Bandbreite kommunizieren. Bei Stern- und Baum-Topologien ist außerdem zu erkennen, dass an zentraler Stelle zumeist ein besonders potent ausgestatteter PC oder Server platziert ist. Der Grund ist in den meisten Fällen, dass diese zentralen Knoten gegenüber den übrigen Knoten einen Mehraufwand leisten müssen, zum Beispiel Verwaltungs-, Koordinierungs- und Überwachungsfunktionen für den gesamten Cluster.

Es stellt sich natürlich die Frage, warum überhaupt Datenbanksysteme verteilt ausgeführt werden. Die beiden meistgenannten Gründe sind Ausfallsicherheit und Lastverteilung. Die Ausfallsicherheit wird dadurch gewährleistet, dass die gespeicherten Daten der Datenbank mehrfach im Cluster vorgehalten werden. Sofern also ein Knoten ausfällt, kann dennoch auf eine verteilte Kopie zurückgegriffen werden. Diese Verteilung von Kopien wird als *Replikation* bezeichnet. Der *Replikationsfaktor* gibt an, wieviele Repliken im Cluster verteilt werden sollen. Die Repliken bieten auch den Vorteil, dass die replizierten Daten lokal am Knoten vorliegen. Damit ist auch eine Lastverteilung möglich, da die eingehenden Datenbankabfragen auf mehrere Knoten verteilt werden können.

Im Bezug auf die Lastverteilung können zwei Arten unterschieden werden:

1. Die erste Art sind diejenigen verteilten Datenbanksysteme, die eine externe Komponente brauchen, um Datenmanipulations- und -abfrageoperationen auf die Instanzen des Datenbanksystems zu verteilen, die an sich unabhängig und als zentrales Datenbanksystem konzipiert sind.

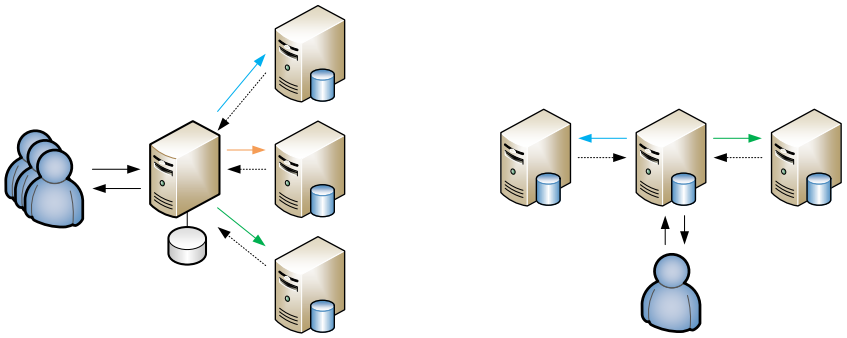
   Eine solche Komponente ist zum Beispiel *Slony-I* für einen Verbund von *Postgresql*-Servern. Dabei kommunizieren die *Postgresql*-Nutzer nicht direkt mit den Knoten, sondern nur mit *Slony-I*, das als eine Lastverteilung für die Server dient. Abbildung 2.8(a) auf Seite 36 zeigt einen schematischen Aufbau eines verteilten Datenbanksystems mit einer externen Komponente.

   Eingehende Datenmanipulations- und -abfrageoperationen (eingehender schwarzer Pfeil auf die externe Komponente) werden analysiert und in einzelne Operationen aufgetrennt, die dann simultan auf den verbundenen Knoten ausgeführt werden (blauer, oranger und grüner Pfeil). Die zurückgegebenen Antwortdatensätze werden durch die externe Komponente zusammengefügt und an den anfragenden Nutzer zurückgeliefert. Damit diese Lastverteilung korrekt ausgeführt werden kann, muss die externe Komponente Informationen vorhalten, wieviele Knoten aktuell verfügbar sind sowie permanente Verbindungen herstellen. Das ist insofern wichtig, da die externe Komponente auch Heuristiken über die Verteilung der einzelnen Datensätze in den Datenbanken führen muss (Datenbank unterhalb der externen Komponente in Abbildung 2.8(a)). Diese sogenannte *Partitionierung* bildet die Grundlage, um die eingehenden Datenmanipulations- und -abfrageoperationen in verteilte umzuformen und auszuführen, da die externe Komponente immer Kenntnis über den Ablageort jedes einzelnen Datensatzes hat.

   Insgesamt bietet diese Art von Lastverteilung den Vorteil, dass auch zentrale Datenbanksysteme als ein verteiltes ausgeführt werden können, um zum Beispiel die Ausfallsicherheit, die Zugriffsgeschwindigkeit und den Datendurchsatz der Datenbank zu erhöhen. Zudem ist auch eine gewisse Heterogenität bei der Auswahl der eingesetzten Datenbanksysteme auf den Knoten möglich. Sofern alle eingesetzten Datenbanksysteme dieselbe Sprache zur Datenmanipulation- und -abfrage wie etwa *SQL* nutzen, ist es unerheblich, welches Datenbanksystem von welchem Anbieter eingesetzt wird. Dasselbe gilt natürlich auch für die

externe Komponente. Hier können mehrere Zugriffssprachen sowohl in Richtung der Nutzer als auch in Richtung der Knoten im Cluster genutzt werden. Somit wird insgesamt diese Art der Lastverteilung zumeist bei heterogenen sowie föderativen Datenbanksystemen eingesetzt.

Der gravierende Nachteil ist, dass die externe Komponente ein bedeutendes Risiko ist: wenn sie ausfällt, kann aus Sicht des Nutzers nicht mehr auf den Cluster und damit auf die Datenbank zugegriffen werden.



(a) über externe Komponente (b) intern

Abbildung 2.8: Lastverteilung von Datenbankoperationen: eingehende Nutzeranfragen (schwarze Pfeile) werden analysiert, aufgeteilt und simultan auf den Clusterknoten ausgeführt (blaue, orange und grüne Pfeile). Die Antworten der Clusterknoten werden zusammengefügt und an den Nutzer zurückgegeben (schwarze Pfeile).

2. Bei der zweiten Art von Lastverteilung sind die beschriebenen Funktionalitäten bereits fest in den Datenbanksystemen implementiert und benötigen daher keine externe Komponente. Sie enthalten also bereits Funktionen, um einen Cluster zu bilden und Daten je nach Replikationsfaktor im Cluster zu replizieren. Abbildung 2.8(b) zeigt schematisch ein verteiltes Datenbanksystem, dass aus drei Knoten einen Cluster formt.

   Prinzipiell können sich Nutzer mit jedem der Knoten im Cluster verbinden und Datenmanipulations- und -abfrageoperationen ausführen. In Abbildung 2.8(b) ist nur ein Nutzer dargestellt, wobei die Vorgehensweise dieselbe ist: die eingehende Operation wird analysiert und je nach Partitionierung in simultan ausgeführte Operationen aufgeteilt (blauer und grüner Pfeil). Simultan bedeutet in diesem Zusammenhang auch, dass sich die Antwortdatensätze an den Nutzer aus den Teildatensätzen zusammensetzen, die von den Knoten zurückgeliefert werden. Das betrifft auch die Datensätze, die lokal auf dem Knoten vorhanden sind, auf den der Nutzer zugreift[17].

   Diese Art der Lastverteilung in einem verteilten Datenbanksystem bietet den Vorteil, dass die Ausfallsicherheit des Clusters höher ist, da keine zentrale externe Komponente zur Lastverteilung und Replikation nötig ist. Allerdings müssen die Heuristiken und die Partitionierungsdaten auf jedem Knoten im Cluster gespeichert werden. Hier ergeben sich eindeutig Datenredundanzen. Der Nachteil ist, dass innerhalb eines Clusters fast immer nur das verteilte Datenbanksystem eines Anbieters eingesetzt werden kann, da zumeist

[17] Der orange Pfeil, wie er in Abbildung 2.8(a) ersichtlich ist, ergibt sich implizit aus den lokal vorhandenen Datensätzen und ist daher in Abbildung 2.8(b) nicht dargestellt.

proprietäre Protokolle zwischen den Clusterknoten zur Replikation und Kommunikation eingesetzt werden. Somit kann man davon ausgehen, dass diese Art der Lastverteilung nur bei verteilten, homogenen Datenbanksystem zum Einsatz kommt.

Die Praxis zeigt aber, dass die Vorteile der Lastverteilung bei verteilten Datenbanksystemen ohne externe Komponente gegenüber den Nachteilen überwiegen. Fast alle modernen verteilten Datenbanksysteme bieten die Möglichkeit, Cluster ohne externe Komponente zu erstellen. Beispiele sind *MySQL Cluster* für relationale, *Oracle* für objekt-relationale sowie *Cassandra* für Spaltenorientierte Datenbanksysteme.

Dennoch wurde bereits im Jahr 2000 prognostiziert, dass verteilte Datenbanksysteme fundamentale Einschränkungen besitzen. Die Einschränkungen sind als *CAP*-Theorem[18] bekannt ([GL02, Bre12]) und besagen, dass verteilte Datenbanksysteme nur zwei der drei folgenden Prinzipien gleichzeitig garantieren können: Konsistenz (engl. *consistency*), Verfügbarkeit (engl. *availability*) und Robustheit gegen den Ausfall von Teilsystemen (engl. *partition tolerance*). Konsistenz bedeutet hierbei, dass die Daten der verteilten Datenbank zu jeder Zeit übereinstimmen müssen. Verfügbarkeit bedeutet, dass jede Datenmanipulations- und -abfrageoperation eines Nutzer garantiert eine Antwort erhält. Dabei ist Erfolg oder Misserfolg der Nutzeranfrage unerheblich, allerdings ist die Antwortzeit relevant. Die Robustheit gegenüber einem Teilsystemausfalls besagt, dass die verteilte Datenbank auch dann noch operabel ist, wenn ein Teilsystem ausgefallen ist. Dass kann zum Beispiel ein Knoten oder ein ganzes Teilnetz eines Clusters sein, wenn ein ein Hub oder Switch versagt.

Das *CAP*-Theorem wurde in der Literatur sehr kontrovers diskutiert. Das ursprüngliche *CAP*-Theorem war eine Annahme, die im Jahr 2000 von *Brewer* während eines Symposiums der Fachwelt vorgestellt wurde. Sie war das Ergebnis einer Analyse der damaligen verteilten Datenbanksysteme beziehungsweise deren vermuteten Unzulänglichkeiten. *Gilbert und Lynch* konkretisierten 2002 *Brewer*'s Annahme und etablierten unter Verwendung von Axiomen die Annahme als Theorem [GL02]. Das bedeutet, dass *Brewer*'s Annahme in der Theorie als bewiesen gilt. Die axiomatische Beweiskette von *Gilbert und Lynch* ist das Kernstück der Kontroverse. Das *CAP*-Theorem ist nachvollziehbar, sofern man von der Situation im Jahre 2000 ausgeht. *Brewer*'s Annahme basiert auf einer Studie von ausschließlich verteilten, relationalen Datenbanksystemen beziehungsweise den damals hauptsächlich implementierten *ACID*-Paradigma. Für diesen Typus von Datenbanksystemen ist das *CAP*-Theorem tatsächlich plausibel. Die Situation hat sich aber geändert, als *NoSQL*-Datenbanksysteme eingeführt und genutzt wurden. Nach *Abadi* sind sie sozusagen der praktische Beweis, dass das *CAP*-Theorem gemäß *Gilbert und Lynch* nicht überzeugend ist [Aba12]. Dazu untersuchte *Abadi* die verteilten *NoSQL*-Datenbanksysteme *Amazon Dynamo*, *Cassandra*, *Voldemort*, *Yahoo PNUTS* und *Riak*. Dabei stellte er fest, dass das *CAP*-Theorem zwar nicht gebrochen, aber zu strikt ausgelegt ist. Wie bereits erwähnt, besagt es, dass ein verteiltes Datenbanksystem nur zwei der drei Aspekte Konsistenz, Verfügbarkeit und Robustheit gleichzeitig unter dem Einfluss des *ACID*-Paradigmas garantieren kann. Die betrachteten Datenbanksysteme können alle drei Aspekte gleichzeitig garantieren, aber dafür wurde das *ACID*-Paradigma außer Acht gelassen. Als Resultat seiner Betrachtungen hat *Abadi* den Term *CAP* zu *PACELC* umdefiniert. *PACELC* ist ein Akronym für den englischen Satz "*If there is a partition (P), how does the system trade off availability and consistency (A and C); else (E), when the system is running normally in the absence of partitions, how does the system trade off latency (L) and consistency (C)?*" [Aba12, S.41]. Frei ins Deutsche übersetzt bedeutet es: Wenn ein Paritionierungsschema für die Daten eines Datenbanksystem existiert, welche Abstriche beziehungsweise Kompromisse werden hinsichtlich der Konsistenz und Verfügbarkeit gemacht? Sofern keine Paritionierung eingesetzt

[18] *CAP* ist ein Akronym für die englischen Wörter *Consistency*, *Availability* und *Partition tolerance*.

wird und das Datenbanksystem normal ausgeführt wird, welche Abstriche beziehungsweise Kompromisse werden hinsichtlich der Antwortzeiten und Konsistenz gemacht? Zusammengefasst hat *Abadi* versucht, das *CAP*-Theorem insofern zu modernisieren, dass neben verteilten auch zentrale Datenbanksysteme mit und ohne der Unterstützung des *ACID*-Paradigmas betrachtet werden können.

Pritchett hat parallel zu *Abadi* ebenfalls *NoSQL*-Datenbanksysteme im Kontext des *CAP*-Theorems untersucht und kam zu fast gleichlautenden Ergebnissen [Pri08]. Dabei hat *Pritchett* seine Ergebnisse als *BASE*-Prinzip publiziert, wie es bereits in diesem Textabschnitt erläutert wurde. In der Kontroverse um das *CAP*-Theorem hat 2012 auch *Brewer* seine originale Annahme rekapituliert und kam zu der Erkenntnis, dass seine Annahme aufgrund der geänderten Situation zu strikt ist [Bre12]. Zusammengefasst bedeutet es, dass das *CAP*-Theorem prinzipiell kein Theorem mehr ist und sich *Brewer*'s Annahme als falsch herausgestellt hat. Dementsprechend hat das *CAP*-Theorem bei der Entwicklung und Implementierung von verteilten Datenbanksystemen keine Bedeutung mehr.

Im weiteren Verlauf dieser Dissertation werden wiederholt die Terme Datenbank- und Datenbankmanagementsystem genutzt. Eine genaue Definition dieser Terme im Kontext dieser Dissertation ist notwendig, um Abgrenzungen vornehmen zu können.

Definition 1: Datenbankmanagementsystem
Ein Datenbankmanagementsystem ist eine Computersoftware oder Softwarebibliothek, das die Erstellung und Manipulation einer Datenbasis (der Datenbank) ermöglicht. Das Datenbankmanagementsystem organisiert und verwaltet die gespeicherten Datenbankdaten nach einem Daten- und Speichermodell, wie sie in Textabschnitt 2.1 und 2.2 beschrieben sind. Zusätzlich wird eine Schnittstelle bereitgestellt, durch die die gespeicherten Datenbankdaten gemäß dem Ablauf in Textabschnitt 2.3 abgefragt und manipuliert werden können.

Definition 2: Datenbanksystem
Ein Datenbanksystem ist eine Computersoftware, die auf einem PC oder Server ausgeführt werden kann und mindestens einem Kriterium entspricht, wie sie im Textabschnitt 2.4 erläutert sind. Das bedeutet, dass ein Datenbanksystem eine Computersoftware ist, die ein Datenbankmanagementsystem gemäß der Definition 1 ausführt, wobei insbesondere zwischen zentral betriebenen und verteilten Datenbanksystem unterschieden wird. Es ermöglicht einem oder mehreren Benutzern, Datenbankanwendungen auszuführen, die auf die Datenbanken lesend oder modifizierend zugreifen beziehungsweise neue Datenbanken erstellen und existierende löschen oder in ihrem Schema ändern können.

2.5 Petri-Netze und dessen Derivate

Dieser Textabschnitt soll dazu dienen, die Grundlagen und Eigenschaften von *Petri*-Netzen zu erklären, da sie für die Kapitel 5 und 6 von Bedeutung sind. Daher werden sie hinsichtlich formaler Definition, Eigenschaften und Arbeitsweise näher betrachtet. Insbesondere das aktuellste Derrivat von *Petri*-Netzen, *Queued Petri Nets*, soll in diesem Textabschnitt eingehender erläutert werden.

Petri-Netze

Petri-Netze sind neben Warteschlangen-Netzwerken eine Form der Datenflussmodellierung. Sie wurden 1962 von *Carl Adam Petri* definiert. Sie sind ein Formalismus für die Beschreibung von Nebenläufigkeiten und Synchronisationsmechanismen in verteilten Systemen [BK02a, PW08].

Die einfachste Form von *Petri*-Netzen sind sogenannte *PT*-Netze (*Place-Transition-Nets*, abgekürzt PTN), die einen bipartiten, gerichteten Graphen darstellen. Nach [BK02a] und [PW08] besteht dieser Graph aus

- Stellen (engl. *places*), dargestellt als Kreis. Stellen repräsentieren Bedingungen oder Objekte, zum Beispiel eine Variable in einem Computerprogramm.
- Marken (engl. *tokens*), dargestellt als schwarze Punkte in Stellen. Sie repräsentieren den Zustand beziehungsweise den Wert einer Bedingung oder eines Objektes, zum Beispiel den Inhalt einer Variablen in einem Computerprogramm.
- Transitionen (engl. *transitions*), dargestellt als Rechteck. Transitionen repräsentieren Zustandsänderungen, zum Beispiel von Variablen in einem Computerprogramm.
- Kanten (engl. *arcs*), dargestellt als Pfeile, die Stellen mit Transitionen verbinden. Sie definieren somit, welche Bedingungen oder Objekte durch welche Zustandsänderungen geändert werden dürfen.

Diese Bestandteile des Graphen sind in Abbildung 2.9 dargestellt.

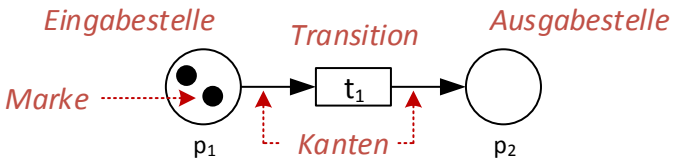


Abbildung 2.9: Bestandteile eines *Place-Transition*-Netzes

Ein PTN ist nach [BK02a, S.84-85] als 5-Tupel *PN* definiert:

Definition 3: *Place-Transition*-Netz (PTN)
Ein Place-Transition-Netz (PTN) ist definiert als

$$PN = (P, T, I^-, I^+, M_0)$$

wobei

- $P = \{p_1, p_2, \ldots, p_n\}$ *eine endliche Menge von Stellen ist*
- $T = \{t_1, t_2, \ldots, t_m\}$ *eine endliche Menge von Transitionen ist*
- P *und* T *sind disjunkt:* $P \cap T = \emptyset$
- $I^+, I^- : P \times T \to \mathbb{N}_0$ *die vorwärts beziehungsweise rückwärts gerichtete Kantenfunktionen sind*
- $M_0 : P \to \mathbb{N}_0$ *die initiale Markenbelegung der Stellen ist.*

Weiterhin soll für die Stellen und Transitionen gelten:

- *Eingabestellen von Transition t:* $\bullet t := \{p \in P | I^+(p,t) > 0\}$
- *Ausgabestellen von Transition t:* $t\bullet := \{p \in P | I^-(p,t) > 0\}$
- *Eingabetransitionen von Stelle p:* $\bullet p := \{t \in T | I^+(p,t) > 0\}$
- *Ausgabetransitionen von Stelle p:* $p\bullet := \{t \in T | I^-(p,t) > 0\}$

womit jede Teilmenge $X \subseteq P \cup T$ *als* $\bullet X = \bigcup_{x \in X} \bullet x$ *und* $X\bullet = \bigcup_{x \in X} x\bullet$ *definiert werden kann.*

Es sei auch angemerkt, dass die Defintion eines PTNs in der Literatur unterschiedlich angegeben ist. Nach *Bause und Kitzinger* [BK02a] definiert bereits das Quadrupel (P, T, I^-, I^+) ein PTN. Sofern noch M_0 angegeben wird, so spricht man von einem *System*. Nach *Priese und Wimmel* [PW08] sind die Mengen P und T zusätzlich angeordnet ($P = \{p_1, p_2, \ldots, p_{|P|}\}$, $T = \{t_1, t_2, \ldots, t_{|T|}\}$). Die Kantenfunktionen I^+ und I^- werden als Matrizen angegeben ($\mathbb{F} = |P| \times |T|$ *über* $\mathbb{N}$, $\mathbb{B} = |P| \times |T|$ *über* $\mathbb{N}$). Man kann beide Sichtweisen kombinieren, indem eine Abbildung F genutzt wird:

$$F : P \times T \cup T \times P \to \mathbb{N}$$

dessen Definition

$$\forall x, y \in P \cup T : F(x,y) := \begin{cases} \mathbb{F}_{i,j} & \textit{falls } x = p_i \textit{ und } y = t_j \\ \mathbb{B}_{i,j} & \textit{falls } x = t_j \textit{ und } y = p_i \end{cases}$$

lautet. Somit führen genau $\mathbb{F}_{i,j}$ Kanten von der i-ten Stelle p_i zur j-ten Transition t_j. Umgekehrt führen genau $\mathbb{B}_{i,j}$ Kanten von der j-ten Transition t_j zur Stelle p_i.

Stellen, die durch eine Kante zu einer Transition führen, heissen *Eingabestellen*. Umgekehrt werden Stellen, die von einer Transition durch eine Kante führen, als *Ausgabestellen* bezeichnet. Die Ablauf eines PTN kann damit durch die folgenden beiden Regeln definiert werden:

1. Eine Transition gilt als *aktiviert*, sofern alle Eingabestellen mindestens eine Marke enthalten.
2. Eine Transition *konsumiert* eine Marke von allen Eingabestellen und *erstellt* beziehungsweise *erzeugt* eine Marke in allen Ausgabestellen[19].

Durch diese beiden Regeln kommt es bei Petri-Netzen zu einer Serie von aufeinanderfolgende *Zustandsänderungen*, wobei die Anzahl an Marken in den Stellen als *Zustand* bezeichnet wird. Dafür definieren *Bause und Kitzinger* den *Zustandsraum* $\mathbb{N}^P$. Der Zustand wird durch die Abbildung $s : P \to \mathbb{N}$ ausgedrückt. Der initiale Zustand s_0 (beziehungsweise M_0 in Definition 3) ist demnach $s_0 \in \mathbb{N}^P$. Die Abbildung s wird auch dafür genutzt, Zustandsänderungen formal zu beschreiben [BK02a, S.86]. Dabei werden die beiden oben genannten Regeln angewandt, womit eine Folgezustand hergestellt wird. Die formale Definition der Zustandsänderungen ist recht umfangreich, daher muss auf externe Literatur verwiesen werden, zum Beispiel *Bause und Kitzinger* und *Priese und Wimmel* ([BK02a, S.88 ff.], [PW08, S.52 ff.]).

Zur Illustration eines PTN kann das Beispiel der Übertragung einer Nachricht zwischen einem Sender und einem Empfänger herangezogen werden [BK02a]. Der Ablauf dieser Übertragung ist in Abbildung 2.10 jeweils für den Sender und dem Empfänger dargestellt. Im Prinzip handelt es

[19] In den Beschreibung von *Bause und Kitzinger* sowie *Priese und Wimmel* ([BK02a, PW08]) wird der Begriff des "Feuerns" genutzt. Das bedeutet, dass eine Transition feuert, sobald alle Voraussetzungen erfüllt sind.

sich um die Übertragung einer Nachricht, die der Sender vorbereitet und dann an den Empfänger übermittelt. Er validiert die Nachricht und sendet eine Bestätigung an den Sender zurück. Dabei kommunizieren Sender und Empfänger über jeweils einen unidirektionalen Datenkanal. Um Ressourcen zu sparen, können die Datenkanäle nur eine Nachricht speichern.

Das Sender-Empfänger-Beispiel zeigt auch, dass einige spezifische Objekte dieses Systems nicht von Belang sind, zum Beispiel der Inhalt der übertragenen Nachricht. Daher interessieren nur die Aktionen, die den Zustand des Sender und des Empfängers ändern. Diese Zustandsänderungen sind in Abbildung 2.10 als t_1 bis t_4 auf Sender- und als t_5 bis t_8 auf Empfängerseite zu erkennen. Abbildung 2.11 auf Seite 42 zeigt das Sender-Empfänger-Beispiel als PTN beziehungsweise als System. Die Stellen p_9 und p_{10} stellen hierbei die unidirektionalen Kanäle dar.

while *true* **do**
t_1 prepare message;
t_2 send message to receiver via channel1;
t_3 read acknowledgement of receiver on channel2;
t_4 interpret acknowledgement;
end

(a) Sender

while *true* **do**
t_5 read message from sender on channel1;
t_6 interpret message;
t_7 prepare acknowledgement;
t_8 send acknowledgement to sender via channel2;
end

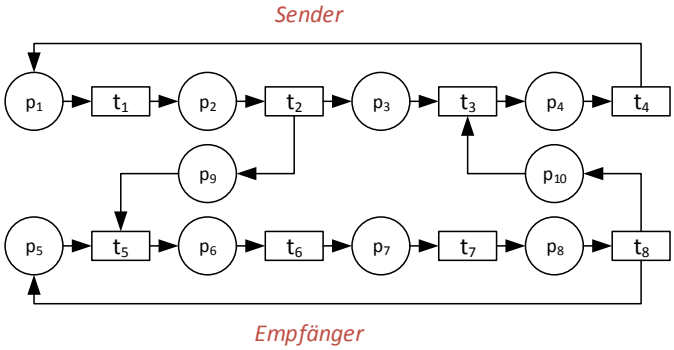
(b) Empfänger

Abbildung 2.10: Pseudocode der Übertragung einer Nachricht von einem Sender zu einem Empfänger mit Bestätigung nach [BK02a]

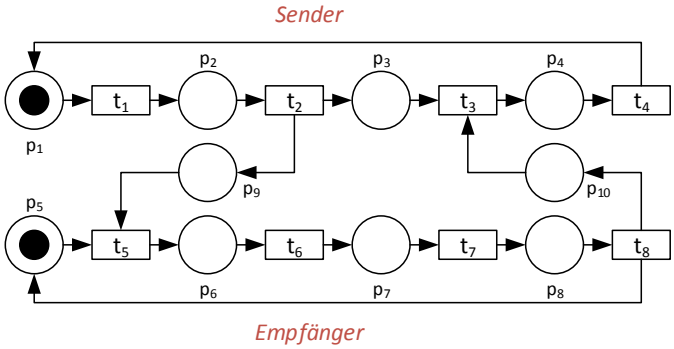
Formal kann auf Basis von Definition 3 das Sender-Empfänger-Beispiel folgendermaßen als $PN = (P, T, I^+, I^-, M_0)$ definiert werden, wobei

- $P = \{p_1, p_2, p_3, p_4, \ldots, p_{10}\}$
- $T = \{t_1, t_2, t_3, t_4, \ldots, t_8\}$
- $I^-(p_1, t_1) = 1$, $I^-(p_2, t_2) = 1$, $I^-(p_3, t_3) = 1$, $I^-(p_4, t_4) = 1$, $I^-(p_5, t_5) = 1$, $I^-(p_6, t_6) = 1$, $I^-(p_7, t_7) = 1$, $I^-(p_8, t_8) = 1$, $I^-(p_9, t_5) = 1$, $I^-(p_{10}, t_3) = 1$. Alle anderen Belegungen von I^- sind 0.
- $I^+(p_2, t_1) = 1$, $I^+(p_3, t_2) = 1$, $I^+(p_4, t_3) = 1$, $I^+(p_1, t_4) = 1$, $I^+(p_6, t_5) = 1$, $I^+(p_7, t_6) = 1$, $I^+(p_8, t_7) = 1$, $I^+(p_5, t_8) = 1$, $I^+(p_{10}, t_8) = 1$, $I^+(p_9, t_2) = 1$. Alle anderen Belegungen von I^+ sind 0.
- $M_0(p) = \begin{cases} 1 & \textit{wenn } p \in \{p_1, p_5\} \\ 0 & \textit{sonst} \end{cases}$

Insgesamt stellen PTNs gute Möglichkeiten bereit, die Dynamik von Systemen zu modellieren und zu simulieren. Für einfache Systeme, die nur wenige Stellen und Transitionen besitzen, sind sie übersichtlich und eingängig zu verstehen. Weit mehr problematisch sind komplexere Systeme. Ein gutes Beispiel dafür ist die Modellierung des exklusiven Zugriffes auf eine Resource.



(a) als PTN ohne initiale Marken

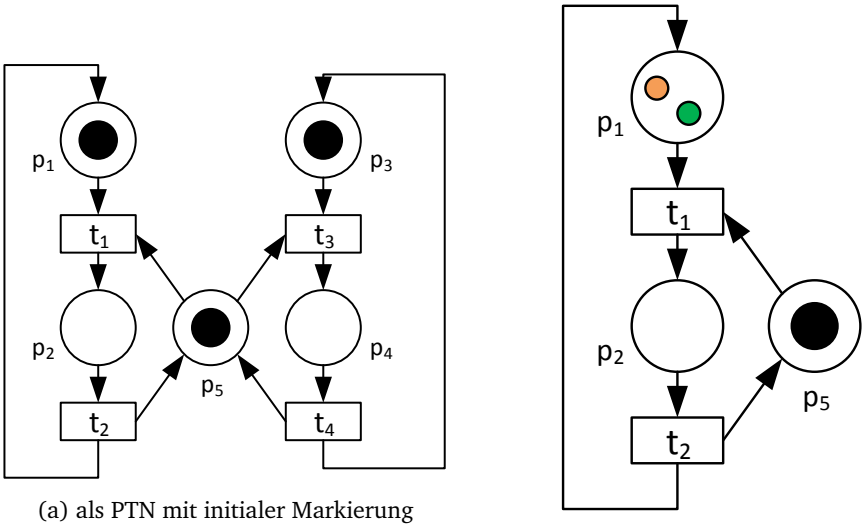


(b) als System (PTN mit initialen Marken)

Abbildung 2.11: Darstellung des Pseudocodes aus Abbildung 2.10 als PTN nach [BK02a]

Zur Illustration der Modellierung des exklusiven Zugriffes auf eine Ressource sind in Abbildung 2.12(a) auf Seite 43 zwei Prozesse dargestellt, die jeweils exklusiven Zugriff auf eine Ressource erlangen wollen. Die Ressource wird durch Stelle p_5 symbolisiert. Die beiden Prozesse werden durch die Stellen p_1 und p_2, p_3 und p_4 sowie deren Transitionen t_1 bis t_4 repräsentiert. Wie bereits erwähnt, ist die Darstellung des exklusiven Zugriffes für zwei Prozesse übersichtlich. Sobald allerdings Dutzende oder Hunderte Prozesse modelliert werden sollen, die ihren exklusiven Zugriff synchronisieren wollen, wird das Modell sowohl unübersichtlich als auch unperformant, wenn es zu einer Simulation kommt. Letzteres hat den Grund, dass zur Modellierung eines Prozess in Abbildung 2.12(a) jeweils zwei Stellen und zwei Transitionen gebraucht werden. Daher wachsen die Mengen P und T schnell an.

Zudem zeigt Abbildung 2.12(a) auch ein fundamentales Problem bei PTNs. In dieser Abbildung ist erkennbar, dass sowohl Transition t_1 als t_3 aktiviert sind. Nun ist fraglich, welche dieser beiden Transition als erstes die verfügbaren Marken konsumiert und eine neue Marke in die entsprechenden Ausgabestellen erstellt.



(a) als PTN mit initialer Markierung

(b) als CPN mit initialer Markierung

Abbildung 2.12: Darstellung des exklusiven Zugriffes auf eine Ressource p_5 durch zwei Prozesse als PTN nach [BK02a]

Farbige Petri-Netze

Farbige Petri-Netze (sogenannte *Coloured Petri Nets*, kurz CPN) erweitern Petri-Netze dadurch, dass die vormals monochromen Marken durch Farben unterschieden werden können. CPNs wurden von *Kurt Jensen* 1981 vorgestellt [Jen81][BK02a, S.119]. Sie dienen in erster Linie dazu, sehr große Petri-Netze zu generalisieren und zu vereinfachen. Dabei werden gleichlautende Teilgraphen des Graphen eines PT-Netzes auf einen Teilgraphen reduziert, aber dennoch durch verschieden farbige Marken unterschieden. Somit geht die ursprüngliche Semantik nicht verloren [Jen81].

Eine solche Reduktion ist in Abbildung 2.12 dargestellt. Dabei ist in der linken Teilabbildung 2.12(a) der ursprüngliche Graph des PT-Netzes abgebildet. Dabei bilden die Stellen p_1 und p_2 zusammen mit den Transitionen t_1 und t_2 einen Teilgraphen. Der andere Teilgraph setzt sich aus den Stellen p_3 und p_4 sowie den Transitionen t_3 und t_4 zusammen. Beide Teilgraphen haben denselben Aufbau und können dadurch in einem CPN reduziert werden. Diese Reduktion ist in Teilabbildung 2.12(b) dargestellt, wobei beide Teilgraphen durch die farbigen Marken in Stelle p_1 symbolisiert werden.

Bevor eine formale Definition von CPNs gegeben werden, müssen zunächst sogenannte *Multimengen* eingeführt und beschrieben werden. Kurz gesagt agieren Multimengen wie normale Mengen, aber erlauben, dass Elemente mehrmals vorkommen dürfen. Es werden also Duplikate ermöglicht. Die formale Definition von Multimengen ist in Definition 4 gegeben (vgl. auch [PW08, S.246] und [BK02a, S.120]).

Definition 4: Multimengen
Eine Multimenge M über eine nicht-leere Grundmenge A ist eine Abbildung in die Menge der natürlichen Zahlen $\mathbb{N}_0$:

$$M \in [A \mapsto \mathbb{N}_0]$$

Die Multimenge M stellt somit eine Menge dar, in der die Elemente von A mehrfach vorkommen dürfen. Die Funktion $M(s) \in \mathbb{N}_0$ gibt an, wie oft das Element s in M vorkommt. Zusätzlich sei die Menge aller Multimengen der nicht-leeren Grundmenge A mit $A_{\mathfrak{M}}$ bezeichnet[20].

Nach *Bause und Kitzinger* ist ein farbiges Petri-Netz ein 6-Tupel *CPN* definiert [BK02a, S.121]:

Definition 5: Farbiges Petri-Netz (CPN)
Ein farbiges Petri-Netz ist ein 6-Tupel CPN

$$CPN = (P, T, C, I^-, I^+, M_0)$$

wobei

- *P eine endliche Menge von Stellen ist*
- *T eine endliche Menge von Transitionen ist*
- *P und T sind disjunkt: $P \cap T = \emptyset$*
- *C ist eine Abbildung aus der Menge $P \cup T$ in endliche, nicht-leere Mengen*[21]
- *I^+ und I^- sind die vorwärts und rückwärts gerichteten Kantenfunktionen aus der Menge $P \times T$, wobei gilt*
  $\forall (p,t) \in P \times T : I^-(p,t), I^+(p,t) \in [C(t) \to C(p)_{\mathfrak{M}}]$
- *M_0 ist die initiale Markenbelegung der Stellen, wobei gilt:*
  $\forall p \in P : M_0(p) \in C(p)_{\mathfrak{M}}$

Gegenüber der Definition eines PT-Netzes auf Seite 39 fallen die zusätzliche Abbildung C sowie die modifizierten Kantenfunktionen und die initiale Markenbelegung auf. Die Abbildung C ist in der Literatur uneinheitlich definiert. *Bause und Kitzinger* definieren diese Abbildung als Zuweisung von farbig unterscheidbaren Markenmengen an Stellen und Transitionen [BK02a]. Sie erweitert dadurch PT-Netze um die Möglichkeit, welche Eingabestellen welche farbigen Marken enthalten dürfen sowie welche farbigen Marken durch die Transitionen in andere farbige Marken verarbeitet werden dürfen, die dann in die Ausgabestellen abgelegt werden. Dementsprechend sind auch die Kantenfunktionen I^+ und I^- sowie die initale Markenbelegung M_0 in Definition 5 modifiziert. Nach *Priese und Wimmel* bezeichnet C keine Abbildung, sondern die Menge der insgesamt verfügbaren Farben [PW08]. Es werden dann zwei zusätzliche Abbildung $C_t : T \to C_{\mathfrak{M}}$ und $C_p : P \to C_{\mathfrak{M}}$ genutzt, um den Eingabe- und Ausgabestellen sowie den Transitionen die entsprechenden Eingabe- und Ausgabemengen an farbigen Marken zuzuweisen. Die Kantenfunktionen sind dergestalt modifiziert, dass sie dann Teilmengen von C nutzen. Innerhalb dieser Dissertation wird die Defintion von *Bause und Kitzinger* genutzt.

Es sei angemerkt, dass farbige Petri-Netze seit der erstmaligen Publikation 1981 ständig weiterentwickelt wurden. Beispiele dieser Weiterentwicklung sind zeitliche Beschränkungen bei

[20] In der Literatur ist die Bezeichnung der Menge aller Multimengen uneinheitlich. Zusätzlich sei auch nach [BK02a, S.120] die Addition von Multimengen sowie die Multiplikation mit normalen Zahlen wie folgt definiert: $\forall m_1, m_2 \in A_{\mathfrak{M}}$ und $r \in \mathbb{R}$ gilt $(m_1 + m_2)(s) := m_1(s) + m_2(s)$ sowie $(r \cdot m_1) := r \cdot m_1(s)$.

[21] Die endlichen, nicht-leeren Mengen bezeichnen hierbei die Mengen der insgesamt verfügbaren Farben.

den Transitionen, eine erweiterte Notation bei grafischen Repräsentation zum Zwecke der Unterstützung von Programmierhochsprachen sowie die Möglichkeit, Hierarchien zu nutzen. Da die Erläuterung dieser Erweiterungen den Rahmen dieser Dissertation sprengen würde, sei auf externe Literatur verwiesen. Dabei seien die Publikationen von *Kristensen et al.* sowie *Jensen und Kristensen* als besonders lesenswert zu empfehlen [KCJ98, JK07].

(Generalisierte) Stochastische Petri-Netze

Sowohl das originale *Place-Transition*-Netz als auch farbige Petri-Netze beinhalten keinerlei Möglichkeiten, das Zeitverhalten eines Systems abzubilden. Das bedeutet, es gibt keine Konvention *wann* eine Transition feuert. Somit ist nur eine funktionale und qualitative Analyse eines Systems möglich, aber eine quantitative oder Performance-Analyse ist unmöglich [BK02a, S.133]. Daher gab es laut *Bause und Kitzinger* zwei Ansätze, das Zeitverhalten eines Systems in Petri-Netzen zu beschreiben [BK02a, S.133]:

1. **Definition der Verweildauer von Marken in den Stellen** Wenn Marken eine Stelle $p \in P$ erreichen, dann sind sie für eine bestimmte Zeit nicht verfügbar. Sobald diese Verweildauer abgelaufen ist, werden sie wieder als verfügbar markiert und können von p's nachfolgenden Transitionen konsumiert werden.
2. **Definition von Transitionszeiten bei aktivierten Transitionen** Sobald eine Transition $t \in T$ aktiviert ist, feuert sie nach einer bestimmten Zeit. Dabei werden zwei Varianten unterschieden:
   - (Vorselektierung der Marken) Sobald die Transition t aktiviert ist, markiert sie alle zu konsumierende Marken für die übrigen Transitionen als nicht verfügbar. Nachdem die Transitionszeit abgelaufen ist, feuert die Transition t. Das bedeutet, dass die vormals markierten Marken konsumiert und unter Umständen neue Marken in die Ausgabestellen erstellt werden.
   - (Konkurrierende Transitionen) Nachdem die definierte Transitionszeit von t abgelaufen ist, wird geprüft, ob t noch aktiviert ist. Das bedeutet, es werden die Vorbedingungen zum Feuern erneut geprüft. Sofern dies gegeben ist, werden die entsprechenden Marken konsumiert und unter Umständen neue in die Ausgabestellen erzeugt. Es ist ersichtlich, dass Transitionen mit kürzeren Transitionszeiten durchaus andere, vormals aktivierte Transitionen deaktivieren können, womit diese nicht feuern können. Insgesamt konkurrieren aktivierte Transitionen um die Marken.

Ein Derrivat eines *Place-Transition*-Netzes (PTN), das das Zeitverhalten eines Systems beinhaltet, sind nach *Bause und Kitzinger* sogenannte *Stochastische Petri-Netze* (kurz SPN, [BK02a, S.135]). Sie ergänzen die formalen Definition eines PTN laut Definition 3 von Seite 39 um die Transitionszeiten, die durch eine zusätzliche Menge $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_m\}$ angegeben werden. Die Elemente λ_i der Menge Λ geben dabei die Transitionszeiten jeder Transition t_i an. Genauer gesagt, der Zeitpunkt des möglichen Feuerns einer Transition ist exponentiell verteilt. Die Exponentialverteilung jeder Transition t_i wird durch die Zufallsvariable χ_i angegeben:

$$F_{\chi_i}(x) = 1 - e^{-\lambda_i x}$$

Da keine Vorselektierung der Marken durch SPNs vorliegt, konkurrieren die Transitionen um die Marken. Dieser Fakt ist wichtig, da unterschiedliche Zustände entstehen können, je nachdem, welche Transitionen in einer Konkurrenzsituation feuern und welche dadurch am Feuern gehindert werden. Somit sind Zustandsübergänge von Wahrscheinlichkeiten abhängig, die durch

die zusätzliche Menge Λ angegeben werden. Insgesamt kann aber gezeigt werden, dass die Zustandsübergänge durch sogenannte *Markov-Ketten* dargestellt und errechnet werden können. Der Beweis sowie Beispiele dafür sind recht umfangreich. Daher muss auf externe Literatur verwiesen werden, zum Beispiel die sehr gute Erklärung von *Bause und Kitzinger* in [BK02a, S.136 ff.]. Beide Autoren erklären auch sehr gut *Markov-Ketten* in [BK02a, S.25 ff.].

Ein weiteres, weiter fortgeschrittenes Derrivat eines *Place-Transition*-Netzes (PTN) stellen sogenannte *Generalisierte Stochastische Petri-Netze* (kurz GSPN) dar. Formal agieren sie wie die vorher beschriebenen stochastischen Petri-Netze (SPNs), aber sie unterscheiden zwischen zeitverzögerten Transitionen und Transitionen ohne Zeitverzug. Erstere feuern zeitgesteuert, wie es für SPNs beschrieben wurde. Zweitere feuern sofort, also wie bei PTNs ohne Zeitverzug. Dementsprechend können GSPN als eine Mischung zwischen PTNs und SPNs hinsichtlich der Transitionszeiten angesehen werden. Diese Unterscheidung macht sich auch in der grafischen Notation von GSPNs bemerkbar. Sie ist in Abbildung 2.13 dargestellt: zeitverzögerte Transitionen werden als nicht-ausgefüllte Rechtecke und Transitionen ohne Zeitverzug als ausgefüllte Rechtecke symbolisiert.

In Abbildung 2.13 wird zudem ein Produzenten-Konsumenten-System als GSPN nach [BK02a, S.134] gezeigt. Dabei erzeugt ein Produzent genau einen Gegenstand, der von einem Konsumenten wieder verbraucht wird. Dieser Gegenstand wird als Marke symbolisiert. Zwischen dem Produzenten und Konsumenten kommen Puffer zu Einsatz, die in der Mitte von Abbildung 2.13 als Stellen dargestellt sind, die mit den beiden Transitionen t_1 und t_4 durch Kanten verbunden sind. Der eigentliche Prozess des Produzierens und des Konsumierens wird durch die Transitionen t_3 und t_6 in Abbildung 2.13 dargestellt. Da angenommen wird, dass diese Prozesse gegenübern der Pufferung mehr Zeit benötigen wird, sind diese beiden Transition als zeitverzögert modelliert.

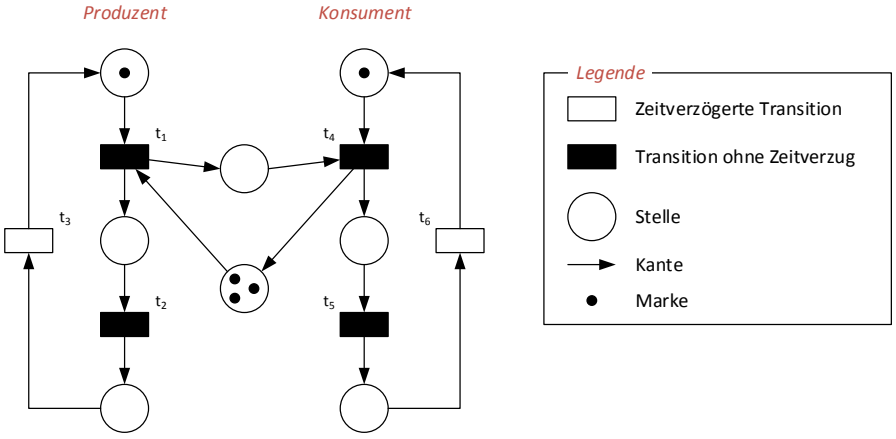


Abbildung 2.13: Darstellung eines Produzenten-Konsumenten-Systems als GSPN nach [BK02a, S.143]

Wie bereits erwähnt, stellen GSPNs eine Mischung zwischen PTNs und SPNs dar. Daraus resultiert eine andere Verfahrensweise hinsichtlich der Zustandsübergänge. Zur Illustration sei M die Zustandsübergangsfunktion, die den Zustand vor und nach einem Zustandsübergang wiedergibt. Das bedeutet, dass die Markenbelegungen der Stellen als Funktionswert erhalten werden können. Sei ferner $EN_T(M)$ eine Abbildung, die zu einem Zustand M die Menge der aktivierten Transitionen liefert. Nach [BK02a, S.144] ergeben sich folgende Verfahrensweisen:

1. Enthält $EN_T(M)$ ausschließlich zeitverzögerte, aktivierte Transitionen, so verhält die Zustandsübergangsfunktion M wie ein stochastisches Petri-Netz. Das bedeutet, dass die gegebenen Feuerwahrscheinlichkeiten der Transitionen mittels *Markov-Ketten* errechnet werden können. Die Feuerwahrscheinlichkeiten ergeben sich für jede Transition $t_i \in EN_T(M)$ durch
$$\frac{\lambda_i}{\sum_{j:t_j \in EN_T(M)} \lambda_j}$$
2. Enthält $EN_T(M)$ ausschließlich aktivierte Transitionen ohne Zeitverzug, so muss spezifiziert werden, mit welcher Wahrscheinlichkeit sie feuern können. Da eine Wahrscheinlichkeitsverteilung immer davon abhängt, welche aktivierten Transitionen ohne Zeitverzug in M zu dieser beitragen, ist eine statische Analyse sehr schwierig. Ein einfachere Lösung ist es, *Gewichtungen* für diese Transitionen einzuführen. Seien dafür $t_i, t_j \in EN_T(M)$ zwei aktivierte Transitionen ohne Zeitverzug und w_i sowie w_j die dazugehörigen Gewichtungen, so ergibt sich eine Feuerwahrscheinlichkeit für t_i von
$$\frac{w_i}{w_i + w_j}$$
3. Enthält $EN_T(M)$ sowohl aktivierte Transitionen mit als auch ohne Zeitverzug, so haben Transition ohne Zeitverzug Vorrang beim Feuern gegenüber denjenigen mit Zeitverzug. Die Evaluation, welche Transition feuert, ist äußerst komplex und erfordert unter anderem die Betrachtung der Erreichbarkeiten der Ein- und Ausgabestellen sowie eine stochastische Betrachtung der Feuerwahrscheinlichkeiten. Die formale Erläuterung dieser Evaluation würde den Rahmen dieser Dissertation sprengen. Daher sei auf externe Literatur verwiesen, zum Beispiel die Publikation von *Bause und Kitzinger* in [BK02a, S.145 ff.], in der sowohl GSPNs qualitativ und quantitativ erläutert als auch Beispiele gegeben werden.

Formal ist ein generalisiertes stochastisches Petri-Netz nach [BK02a, S.144] als Quadrupel *GSPN* definiert:

Definition 6: Generalisiertes Stochastisches Petri-Netz (GSPN)
Ein GSPN ist ein Quadrupel GSPN

$$GSPN = (PN, T_1, T_2, W)$$

wobei

- $PN = (P, T, I^-, I^+, M_0)$ *das zugrunde liegende Place-Transition-Netz ist*
- $T_1 \subseteq T, T_1 \neq \emptyset$ *ist die nicht-leere Menge an verzögerten Transitionen*
- $T_2 \subset T$ *ist die Menge an Transitionen ohne Zeitverzug, wobei* $T_1 \cap T_2 = \emptyset$ *und* $T = T_1 \cup T_2$
- W *bezeichnet ein Tupel* $W = (w_1, w_2, \ldots, w_{|T|})$*, wobei die Komponenten* $w_i \in \mathbb{R}^+$ *folgende Bedeutung haben:*
  - w_i *bezeichnet die Feuerrate einer negativen Exponentialverteilung, wenn* t_i *eine Transition mit Zeitverzug ist (*$t_i \in T_1$*) oder*
  - w_i *bezeichnet die Gewichtung beim Feuern einer Transition* t_i *ohne Zeitverzug (*$t_i \in T_2$*)*

Zu beachten ist, dass wenn $T_2 = \emptyset$*, dann verhält sich das GSPN wie ein SPN, also SPNs* $\subset$ *GSPNs.*

Insbesondere dem Tupel W in Defintion 6 kommt besondere Bedeutung zu. Durch dessen Definition ist es möglich, die Eigenschaften von PTNs und SPNs zu vereinen. Aus diesem Grund spricht man auch von einem *generalisierten* stochastischen Petri-Netz (GSPN).

Zusätzlich muss noch angemerkt werden, dass GSPNs durchaus auch mit farbigen Marken gemäß der Definition 5 eines farbigen Petri-Netzes auf Seite 44 genutzt werden können. In diesem Fall spricht man von einem *farbigen GSPN* (sogenanntes *Coloured GSPN*, kurz CGSPN).

Warteschlangen-Petri-Netze

Das bis dato jüngste Derrivat von *Place-Transition*-Netzen sind Warteschlangen-Petri-Netze (sogenannte *Queued Petri Nets*, kurz QPN). Sie basieren auf farbigen generalisierten stochastischen Petri-Netzen (CGSPNs), aber teilen Stellen in zwei Arten auf: Stellen mit Warteschlangen und gewöhnliche Stellen, wie sie bei CGSPNs enthalten sind. Der Grund dafür ist, dass man Konzepte, Verfahren und Semantiken der Warteschlangen-Theorie zwar mittels CGSPNs zu modellieren vermag, aber die entstandenen Modelle komplex und schwierig zu simulieren sind [BK02a, S.166].

Die Unterscheidung der Stellen in diejenigen mit Warteschlangen und diejenigen ohne wird auch in der grafischen Repräsentation eines QPNs deutlich gemacht. Stellen ohne Warteschlangen werden weiterhin als Kreise dargestellt und Stellen mit Warteschlange werden durch einen Kreis mit senkrechtem Strich symbolisiert. Letzteres ist in Abbildung 2.14 dargestellt. Nach *Bause und Kitzinger* sowie *Kounev und Buchmann* bestehen Stellen mit Warteschlangen aus zwei Komponenten: der Warteschlange, in der eingehende Marken gespeichert werden und einem Ausgabebereich, in denen Marken gespeichert werden, sobald sie durch die Warteschlange verarbeitet wurden [BK02a, S.166],[KB03]. Beide Komponenten sind in Abbildung 2.14 auf der linken Seite dargestellt.

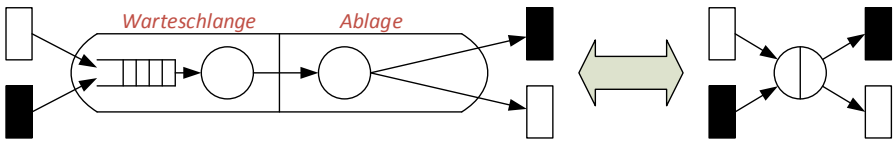


Abbildung 2.14: Interna einer Stelle mit Warteschlange und dessen Symbol nach [Kou08]

Eingehende Marken werden anhand der definierten Eigenschaften der Warteschlange in diese einsortiert. Marken, die sich bereits in der Warteschlange befinden, stehen für nachfolgende Transitionen nicht zur Verfügung. Das ist erst möglich, nachdem die Marken durch die Warteschlange verarbeitet und im Ablagebereich gespeichert wurden.

Zur Beschreibung der Eigenschaften der eingesetzten Warteschlange wird im allgemeinen die sogenannte *Kendall*-Notation verwendet. Nach *Bause und Kitzinger* erfolgt diese Notation in einer durch Schrägstriche getrennten Schreibweise "*A/B/m/K/Z/Sched*" [BK02a, S.60], wobei

- *A* den Ankunftsprozess als statistische Verteilung beschreibt
- *B* den Serviceprozess als statistische Verteilung beschreibt
- *m* die Anzahl der Serviceeinheiten für den Serviceprozess angibt, wobei $m \geq 1$
- *K* die maximale Warteschlangengröße angibt

- Z die Populationsgröße angibt (maximale Anzahl an Ankünften)
- *Sched* die Servicestrategie angibt (im Sinne eines *schedulers*)

Für den Ankunfts- und Serviceprozess als auch für die Servicestrategie haben sich einige Abkürzungen durchgesetzt. Sie sind in Tabelle 2.3 aufgelistet[22]. So bedeutet eine *Kendall*-Notation `D/M/2/10/`∞`/LIFO` für ein Warteschlangensystem konstante Ankunftszeiten, exponentiell verteile Servicezeiten, 2 Serviceeinheiten, maximal 10 Warteschlangenplätze, einer beliebig großen Population und eine Servicestrategie nach dem LIFO-Prinzip.

Abkürzung	Bedeutung
M	Exponentialverteilung
D	Deterministische Verteilung
H	Hyperexponentialverteilung
E_k	Erlang-Verteilung
PH	Phasenverteilung
G, *GI*	Beliebige Verteilung

(a) für Ankunfts- und Serviceprozess

Abkürzung	Bedeutung
FIFO, *FCFS*	First-in-First-out
LIFO, *LCFS*	Last-in-First-out
SJN	Shortest-Job-Next
RANDOM	zufällig
CPU, *PS*	Prozess-Scheduler
PRIO	Prioritäts-basiertes Scheduling

(b) für Servicestrategie

Tabelle 2.3: Häufig genutzte Abkürzungen in der *Kendall*-Notation

Es sei an dieser Stelle angemerkt, dass die theoretischen Betrachtung sowie die formalen Beschreibungen von Warteschlangen komplex sind und aus Platzgründen nicht in dieser Dissertation behandelt werden können. Es sei daher auf die sehr gute Einführung in die Warteschlangen-Theorie von *Bause und Kitzinger* in [BK02a, S.58-76] verwiesen. *Buchholz et al.* erläutern in [BKF14] die verschiedenen Verteilungen, die bei Warteschlangen zum Einsatz kommen können.

Die formale Definition eines Warteschlangen-Petri-Netzes ist in der Literatur uneinheitlich. *Bause und Kitzinger* nutzen in [BK02a, S.167] zum Beispiel ein Tripel $QPN = (CGSPN, P_1, P_2)$, wobei

- *CGSPN* das zugrunde liegende farbige generalisierte stochastische Petri-Netz darstellt
- P_1 stellt die Menge der Stellen dar, die eine Warteschlange besitzen
- P_2 bezeichnet die Menge der gewöhnliche Stellen ohne Warteschlange

Dabei gilt, dass $P_1 \cap P_2 = \emptyset$ und $P = P_1 \cup P_2$. Zu beachten ist, dass sich ein QPN wie ein CGSPN verhält, wenn $P_1 = \emptyset$. Das bedeutet, dass CGSPN $\subset$ QPN.

Kounev und Buchmann verwenden hingegen ein 8-Tupel [KB03]. Innerhalb dieser Dissertation wird diese Defintion aufgrund der präziseren Erklärung eines Warteschlangen-Petri-Netz verwendet.

[22] Die Auflistung erhebt keinen Anspruch auf Vollständigkeit.

Definition 7: Warteschlangen-Petri-Netz (QPN)
Ein Warteschlangen-Petri-Netz ist ein 8-Tupel QPN

$$QPN = (P, T, C, I^-, I^+, M_0, Q, W)$$

wobei

- *CPN* $= (P, T, C, I^-, I^+, M_0)$ *das zugrunde liegende farbige Place-Transition-Netz ist*
- $Q = (q_1, q_2, \ldots, q_{|P|})$ *bezeichnet ein Tupel, dessen Komponenten* q_i
  - *die Warteschlange gemäß der Kendall-Notation beschreiben und alle Farben von* $C(P)$ *berücksichtigen, sofern es bei* p_i *um eine Stelle mit Warteschlange handelt oder*
  - *mit Schlüsselwort* `untimed` *übereinstimmen, wenn* p_i *eine gewöhnliche Stelle ohne Warteschlange ist*
- $W = (w_1, w_2, \ldots, w_{|T|})$ *bezeichnet ein Tupel von Abbildungen, dessen Komponenten* w_i *auf* $C(t_i)$ *definiert sind und* $\forall c \in C(t_i) : w_i(c)$ *gilt:*
  - w_i *ist die Beschreibung einer Wahrscheinlichkeitsverteilungsfunktion, die die Feuerverzögerung für die Farbe* $c \in C(t_i)$ *angibt, sofern es sich bei* t_i *um eine zeitverzögerte Transition handelt oder*
  - w_i *gibt die Gewichtung an, die die relative Feuerfrequenz für die Farbe* $c \in C(t_i)$ *beschreibt, wenn* t_i *eine Transition ohne Zeitverzug ist*

Kounev und Buchmann geben auch ein Beispiel für ein QPN-Modell für ein zentrales Serversystem in [KB03], das in Abbildung 2.15 auf Seite 51 dargestellt ist. In dieser Abbildung repräsentiert Stelle p_2 ein oder mehrere Benutzerterminals, an denen die Benutzer Aufgaben im Sinne von Kommandozeilenprogrammen auf dem zentralen Serversystem starten können. Diese Aufgaben werden mit Marken symbolisiert, die in Abbildung 2.15 die Bezeichnung “o” tragen. Zusätzlich sei für Stelle p_2 festgelegt, dass zwischen zwei Aufgaben eines Benutzers eine gewisse Pause, also eine Zeitverzögerung, sein soll[23]. Die von den Benutzern gestarteten Aufgaben werden dann vom zentralen Serversystem ausgeführt, das durch die Stelle p_3 mit der Bezeichnung “CPU” repräsentiert wird. Zur Ausführung einer Aufgabe auf dem zentralen Serversystem wird Arbeitsspeicher benötigt, dass durch die Stelle p_1 in Abbildung 2.15 repräsentiert wird sowie durch Marken mit der Bezeichnung “m” symbolisiert werden. Die Ausführung der Aufgaben resultiert in der Benutzung des I/O-Subsystems des zentralen Serversystems, das durch die Stellen p_5 und p_6 in Abbildung 2.15 repräsentiert wird. Der Einfachheit halber wird angenommen, dass für die Ausführung einer Aufgabe immer dieselbe Menge an Hauptspeicher nötig ist. Daher wird im QPN-Modell, das in Abbildung 2.15 dargestellt ist, immer nur eine Marke mit der Bezeichnung “m” benötigt. Nach der Ausführung einer Aufgabe werden alle genutzten Marken wieder in ihre ursprünglichen Stellen (p_1 und p_2) überführt. Damit stehen sie dann für die Ausführung neuer Aufgaben durch die Benutzer zur Verfügung.

[23] Im Original in [KB03] heisst es “*user start jobs after a certain thinking time.*”

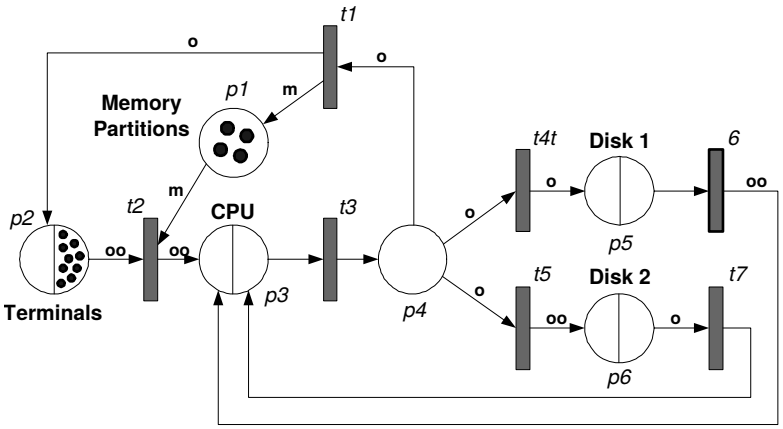


Abbildung 2.15: QPN-Modell eines zentralen Serversystems mit Beschränkung des Haupt- und Arbeitsspeichers nach [KB03]

In Abbildung 2.15 sind die Eigenschaften der Warteschlangen in den Stellen p_2, p_3, p_5 und p_6 nicht dargestellt. Sie können aber aus der formalen Definition für das QPN-Modell abgelesen werden [KB03]. Basierend auf Defintion 7 lautet sie:

$$QPN = (P, T, C, I^-, I^+, M_0, Q, W)$$

- wobei $CPN = (P, T, C, I^-, I^+, M_0)$ das zugrunde liegende farbige Place-Transition-Netz ist, wie es in Abbildung 2.15 dargestellt ist
- $Q = (\texttt{untimed}, -/M/\infty - IS, -/M/1 - PS, \texttt{untimed}, -/M/1 - FCFS, -/M/1 - FCFS)$
- $W = (w_1, w_2, \ldots, w_{|T|})$, wobei alle Transitionen ohne Zeitverzug feuern und wobei gilt: $\forall c \in C(t_i) : w_i(c) := 1$, womit die Feuerwahrscheinlichkeit aller Transitionen gleichwahrscheinlich ist.

Hierarchische *Queued Petri Nets*

Nach *Kounev und Buchmann* ist es schwierig, mit aktuellen Techniken eine quantitative Analyse von QPNs durchzuführen [KB03]. Die Techniken basieren zumeist auf *Markov-Ketten* und sind damit für ein Problem anfällig, das als *state space explosion* bezeichnet wird. Damit wird gemeint, dass mit zunehmender Anzahl von Stellen und Marken in einem QPN-Modell der Zustandsraum der zugrunde liegende *Markov-Kette* exponentiell anwächst. Das überfordert die Rechen- und Speicherkapazitäten von selbst aktuellen Computern und zwingt Anwender dazu, ihre QPN-Modelle hinsichtlich der Anzahl der Stellen und Marken zu limitieren. Damit wird die Modellierung und Analyse von komplexen System unmöglich.

Ein Verfahren, um das oben genannten Problem abzumildern, ist die hierarchische Gliederung des QPN-Modells. Solche derart untergliederte QPN-Modelle werden als *Hierarchically-Combined Queuing Petri Nets* (kurz HQPNs) bezeichnet, die 1994 von *Bause et al.* formalisiert wurden [BBK94]. Der Hauptgedanke ist, dass es in HQPNs neben gewöhnlichen Stellen und Stellen mit Warteschlangen eine zusätzliche Art von Stellen gibt, die anstatt einer Warteschlange ein inneres QPN-Modell besitzt. Eine solche Stelle wird als sogenanntes *subnet place* bezeichnet und ist in

Abbildung 2.16 dargestellt. Das Symbol dafür ist ein Kreis mit doppelter Umrandung und einem senkrechten Strich.

Eine Stelle mit einem inneren Modell hat einen spezifischen Aufbau. Alle Marken, die diese Stelle erreichen, werden in einer gewöhnlichen Stelle abgelegt, die in Abbildung 2.16 die Bezeichnung “Input” trägt. Von dieser Stelle aus stehen die Marken dann dem eigentlichen inneren Modell zur Verfügung, was in Abbildung 2.16 mit “user specified part of the subnet” bezeichnet wird. Dieses innere Modell kann ein QPN gemäß Definition 7 oder ein weiteres HQPN-Modell sein. Letzteres erlaubt prinzipiell unendlich viele Stufen der Verschachtelung beziehungsweise Hierarchieebenen. Alle Marken, die aus dem inneren Modell stammen, werden in einer gewöhnlichen Stelle abgelegt, die in Abbildung 2.16 mit “Output” annotiert ist. Eine Besonderheit ist die gewöhnliche Stelle mit der Bezeichnung “Actual Population” in Abbildung 2.16. In diese Stelle werden automatisch alle eingehenden Marken gespeichert und dienen nach [KB03] nur dazu, die Gesamtanzahl aller Marken zu ermitteln und aktuell zu halten. Insgesamt hat eine Stelle mit einem inneren Modell eine ähnliche Semantik wie Stellen mit einer Warteschlange: eingehende Marken werden in einem Eingangsbereich und ausgehende Marke in einem Ausgabebereich abgelegt. Anstatt einer Warteschlange wird aber ein inneres Modell ausgeführt, das wie bereits erwähnt entweder ein gewöhnliches QPN- oder ein HQPN-Modell sein kann.

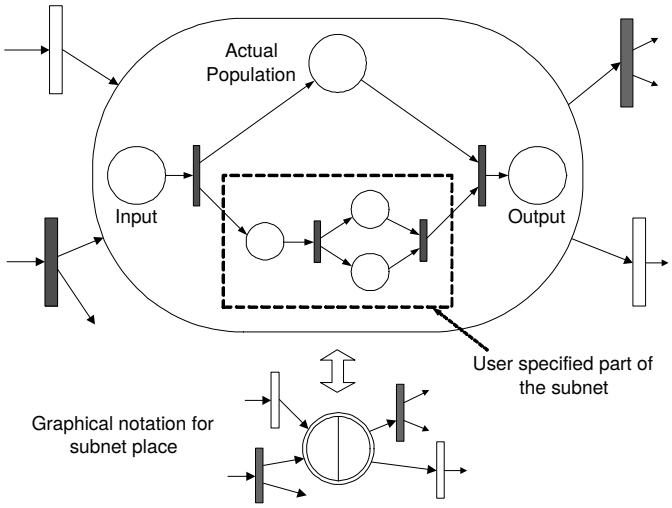


Abbildung 2.16: Stelle mit innerem Modell und dessen Symbol nach [KB03]

Der Vorteil von Stellen mit inneren Modell ist, dass QPN-Modelle hierarchisch organisiert werden können. Durch dieser Untergliederung eröffnet sich die Möglichkeit, das Gesamtmodell effizienter numerisch analysieren zu können. Nach [KB03] werden dafür Verfahren der strukturierten Analyse verwendet. Damit können QPN-Modelle unterstützt und simuliert werden, die um eine Größenordnung größer sind als diejenigen, die konventionelle Verfahren nutzen.

3 Metriken von Datenbanksystemen

Im vorherigen Kapitel 2 wurden die grundlegenden Eigenschaften und Charakteristiken der Architektur von Datenbanksystemen erläutert. In diesem Kapitel sollen die Metriken untersucht und beschrieben werden, die dazu genutzt werden, um Datenbanksysteme zu vergleichen.

3.1 Leistung von Datenbanksystemen

Die Definition der "Leistung" eines Datenbanksystems kann im physikalischen Sinn nicht erfolgen, da es sich bei ihnen um eine Computersoftware handelt. Wie bei jeder anderen Computersoftware auch ist es die naheliegendste und einfachste Möglichkeit, eine Aufgabe zu definieren und die Computersoftware diese Aufgabe erledigen zu lassen, um dabei die Ausführungs- beziehungsweise Antwortzeit zu messen. Je nachdem, wie die Aufgabe definiert ist und welche Bemessungsgröße dabei genutzt wird, kann diese Größe dann durch Division in Relation zur Antwortzeit gesetzt werden.

Um diesen Sachverhalt zu illustrieren, kann sich eine Computersoftware gedacht werden, die eine einfache Textdatei einliest und verarbeitet, um schließlich eine finale Ausgabe zu erzeugen. Um die "Leistung" dieser Computersoftware zu quantifizieren und zu bewerten, kann als Aufgabe die Verarbeitung einer Textdatei mit beispielsweise 100.000 Zeilen Text definiert werden. Nun kann die Zeit gemessen werden, die die Computersoftware braucht, um diese 100.000 Textzeilen verarbeiten. Wenn sie zum Beispiel die Antwortzeit 10 Sekunden und die Bemessungsgröße die Gesamtanzahl der Textzeilen ist, so ergibt sich eine Metrik von 10.000 Zeilen pro Sekunde. Zudem kann man sich vorstellen, dass diese Computersoftware von verschiedenen Herstellern implementiert ist. Sofern dieselbe Arbeitslast und dieselbe technische Plattform für die Ausführung genutzt wird, so lassen sich die Antwortzeiten beider Implementation messen. Nutzt man dieselbe Metrik *Textzeilen pro Sekunde* als Maß für die Leistungsfähigkeit, so ermöglicht es einen direkten Vergleich.

Nun kann sich auch gedacht werden, dass die Computersoftware nicht nur einmal, sondern auch mehrfach simultan ausgeführt wird. Es kann angenommen werden, dass die Verarbeitungsleistung derselben einfachen Textdatei mit steigender Anzahl an Ausführungen aufgrund von Beschränkungen der technischen Plattform und des Betriebssystems sinken wird. Als Maß der Verarbeitungsleistung kann auch hier *Textzeilen pro Sekunde* verwendet werden, aber in Abhängigkeit der Anzahl an gleichzeitig ausgeführten Instanzen der Computersoftware. Ein solches Maß kann zum direkten Vergleich herangezogen werden werden, wenn man zum Beispiel den Mehrbenutzerbetrieb untersuchen möchte.

Da es sich bei Datenbanksystemen um eine spezalisierte Art von Computersoftware handelt, kann deren Bemessungsgrößen klar eingegrenzt werden und beschränken sich zumeist auf Datenmanipulations- und -abfrageoperationen. Genau diese Operationen gilt es mess- beziehungsweise bestimmbar zu gestalten. Dabei können zwei Bereiche zur Leistungsmessung unterschieden werden:

1. Die *Antwortzeit* gibt die Zeitdauer t an, die es insgesamt braucht, bis eine Datenbankoperation durch ein Datenbanksystem entgegengenommen, verarbeit und das Ergebnis an den Anfragenden zurückgegeben wurde. Der Fokus liegt in diesem Fall eindeutig auf der Zeit t und dient in den meisten Fällen der Evaluierung der maximalen Leistungsfähigkeit des Datenbanksystems unter optimalen Bedingungen.

2. Der *Durchsatz* gibt die Anzahl der gleichzeitigen Datenbankoperationen n an, die ein Datenbanksystem unabhängig von der Größe der Datenbank in einer Zeiteinheit t verarbeiten kann. Im Gegensatz zu Antwortzeit liegt hier der Fokus in der Evaluation des Lastverhaltens eines Datenbanksystems, zum Beispiel im Mehrbenutzerbetrieb.

Diese Art der Leistungsbestimmung wird im allgemeinen als *Performance* bezeichnet. Insgesamt ist die Performance eine durchaus geeignete Metrik zur Bestimmung und Vergleich der "Leistung" von Datenbanksystemen. Dabei ist zu beachten, dass dieser Metrik keinerlei phyikalische Einheit zugeordnet ist. Die Performance *Perf* kann wie folgt definiert werden:

$$\textit{Perf}(n,t) = \begin{cases} \frac{1}{t} & \textit{(Antwortzeit)}, n = 1 \\ \frac{n}{t} & \textit{(Durchsatz)} \end{cases} \tag{3.1}$$

In Gleichung 3.1 ist zu erkennen, dass der Performance *Perf* eine Doppelbedeutung zukommt, die sich aus der gestellten Aufgabe beziehungsweise der definierten Arbeitslast ergibt. Wird eine solche zwar definiert, aber nicht näher bestimmt, so liegt keine Quantifizierungsgröße vor. In diesem Fall ist $n = 1$ und nur die Antwortzeit kann bestimmt werden. Das kommt in der Praxis allerdings recht selten vor. Zumeist kann aus der Dokumentation die Bemessungsgröße abgelesen werden. Am Beispiel von relationalen Datenbanksystemen kann es sich zum Beispiel um eine festgelegte Anzahl an Tupeln handeln, die es zu verarbeiten gilt. Hierbei kann dieselbe Vorgehensweise wie im einleitenden Beispiel am Anfang dieses Textabschnittes genutzt werden. Für die Performance erhielte man dann als Metrik n Tupel in t Sekunden für die Antwortzeit und n Tupel pro t Sekunde für den Durchsatz, sofern t in Sekunden angegeben wird. Die Performance kann aber auch einen technischen Charakter haben, zum Beispiel die Anzahl der gelesenen oder geschriebenen Tupel pro Zeiteinheit auf den Massenspeicher. Insgesamt kann für die Performance eines Datenbanksystems eine allgemeine Definition gegeben werden:

> **Definition 8: Performance von Datenbanksystemen**
> *Die Performance eines Datenbanksystems gibt die Verarbeitungsleistung hinsichtlich einer Arbeitslast wider. Die Arbeitslast definiert dabei den spezifischen Gegenstand der Untersuchung sowie die Bemessungsgröße (Metrik). Im allgemeinen kann die Performance nach Gleichung 3.1 entweder anhand der Antwortzeit oder als Durchsatz angegeben werden (einfache beziehungsweise mehrfache, simultane Ausführung der Arbeitslast).*

Generell wird die Performance über die Ausführung eines sogenannten *Benchmarks* gemessen und angegeben. Nach *Vossen* handelt es bei einem Benchmark [Vos94, S.536]

> *um ein standardisiertes Programm, mit welchem das Leistungsverhalten eines Hard- oder Softwaresystems quantifiziert, gemessen und bewertet werden kann; darüberhinaus erlauben Benchmark-Ergebnisse einen Vergleich solcher Systeme.*

Benchmarks für Datenbanksysteme definieren dabei ein oder mehrere Anwendungsfälle, die direkt durch Datenbankoperationen ausgedrückt werden. Im allgemeinen dominieren Benchmarks, die reale Anwendungsfälle definieren, gegenüber synthetischen, die eher die oberen Performancegrenzen eines Datenbanksystems testen sollen. Dabei stellt sich die Frage, ob die gängigsten Benchmarks für Datenbanksysteme eher die Antwortzeit oder den Durchsatz als Performancemetrik nutzen. Zudem stellt sich zusätzlich die Frage, ob sich Gemeinsamkeiten und Charakteristiken ableiten lassen. Zur Beantwortung dieser Frage wurde im Rahmen dieser Dissertation eine umfangreiche Untersuchung durchgeführt. Die Untersuchung zeigte folgende Ergebnisse:

- So profan es klingen mag, aber alle untersuchten Benchmarks verfolgen das Ziel, die Performance der Datenbanksysteme, auf denen sie ausgeführt werden können, im Bezug auf einen Anwendungsfall vergleichen zu können. Man erhält also nach der Ausführung eines Benchmarks immer vergleichbare, diskrete und numerische Ergebnisse.

- Auffallend ist, dass die Mehrheit der untersuchten Benchmarks entweder die Antwortzeit oder den Durchsatz als Performancemetrik angeben. Eine Minderheit definiert eine Kombination aus beiden. In Tabelle 3.1 auf Seite 57 sind für die untersuchten Benchmarks dargestellt, welcher Anwendungsfall durch ein Datenbanksystem nachgestellt werden soll und welche Performancemetrik dabei die Leistungsfähigkeit hinsichtlich dieses Anwendungsfalles ausdrücken soll.

  Alle untersuchten Benchmarks für Datenbanksysteme zeigen auch, dass sie eine gewisse Zielstellung verfolgen. So existieren Benchmarks mit eindeutigen *OLAP*-Charakter. Das bedeutet, sie greifen zumeist lesend auf die Datenbank zu und versuchen, die Leistungsfähigkeit der analytischen Funktionen zu bemessen. Im Gegensatz dazu stehen die *OLTP*-zentrischen Benchmarks, die darauf abzielen, den maximalen Durchsatz an Datenbanktransaktionen zu ermitteln. Dabei wird eine Mischung aus lesenden und schreibenden Zugriff auf die Datenbank und ein Muster aus beiden verwendet. Das kann zum Beispiel bedeuten, dass auf eine Sequenz von a Transaktionen, die die Datenbankeinträge modifizieren, eine Sequenz aus b Transaktionen folgt, die die modifizierten Datenbankeinträge wieder ausliest. Es sind aber auch andere Kombinationen und Abfolgen aus a und b Transaktionen möglich.

- Alle untersuchten Benchmarks definieren die Größe der zu verarbeitenden Datenmenge als Konfigurationsparameter für den Benchmark. Dieser sogenannte *scale factor* ist ein Multiplikator, um große Datenbestände des jeweiligen Datenmodells zu generieren. Als Minimum wird im allgemeinen ein Multiplikator von 1 genutzt, was zumeist 1 GByte an Daten repräsentiert. Dabei ist es unerheblich, nach welchem Datenmodell diese Daten organisiert sind; die Priorität liegt in der Datengröße.

  Durch Variation der Datenmengengröße wird das Ziel verfolgt, das Skalierungsverhalten eines Datenbanksystems zu untersuchen. Dabei wird der *scale factor* schrittweise erhöht und der Benchmark für den geänderten Wert wiederholt. Skaliert man den *scale factor* linear,

so lassen die Benchmarkergebnisse Rückschlüsse auf das Skalierungsverhalten hinsichtlich des Einflussfaktors Datenmenge zu. Optimal aber illusorisch wäre bei diesem Beispiel, dass die Antwortzeit konstant verbleibt oder der Durchsatz um denselben Faktor skaliert.

Die Definition von *Vossen* für ein Benchmark ist nach den Untersuchungsergebnissen zutreffend, aber für den Bereich Datenbanksysteme zu allgemein. Daher muss der Begriff eines Benchmarks für Datenbanksysteme im Kontext dieser Dissertation enger gefasst werden.

Definition 9: Benchmark für Datenbanksysteme
Ein Benchmark für ein Datenbanksystem ist eine Computersoftware, die einen oder mehrere Anwendungsfälle nachstellt und die Leistung des Datenbanksystems bemessen und quantifizieren soll. Der oder die Anwendungsfälle definieren dabei zumeist eine Sequenz von Datenbankoperationen (Arbeitslast), die für eine Schnittstelle des Datenbanksystems geeignet sind. Die Datenbankoperationen greifen dabei lesend (OLAP) oder modifizierend (OLTP) auf die Datenbasis zu. Die Datenbasis kann einen realen Ursprung haben oder über einen Datengenerator synthetisch erzeugt werden. Ein Datengenerator hat den Vorteil, dass die Datenbasis skaliert werden kann, um das Skalierungsverhalten zu untersuchen.

Insgesamt besteht ein Benchmark für ein Datenbanksystem aus folgenden Komponenten: einer Beschreibung der Anwendungsfälle, der Sequenz von Datenbankoperationen und die dabei genutzten Metriken sowie gegegebenfalls ein Datengenerator.

Wünschenswert wäre es, wenn ein Benchmark existieren würde, der eine generelle Bestimmung der Performance von Datenbanksystemen zulässt. Das wäre insbesondere für die Endanwender von Datenbanksystemen vom Vorteil, da sie anhand der Benchmarkergebnisse einen Vergleich der Performance ziehen können [Vos94, S.536]. Damit wäre ein Entscheidungskriterium gegeben, anhand dessen sich die Endanwender für das performanteste Datenbanksystem entscheiden können (vgl. dazu [QZ13, BBN+13, LP09, XYB+13, DKL+13]). Eine solche *generelle Vergleichbarkeit* aller bekannten Datenbanksysteme ist aber schlechtweg unmöglich. Dafür gibt es mehrere Gründe: zum einen erfordert die Definition eines *generellen Benchmarks*, der alle denkbaren Anwendungsfälle für ein Datenbanksystem berücksichtigt und zum anderen setzt es eine einheitliche Schnittstelle voraus, die in allen Datenbanksystemen implementiert ist. In Kapitel 2 wurden die verschiedenen Daten- und Speichermodelle sowie die Architekturunterschiede und Zielsetzungen von Datenbanksystem beschrieben.

Es ist nicht realisierbar, einen Anwendungsfall zu definieren, der sich dazu eignet, alle genannten Aspekte allumfassend abzudecken und eine Performancemetrik dafür anzugeben. Erschwerend kommt hinzu, dass ein genereller Benchmark auch die technische Plattformen berücksichtigen muss, auf denen die diversen Datenbanksysteme ausgeführt werden.

Die Praxis zeigt hingegen, dass Benchmarks und insbesondere deren Ergebnisse etabliert und akzeptiert werden, sofern sie sich auf ein Datenmodell beschränken. Das gilt im besonderen Maße, wenn eine einheitliche Schnittstelle zur Verfügung steht. Beides erlaubt, einen Anwendungsfall sowie eine geeignete Metrik zum Ausdruck der Performance für das jeweilige Datenmodell zu definieren. Der Anwendungsfall kann dann in eine Arbeitslast in Form von Operationen überführt werden, die als essentiell für das jeweilige Datenmodell angesehen werden und die es zu vermessen gilt.

Benchmark	Beschreibung	Performance-Metrik
TPC-E	Definiert einen Satz von OLTP-Datenbankabfragen, wie sie typischerweise in der Datenbank einer Maklerfirma vorkommen	Durchsatz (Transaktionen pro Minute)
TPC-C	Definiert einen Satz von OLTP-Datenbankabfragen zur Bestimmung der oberen Grenze der Transaktionsbewältigung einer Datenbank	Durchsatz (Transaktionen pro Minute)
TPC-H	Definiert einen Satz von generellen OLAP-Datenbankabfragen zum Vergleich der Performance verschiedenster Datenbanken	Antwortzeit
TPC-DS	Wie TPC-H, allerdings wurden die Datenbankabfragen so modifiert, dass sie eine wesentlich höhere Last auf den Datenbankserver verursachen	Antwortzeit
TPC-VMS	Definiert eine zusätzliche Methodik, um die TPC-(E,C,H,DS)-Benchmarks in einer virtualisierten Umgebung durchführen zu können	Durchsatz oder Antwortzeit
BigData benchmark	Ein auf [PPR+09] basierender Benchmark zur Bestimmung der massiv-parallelen Fähigkeiten von Cloud-basierten Datenbanken	Antwortzeit
BigDataBench	Definiert einen kombinierten Satz von OLTP- und OLAP-Datenbankabfragen, um die generelle Performance von Datenbanken zu bestimmen	Durchsatz und Antwortzeit
HiBench	Definiert verschiedene reale Anwendungsszenarien für einen *Hadoop*-Cluster wobei auch OLAP-Datenbankabfragen an *Hive* durchgeführt werden [HHD+10]	Durchsatz und Antwortzeit
BigBench	Ein von TPC-DS inspirierter Benchmark [GRH+13]	Durchsatz (Anfragen pro Stunde)
LinkBench	Definiert einen Satz von Abfragen an eine Graphdatenbank [APBC13]	Antwortzeit
YCSB	Benchmark zur allgemeinen Bestimmung der Performance einer Datenbank, speziell für Cloud-basierte Datenbanken [PPR+11, CST+10]	Durchsatz (Operationen pro Sekunden)
XMark	Definiert einen Satz von Abfragen für Datenbanken mit der Fähigkeit, XML-basierten Datensätze verarbeiten zu können [SWK+02]	Antwortzeit
Cloudsuite	Benchmark zur allgemeinen Bestimmung der Performance eines Rechenzentrums [FFA+12]	(bisher nicht entschieden)
TPCx-HS	Definiert synthetische Anwendungsszenarien zur Bestimmung der Performance von *Hadoop*-Clustern [NPD+15]	Durchsatz oder Antwortzeit

Tabelle 3.1: Untersuchte Datenbanksystem-Benchmarks und ihre Performance-Metriken

Diese Vorgehensweise zeigt sich am Beispiel des *Transaction Processing Council* (*TPC*). Es ist ein Zusammenschluss von Datenbanksystem-Herstellern, deren Datenbanksysteme das relationale Datenmodell implementieren und *SQL* als einheitliche Datenmanipulations- und -abfrageschnittstelle nutzen. So wurden zunächst real existierende Anwendungsfälle analysiert, die auf relationalen Datenbanken abgebildet wurden. Danach wurde innerhalb des Konsortiums ein Konsens gefunden, um diese Anwendungsfälle in einzelne Arbeitslasten im Sinne einer Serie von Datenabfrage- und -manipulationsoperationen daraus zu formulieren. Um auch das Skalierungsverhalten eines Datenbanksystems zu untersuchen, wurde auch beschlossen, einen Datengenerator bereitzustellen, der eine synthetische, relationale Datenbasis erzeugen kann. Zudem wurde auch ein Generator bereitgestellt, der einerseits Datenbankoperationen generiert, die nur lesend und abfragend agieren (*OLAP*) und andererseits modifizierend auf die Datenbasis einwirken

(*OLTP*). Im letzteren Fall sei angemerkt, dass der Abfragengenerator dabei mit den Datengenerator zusammenarbeitet, damit die Anzahl der generierten Modifikationsoperationen auch mit der Größe der generierten Datenbasis skaliert. Schlußendlich wurden auch Metriken festgelegt, wie die Performance auszuwerten und bemessen ist. Im Falle der *TPC*-Benchmarks wird eine Metrik verwendet, die nicht zwischen *OLAP*- oder *OLTP*-Operationen unterscheidet. In der *TPC*-Terminologie wird nur von einer "Transaktion" gesprochen. Allerdings ist die Anzahl der Transaktionen und die Antwortzeit für die Berechnung der Metrik maßgeblich. Sie werden automatisch aus den Konfigurationsparametern des Daten- und Abfragegenerator ermittelt und durch weitere Benchmarkparameter ergänzt, zum Beispiel die Anzahl der Benutzer im Mehrbenutzerbetrieb[1]. Aufgrund dieser Angaben wird durch eine recht komplizierte Formel die Metrik als "kombinierte Anzahl an Transaktionen pro Stunde" errechnet und kann direkt zum Vergleich herangezogen werden. Zusätzlich ist es möglich, die Investitionskosten für die technische Plattform, auf der der Benchmark ausgeführt wurde, anzugeben und damit eine weitere Vergleichsmetrik zu errechnen ("kombinierte Anzahl an Transaktion pro Stunde und investiertem US-Dollar").

Ein solches Vorgehen ist für Datenbanksysteme, die ein anderes Datenmodell als das relationale implementieren, bisher nicht zu erkennen, obwohl Benchmarks für die in Textabschnitt 2.1 beschriebenen Datenmodelle bereits vorgeschlagen wurden (siehe Tabelle 3.1). Dadurch ist es für Endanwender von diesen Datenbanksystemen schwierig, die verschiedenen Implementationen zu vergleichen, da objektiv gewonnene, unabhängige Vergleichszahlen auf Basis von Benchmarks fehlen.

3.2 Energieverbrauch von Datenbanksystemen

Der Energieverbrauch eines Datenbanksystems ergibt sich im einfachsten Fall als das Produkt der mittleren elektrischen Leistung P_e über die Zeit t. Sie entspricht also der Definition der elektrischen Arbeit W_e:

$$W_e = P_e \cdot t \tag{3.2}$$

Zur Bestimmung des Energieverbrauches eines Datenbanksystems muss natürlich festgelegt werden, *was* beziehungsweise *welche technischen Komponenten* (für die Leistung P_e) über welche Zeit t vermessen werden müssen. Da es sich bei Datenbanksystemen in erster Linie um Computersoftware handelt, schränkt sich die Festlegung auf die technische Plattform ein, auf der das Datenbanksystem ausgeführt wird, zum Beispiel PCs, Server beziehungsweise Serververbünde.

Ähnlich wie bei der Bestimmung der Performance eines Datenbanksystems existieren für die Bestimmung des Energieverbrauches keinerlei generellen Konventionen. Es existieren zwar zwei Standards, die anerkannt sind, aber sie werden kaum genutzt. Die Gründe dafür sind Inkonsistenzen bei der Formalisierung und der Handhabung.

- Der erste anerkannte Standard ist *TPC-Energy* und wurde von den Mitgliedern des *Transaction Processing Counsil* (*TPC*, vgl. vorherigen Textabschnitt 3.1) ausgearbeitet. Das Ziel dieses Standards ist es, eine zusätzliche Metrik für ihre bereits bestehenden Benchmarks *TPC*-(*C,E,H,DS*) zu formalisieren. Dementsprechend ist dieser Standard mit diesen Benchmarks eng verzahnt und kann somit nicht für andere Benchmarks außerhalb der *TPC* genutzt werden.
  Im *TPC-Energy*-Standard werden Vorgehensweise, zu messende technische Komponente sowie die zu verwendenden Messgeräte definiert. Der Standard verlangt auch den Ein-

[1] Der Mehrbenutzerbetrieb wird dadurch nachgestellt, indem der Benchmark mehrfach simultan ausgeführt wird.

satz einer von der *TPC* entwickelten Computersoftware, die parallel zu dem eigentlichen *TPC*-Benchmark ausgeführt werden muss. Das Resultat ist eine Metrik ähnlich der im vorhergehenden Textabschnitt vorgestellten "kombinierte Anzahl an Transaktionen pro Stunde und investiertem US-Dollar", wobei anstatt der Investitionskosten die durchschnittliche elektrische Arbeit genutzt wird ("kombinierte Anzahl an Transaktionen pro Wattstunde"). Dadurch entsteht eine Metrik, die die erreichte Performance in Relation zur dabei verbrauchten elektrischen Arbeit setzt.

- Der zweite anerkannte Standard *SPECpower* stammt von der *Standard Performance Evaluation Coorperation* (*SPEC*) und ist darauf ausgerichtet, allgemein die Performance einer beliebigen Computersoftware in Relation zu dem Energieverbrauch zu setzen. Das bedeutet genauer, dass der *SPECpower*-Standard definiert, welche technischen Komponenten zu vermessen sind, während die Computersoftware ausgeführt wird. Damit eignet sich dieser Standard auch zur Evaluierung des Energieverbrauches eines Datenbanksystems, wenn zum Beispiel aus Vergleichsgründen ein geeigneter Benchmark als Arbeitslast ausgeführt wird. Als Fallstudien wird der *SPECpower*-Standard unter anderem für drei andere *SPEC*-Benchmarks genutzt (*SPEC ACCEL*, *SPEC OMP2012* und *SPECvirt_sc2012*) und soll in Zukunft als zusätzliche Metrik in allen Benchmarks genutzt werden, die von der *SPEC* stammen.
  Ähnlich wie bei der *TPC* ist auch eine Computersoftware Teil des Standards, die zusätzlich zu der zu evaluierenden Computersoftware ausgeführt werden muss. Durch den Umstand, dass der *SPECpower*-Standard zur universellen Bestimmung der Performance-Energieverbrauch-Relation dient, ist allerdings die Konfiguration recht komplex.

Beide Standards verdeutlichen, dass es das generelle Ziel ist, den Energieverbrauch eines Datenbanksystems immer in Relation zur Performance zu stellen. In Textabschnitt 3.1 ist dargelegt worden, dass zur Performancebestimmung eines Datenbanksystems in fast allen Fällen ein geeigneter Benchmark ausgeführt wird, um eine vergleichbare Arbeitslast zu erzeugen. Damit ergibt sich die Situation, dass es keinen generellen Vergleich des Energieverbrauches von Datenbanksystemen geben kann, da ein genereller Performance-Benchmark unmöglich ist.

Eine genauere Analyse der beiden oben genannten Standards zeigt zudem, dass Gleichung 3.2 als Basis zur Berechnung der jeweiligen Metriken dient. Folglich definieren beide Standards, welche technischen Komponenten (zur Ermittlung der mittleren elektrischen Leistung P_e) über welche Zeitdauer t vermessen werden müssen. Die Analyse ergab, dass beide Standards bezüglich dieser beiden Aspekte inkonsistent sind:

- Beide Standards definieren einen Satz an technischen Kernkomponenten, deren elektrischer Leistungsverbrauch separat gemessen werden muss. Darüber hinaus gehende technische Komponenten müssen je nach Standard nur optional vermessen werden. Das sind zum Beispiel Komponenten, die zur Formung eines Computernetzwerkes notwendig sind (Switche, Router und so weiter). Beide Standards vernachlässigen völlig, dass auch zur Kühlung elektrische Energie notwendig ist. Dieser Energieverbrauch wird bei der Berechnung der Metriken komplett ausgeklammert[2].

[2] Es sei angemerkt, dass mit der *Power Usage Effectiveness* (PUE) eine neue Metrik eingeführt wurde, die allerdings für ein gesamtes Datenverarbeitungszentrum gilt. Die PUE ist nach [YM13] als Quotient zwischen der gesamten zugeführten Energie und der Energie für Server definiert und ist im optimalsten Fall genau 1. Werte größer als 1 geben dabei den Mehraufwand an Energie an, zum Beispiel für die Kühlung der Server oder für die Beleuchtung der Räume. Allerdings ist die PUE als Metrik für Energieeffizienz in der Literatur umstritten. Der Studie in [YM13] zufolge haben zum Beispiel die sechs großen *Google*-Rechenzentren eine PUE von 1.21, gefolgt von *Microsoft* mit 1.22. Damit ist ersichtlich, dass mehr ein Fünftel der verbrauchten Energie nicht direkt von den Servern verbraucht wird.

Zudem verlangen beide Standards, dass die eingesetzten Messgeräte zumindestens geeicht sein müssen. Der *TPC-Energy*-Standard verlangt sogar den Einsatz spezieller, akkreditierter und zumeist teurer Messgeräten.

- *TPC-Energy* definiert den Zeitraum t, in der die Messung der elektrischen Leistung vollzogen wird, exakt mit genau dem Zeitraum, in der der jeweilige Benchmark ausgeführt wurde. *SPECpower* erweitert diesen Zeitrahmen und eine kurze Zeitperiode davor und danach (sogenannte *ramp-up*-Periode). Diese zusätzliche Zeitspanne ist aus dem Grund eingeführt worden, um auch die Vor- und Nachbereitung des Datenbanksystems auf den Benchmark einzubeziehen.

Beide Punkte haben dazu geführt, dass die eigentlich anerkannten Standards kaum zur objektiven Evaluierung des Energieverbrauches genutzt werden. Das liegt zum einen an der nicht trivialen Konfiguration und zum anderen am recht komplexen Aufbau der Messapparatur. Ein praktisches Beispiel kann diesen Sachverhalt näher verdeutlichen: es wird angenommen, dass die Performance und der Energievebrauch eines verteilten, relationen Datenbanksystems anhand des *TPC-H*-Benchmarks und *TPC-Energy*-Standards ermittelt werden soll. Letzgenannter Standard verlangt, dass der momentane Energieverbrauch eines jeden Knoten des Clusters, auf dem das verteilte Datenbanksystem ausgeführt wird, separat durch ein eigenes Messgerät gemessen werden muss. Besitzen die Knoten keinen lokalen Massenspeicher in Form von magnetischen Festplatten oder dergleichen, sondern nutzen entfernt liegenden Netzwerkmassenspeicher, so muss auch diese Komponente separat vermessen werden. Zudem muss die Computersoftware, die vom *TPC-Energy*-Standard vorgeschrieben wird, mit den einzelnen Messgeräten verbunden werden, damit gewährleistet ist, dass die Messungen nicht beeinflusst werden. Diese Computersoftware muss auf einem externen PC oder Server ausgeführt werden. Diese Konfiguration ist wie bereits erwähnt nicht trivial und erfordert spezielle Messgeräte mit externer Schnittstelle. Diese Computersoftware startet dann automatisch den Benchmark und beginnt mit den Messungen, die dann gestoppt werden, wenn der Benchmark durchgeführt wurde. Diese vorgeschriebene Vorgehensweise hat den Vorteil, dass objektive und automatisierte Messungen vorgenommen werden. Sie hat aber auch den Nachteil, dass sie mit nicht zu unterschätzenden Investitionen in Zeit (Aufbau, Konfiguration der Messapparatur) und Technik (spezielle Messgeräte) verbunden sind.

Infolgedessen sind in der Literatur kaum Vergleichszahlen publiziert worden, die auf Basis der beiden Standards *TPC-Energy* und *SPECpower* gewonnen worden sind. In der Praxis hat sich herausgestellt, dass es viel praktikabler ist, den Gesamtenergieverbrauch der Messapparatur mit nur einem Messgerät zu ermitteln. Auf das obige Beispiel bezogen werden alle Clusterknoten und die notwendige Netzwerktechnik an eine energieführende Schiene angeschlossen. An diese Schiene wird dann das Messgerät angeschlossen, sodass dann der gesamte elektrische Energieverbrauch gemessen werden kann. Dabei wird auch hier der Benchmark durchgeführt. Der Vorteil an dieser Vorgehensweise ist, dass der Versuchsaufbau vergleichsweise simpel ist und die Messwerte dennoch zur Berechnung der elektrischen Arbeit aus Gleichung 3.2 herangezogen werden können. Der Nachteil ist natürlich, dass die Messwerte nicht für einen direkten Vergleich genutzt werden können. Zudem verhindert diese simple Messapparatur auch die spätere, detailliertere Analyse der genutzten technischen Komponenten, da nur der Gesamtenergieverbrauch aller technischen Komponenten gemessen wurde.

Ein weiteres Problem ist die Vielzahl an Kombinationsmöglichkeiten technischer Komponenten. Dieses Problem äußert sich darin, dass sich ein Datenbanksystem durchaus hinsichtlich Performance und Energieverbrauch signifikant unterscheiden kann, wenn eine andere Kombination aus technischen Komponenten verwendet wird. Ein Beispiel ist, dass zunehmend magnetische

Festplatte durch *Solid State Disks* (SSDs) ersetzt werden. Der Grund ist, dass sie eine wesentlich höhere Datentransfer- und Zugriffsrate besitzen. Sie verhilft einem technischen System wie einem PC oder Server dazu, alle Arten von Computersoftware in deren Performance zu beschleunigen, sofern sie häufig Zugriff auf Massenspeicher nehmen. Zu dieser Art Computersoftware gehören auch das Betriebssystem und insbesondere Datenbanksysteme [SHH10, SLM16]. Als Folge des Austausches steigt nicht nur die Performance, sondern es sinkt gleichzeitig der Energieverbrauch. Dies ist besonders auf Systemen zu beobachten, dessen Hauptaufgabe es ist, Massenspeicher zu konzentrieren, zu organisieren und und durch geeignete Schnittstellen zur Verfügung zu stellen, etwa Dateiservern.

Im allgemeinen werden die technischen Komponenten eines Systems von dem Betriebssystem verwaltet, dass auf dem System ausgeführt wird. Dabei stellt das Betriebssystem geeignete, einheitliche Schnittstellen bereit, damit Anwendungen auf die technischen Komponenten zugreifen können. Somit stellt das Betriebssystem eine Zwischenschicht zwischen den technischen Komponenten und der Anwendung dar. Betriebssysteme abstrahieren den direkten Zugriff auf die technischen Komponenten in Form von Treibern, wobei diese selber gewisse Konfigurationsparameter bieten, die ebenfalls die Performance und unter Umständen auch den Energieverbrauch negativ beeinflussen können.

Wie bereits erwähnt, ist der Endanwender eines Datenbanksystems mit einer Vielzahl an Kombinationsmöglichkeiten der technischen Komponenten für ein System konfrontiert, auf dem das Datenbanksystem schlussendlich ausgeführt werden soll. Dazu kommt noch die Auswahl eines geeigneten Betriebssystem, das beide Aspekte vereinigen soll. Dieses Auswahlverfahren wird auch nicht erleichtert, da Betriebssysteme und Datenbanksysteme im allgemeinen darauf ausgelegt sind, auf einer Vielzahl dieser Kombinationsmöglichkeiten agieren zu können.

Dennoch haben sich vier technische Kernkomponenten herausgestellt, von denen man weiss, dass sie einen Einfluss auf die Performance und Energieverbrauch eines Datenbanksystems haben [THS10, VBSK09, XTW10]: Arbeitsspeicher, Massenspeicher, CPU und Hauptplatine (sogenanntes *Mainboard*). Deren Charakteristiken sind in Tabelle 3.2 auf Seite 62 dargestellt.

Diese technischen Kernkomponenten sind in allen bekannten technischen Systemen vorhanden, auf den ein Datenbanksystem nach Definition 2 von Seite 38 ausgeführt werden kann. Im Falle von verteilten Datenbanksystemen kommt eine technische Kernkomponente hinzu. Sie stellt die technische Grundlage für die Vernetzung bereit. Dazu zählen im allgemeinen Hubs und Switche, die als Kerntechnologie Kupferkabel und *Ethernet*[3] nutzen, sowie sogenannte *Fabrics*[4], die Glasfaserkabel zur Datenübertragung nutzen. Drahtlose Vernetzungen via *Wireless LAN* (WLAN, IEEE-Standard 802.11) sind zwar möglich, aber aufgrund des benutzen Mediums Luft und der begrenzten Datenübertragungsrate nicht sinnvoll.

[3] *Ethernet* ist im *IEEE*-Standard 802.3 definiert und beschreibt die Übertragung von digitalen Daten in kabelgebundenen Computernetzwerken. Dabei kommen Kupferkabel mit verschiedenen Abschirmungsgüten und unterschiedlichen Adaptern zum Einsatz.

[4] Ein *Fabric* ist eine Netzwerktopologie, wobei die Endpunkte (Server, Clients, Anwendungen) über mehrere Schnittstellen mit den Switchen und Routern verbunden sind. Damit wird sowohl die Ausfallsicherheit als auch die Datenübertragungsrate zwischen den Endpunkten erhöht. Zumeist kommen spezielle Switche und Router sowie spezialisierte Transportprotokolle und -verfahren zum Einsatz, zum Beispiel *FibreChannel*.

Kernkomponente	Beschreibung und verfügbare Technologien
Arbeitsspeicher	Ermöglicht die Speicherung von flüchtigen, temporären Daten sowie Anwendungsinstruktionen für die CPU. Sehr schnelle Zugriffszeiten, kann aber nur eine begrenzt große Datenmenge speichern. Technologisch wird zwischen statischen und dynamischen Arbeitsspeicher unterschieden, wobei ersterer vornehmlich als Zwischenspeicher, zum Beispiel bei CPUs, genutzt wird. Zudem wird auch nach dem Zugriffspfad unterschieden, dass heisst welche Datenmenge bei welcher Taktrate übertragen werden kann.
Massenspeicher	Ermöglicht die Speicherung von nicht-flüchtigen, permanenten Massendaten. Sehr schnelle (*Solid State Disks*, SSDs) bis schnelle Zugriffszeiten auf die gespeicherten Daten, wobei die Speicherkapazität die des Arbeitsspeichers um ein Mehrfaches übertrifft. Technologisch wird nach dem Trägermedium und dem physikalischen Zugriffsverfahren unterschieden, zum Beispiel optisch bei DVDs, magnetisch bei Bändern und Festplatten sowie per Feldeffekttransistor bei SSDs.
CPU (Prozessor)	Führt arithmetische und logische Instruktionen aus, um Daten zu verarbeiten und zu transformieren. Diese Daten stammen zumeist aus dem Hauptspeicher. Die Instruktionen können von einem oder mehreren unabhängigen Rechenwerken (Kerne) abgearbeitet werden, wobei sie auf einen kleinen, exklusiven und einem großen, gemeinsam genutzten Zwischenspeicher zugreifen können, um vielfach genutzte Daten zu puffern.
Hauptplatine (Mainboard)	Zentrale Platine, die Steckplätze für die anderen Kernkomponenten sowie Erweiterungskarten bietet. Die Kernkomponenten werden über Bussysteme miteinander verbunden und durch Stromschienen mit Energie versorgt.

Tabelle 3.2: Charakteristika der technischen Kernkomponenten

Ein weiterer Aspekt beim Energieverbrauch moderner Server ist, dass er nicht linear mit der Auslastung des Servers ansteigt. Die Gründe dafür sind diverse Energiesparmodi, die in fast allen Kernkomponenten aus Tabelle 3.2 zum Einsatz kommen. Die Auslastung eines Servers wird mit einer Prozentzahl angegeben, wobei einzelne ausgewählte Prozentangaben als *Lastpunkte* bezeichnet werden. In einer Studie von *Barroso et al.* wurde der Energieverbrauch der Server in einem *Google*-Rechenzentrum gemessen und in Relation zur Auslastung der Server gestellt [BHCC07]. Das Ergebnis ist grafisch in Abbildung 3.1 auf Seite 63 dargestellt. Auf der x-Achse ist der Energieverbrauch relativ zum gemessenen Spitzenverbrauch dargestellt. Auf der y-Achse ist die Auslastung (Lastpunkte) zu erkennen.

In Abbildung 3.1 ist ersichtlich, dass der Energieverbrauch mit steigender Auslastung ansteigt (die grüne Messwertlinie, im folgenden als Energieverbrauchskurve bezeichnet). Allerdings wird auch dann Energie verbraucht, wenn der Server keinerlei sinnvolle Aufgaben erledigt. Dabei ist hervorzuheben, dass selbst ohne Auslastung bereits die Hälfte des gemessenen Spitzenverbrauches an Energie verbraucht wird. Im optimalen Fall sollte die Energieverbrauchskurve in Abbildung 3.1 eine Diagonale vom Koordinatenursprung zum Punkt der beiden Achsenmaxima sein, also von (0,0) bis (100,100). Das bedeutet, dass der Energieverbrauch linear mit der Auslastung skaliert und bei keiner Auslastung auch keine Energie verbraucht wird. In Abbildung 3.1 liegt die Energieverbrauchskurve klar oberhalb dieser optimalen Energieverbrauchskurve.

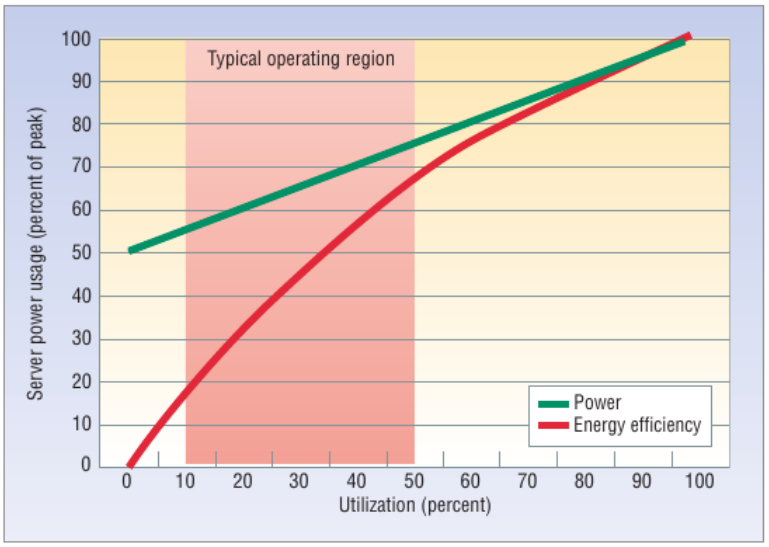


Abbildung 3.1: Energieverbrauch der Server in einem *Google*-Rechenzentrum für verschiedene Lastpunkte nach [BHCC07]

Wie bereits erwähnt, können Energiesparmechanismen dazu beitragen, dass die Energieverbrauchskurve in Teilen unterhalb der optimalen liegen kann. Das kann besonders häufig bei Lastpunkten unterhalb von circa 60 Prozent beobachtet werden. Im Falle von Datenbanksystemen und speziell für verteilte Datenbanksysteme sind die Auswirkungen der Energiesparmechanismen in mehreren Studien publiziert worden, unter anderem in [DaC09, LKP11, LP09]. Dabei sei besonders auf die Studie von *Wang et al.* in [WFXS11] hingewiesen. In ihr werden alle bis dato verfügbaren Energiesparmechanismen, deren Auswirkungen auf den Energieverbrauch sowie die Vor- und Nachteile detailliert beschrieben.

Insgesamt kann im Kontext dieser Dissertation der Energieverbrauch von Datenbanksystemen definiert werden. Sie stützt sich auf die getroffenen Aussagen dieses Textabschnittes.

Definition 10: Energieverbrauch von Datenbanksystemen
Der Energieverbrauch von Datenbanksystemen ist die geleistete elektrische Arbeit nach Gleichung 3.2, wobei das Datenbanksystem eine Arbeitslast ausführt, zum Beispiel einen Benchmark. Im besonderen Maße sind die technischen Kernkomponenten nach Tabelle 3.2 zur Ermittlung der geleisteten elektrischen Arbeit zu berücksichtigen; bei verteilten Datenbanksystemen kommt noch die technischen Komponenten zur Bildung des Computernetzwerkes hinzu.

3.3 Energieeffizienz von Datenbanksystemen und dessen Optimierungsmöglichkeiten

Die Energieeffizienz eines Datenbanksystem ist ein Maß für die Arbeitseffizienz und darf nicht mit dem *Wirkungsgrad* verwechselt werden. Der Wirkungsgrad ist als Quotient zwischen der zugeführten und der genutzten Leistung definiert. Dabei müssen beide Leistungsarten physikalische Größen nutzen. Da es aber keine Entsprechung für die Leistung eines Datenbanksystems gibt, kann also der Wirkungsgrad nicht herangezogen werden.

Daher wird die Energieeffizienz *EE* eines Datenbanksystems als Quotient zwischen der „Arbeit“ W_{DB} des Datenbanksystems und die dabei verbrauchte elektrischen Arbeit W_e definiert [HAM06, WFXS11]. Aus Gleichung 3.1 von Seite 54 für die “Leistung” und Gleichung 3.2 von Seite 58 für die geleistete elektrische Arbeit eines Datenbanksystems ergibt sich:

$$EE = \frac{W_{DB}}{W_e} = \frac{P_{DB} \cdot t}{P_e \cdot t} = \frac{Perf}{P_e} \tag{3.3}$$

Sowohl für die Arbeit des Datenbanksystems W_{DB} als auch für die elektrische Arbeit W_e gilt, dass sie als Leistung über die Zeit t definiert sind. Dabei wurde in Textabschnitt 3.1 gezeigt, dass die Leistung eines Datenbanksystems P_{DB} mit der mittleren Performance *Perf* gleichgesetzt werden kann, da es wie erwähnt keine physikalische Entsprechung gibt. Aus Textabschnitt 3.2 ist bekannt, dass die dabei verbrauchte beziehungsweise verrichtete elektrische Arbeit W_e das Produkt aus der mittleren elektrischen Leistung P_e über dieselbe Zeit t ist. Daher kann die Gleichung 3.3 für die Energieeffizienz *EE* um t gekürzt werden, womit sie der Quotient aus der mittleren Performance *Perf* und der mittleren elektrischen Leistung P_e ist.

Je nachdem, wie die mittlere Performance *Perf* definiert ist (siehe Gleichung 3.1 auf Seite 54), ergibt sich als Einheit für die Energieeffizienz *EE* $^n/_J$, also n pro *Joule*. Liegt der Fokus bei der Performance auf der Antwortzeit, zum Beispiel für die Ausführung eines Benchmarks oder einer einzelnen Datenbankabfrage, so kann $n = 1$ angenommen werden. Damit gibt die Energieeffizienz an, wieviel Joule im Durchschnitt gebraucht wurden, um dieses Ziel zu erreichen, also die komplette Ausführung des Benchmarks oder der Datenbankabfrage. Liegt der Fokus allerdings auf dem Durchsatz, so gibt n zum Beispiel an, wieviele Datensätze pro verbrauchtem Joule elektrischer Energie verarbeitet werden konnten. Es existierenden aber auch andere, eher technische Metriken für die Energieeffizienz. Zum Beispiel kann n auch die durchschnittliche Anzahl an gelesenen oder geschrieben Datenblöcke eines Massenspeichers sein. Die daraus resultierende Metrik wäre dann “durchschnittliche Anzahl an gelesenen beziehungsweise geschriebenen Datenblöcken pro investiertem Joule”. Diese Metriken können noch weitergehend mit anderen Metriken in Relation gebracht werden. So definieren zum Beispiel die *TPC*-Benchmarks *TPC*-(C,E,H) die Investitionskosten für die technische Plattform als zusätzliche Metrik. Es wären allerdings auch andere Metriken denkbar, zum Beispiel die Anforderungen an Serverstellplätzen in einem Rechenzentrum. Insgesamt entstehen damit zusätzliche Vergleichsmöglichkeiten im Bezug zur Energieeffizienz.

Evaluation der Energieeffizienz Aus Gleichung 3.3 ist auch ersichtlich, wie die Energieeffizienz eines Datenbanksystems evaluiert wird. Im allgemeinen wird das Datenbanksystem auf einer technischen Plattform betrieben. Das kann ein einzelner PC, Server oder ein Cluster sein. Das Datenbanksystem und seine technische Plattform wird dergestalt konfiguriert, dass sie eine definierte Arbeitslast ausführen soll, wobei zumeist aus Vergleichsgründen ein standardisierter,

anerkannter Benchmark genutzt wird. Durch die Ausführung des Benchmarks wird dann die Performance ermittelt. Während der Ausführung wird der Energieverbrauch der technischen Plattform gemessen. Beide Metriken, also Performance und Energieverbrauch, können dann genutzt werden, um anhand von Gleichung 3.3 die Energieeffizienz zu berechnen.

Vergleich von Energieeffizienzwerten Generalisiert man dieses Vorgehen, so sind drei zentrale Komponenten zu erkennen, die zur Berechnung der Energieeffizienz essentiell sind: das Datenbanksystem, eine definierte Arbeitslast in Form eines Benchmarks und die technische Plattform. Diese Komponenten sind aus dem Grund essentiell, da sie einen Vergleich den Energieeffizienzwerten ermöglichen. Dafür müssen aber mindestens zwei von diesen Komponenten konstant verbleiben und die verbleibende kann variieren. Zur besseren Erklärung können folgende Beispiele betrachtet werden:

- Man denke sich zwei Hersteller, die jeweils ein relationales Datenbanksystem implementiert haben. Zusätzlich wird angenommen, dass beide Datenbanksysteme auf ein und derselben technischen Plattform betrieben werden können. Ferner sei angenommen, dass ein Benchmark existiert, der für beide Datenbanksysteme auf der technischen Plattform ausführbar ist. Wird nun der Benchmark für beide Datenbanksysteme auf der technischen Plattform ausgeführt, so kann das oben beschriebene Verfahren genutzt werden, um für beide Datenbanksysteme die Energieeffizienz zu errechen. Bezogen auf die drei zentralen Komponenten hieße es, dass die technische Plattform und die Arbeitslast konstant ist und das Datenbanksystem variiert. Als Resultat kann die Energieeffizienzwerte der beiden angenommenen Datenbanksysteme direkt verglichen werden.
  Ein solcher direkter Vergleich kann zum Beispiel genutzt werden, um die Entscheidungsfindung hinsichtlich eines Datenbanksystems zu fördern. Dabei können Endanwender aufgrund der vergleichbaren Energieeffizienzwerte das energieeffizientestem Datenbanksystem auswählen.
- Es kann sich ein Endanwender vorgestellt werden, der in Zukunft ein Datenbanksystem für einen Kunden betreiben soll. Vertraglich ist geregelt, dass das Datenbanksystem innerhalb enger Grenzen agieren soll. Zum Beispiel dürfen die Antwortzeiten bestimmter Datenbankabfragen eine gewisse zeitliche Grenze nicht überschreiten. Der Endanwender möchte nun eine technische Plattform auswählen, die die vom Kunden definierte Arbeitslast sowohl performant als auch energiesparend, also am energieeffizientesten, ausführen kann. Dafür führt der Anwender die Arbeitslast für verschiedene technische Plattformen nach dem oben beschriebenen Vorgehen aus und errechnet die Energieeffizienzwerte. Sie sind dann direkt miteinander vergleichbar und dienen dann der finalen Entscheidungsfindung für eine technische Plattform.
  Für die obigen drei zentralen Komponenten bedeutet dies, dass das Datenbanksystem und die Arbeitslast als konstant angesehen werden können und die technische Plattform variiert wird.
- Ein Endanwender betreibt ein Datenbanksystem auf einer technischen Plattform und hat die Bestrebung, das Skalierungsverhalten des Datenbanksystems untersuchen, um etwa technische Grenzen wie dem Energieverbrauch oder Performanceengpässe auszuloten. Zu diesem Zweck nutzt er einen geeigneten Benchmark, der es zum Beispiel ermöglicht, die zu testende Datenmenge zu konfigurieren. Nun führt er denselben Benchmark mit unterschiedlichen Konfigurationen, zum Beispiel einer linear ansteigenden Datenmenge, nach dem oben beschriebenen Verfahren aus und erhält damit verschiedene Energieeffizienzwerte. Sie sind direkt miteinander vergleichbar und können für weitergehende Analysen

hinsichtlich des Skalierungsverhalten herangezogen werden, zum Beispiel der Relation zwischen Datenmenge und Energieeffizienz.
Bezogen auf die obigen drei zentralen Komponenten hieße es, dass das Datenbanksystem und die technische Plattform konstant sind, während die Arbeitslasten (die verschiedenen Konfigurationen des Benchmarks) variieren[5].

Diese drei Beispiele zeigen, dass Energieeffizienzwerte nicht direkt miteinander vergleichbar sind. Dabei ist zu beachten, dass die Messergebnisse beziehungsweise die Metriken bei der Ausführung eines Benchmarks für ein Datenbanksystem immer davon abhängig sind, auf welcher technischen Plattform das Datenbanksystem ausgeführt wird. Praktisch alle anerkannten, standardisierten Benchmarks für Datenbanksysteme erfordern die Angabe der technischen Plattform beziehungsweise der Messapparatur zur Veröffentlichung der Messergebnisse.

Alle bisher beschriebenen Aspekte können dafür verwendet werden, eine Definition der Energieeffizienz von Datenbanksystemen im Kontext dieser Dissertation zu bilden.

Definition 11: Energieeffizienz von Datenbanksystemen
Die Energieeffizienz eines Datenbanksystems ist ein Maß für die Arbeitseffizienz. Sie gibt die Relation zwischen der Performance eines Datenbanksystems hinsichtlich der Ausführung einer Arbeitslast auf einer technischen Plattform und der dafür notwendigen elektrischen Energie an.

Wie bereits erwähnt, sind zur Bestimmung der Energieeffizienz von Datenbanksystemen drei Komponenten erforderlich: das Datenbanksystem, die Arbeitslast (zumeist in Form eines Benchmarks aus Vergleichsgründen) und der technischen Plattform. Nur wenn zwei dieser drei Komponenten fixiert werden und die verbliebene dritte varriert wird, ist der direkte Vergleich der gewonnenen Energieeffizienzwerte möglich.

Unter Auslassung von Definition 11 muss auch ein anderer Aspekt beim Vergleich von Energieeffizienzwerten betrachtet werden. Betrachtet man Gleichung 3.3 auf Seite 64 zur Berechnung der Energieeffizienz, so erkennt man, dass sie die Performance beziehungsweise den Energieverbrauch relativieren kann. Zur Illustration sei folgendes Beispiel angenommen: es existieren zwei Datenbanksysteme, bei denen es möglich ist, ein und denselben Benchmark durchzuführen. Als Performancemetrik sei die Antwortzeit t angenommen, die es braucht, um den Benchmark komplett auszuführen. Für das erste Datenbanksystem ergibt sich eine Antwortzeit t_1. Für das zweite angenommene Datenbanksystem zeigt sich, dass es doppelt so lange gebraucht hat, den Benchmark komplett auszuführen, da eine andere technische Plattform genutzt wurde. Das bedeutet, dass $t_2 = 2 \cdot t_1$. Es hat sich aber auch gezeigt, dass die technische Plattform, auf der das erste Datenbanksystem ausgeführt wurde, durchschnittlich doppelt soviel elektrische Leistung verbraucht hat, also $P_1 = 2 \cdot P_2$, um die Antwortzeit t_1 zu erreichen. Wendet man Gleichung 3.3 an, um die Energieeffizienz beider Datenbanksysteme hinsichtlich des eingesetzten Benchmarks zu errechnen, so ist erkenntlich, dass beide gleich sind. Ein Vergleich der Energieeffizienzwerte bringt daher keinen zusätzlichen Wissensgewinn, außer wie energieeffizient die Datenbanksysteme den Benchmark *auf den jeweils eingesetzten technischen Plattformen* ausgeführt haben. Einen gewissenen Aussagewert haben beide Energieeffizienzwerte nur dann, wenn man die beiden Aspekte Performance (die Antwortzeit t) oder den Energieverbrauch der technischen Plattform unterschiedlich priorisiert beziehungsweise gewichtet:

[5] Es sei auch bemerkt, dass dieses Beispiel auch für die Variation der technischen Plattform dienen kann. Dabei verbleibt das Datenbanksystem und die Arbeitslast konstant und die technische Plattform variiert. Insgesamt kann so das Skalierungsverhalten des Datenbanksystems hinsichtlich der technischen Plattform evaluiert werden.

- Eine Höherpriorisierung der Performance als Bestandteil der Energieeffizienz kann zum Beispiel vorgenommen werden, wenn ein Datenbanksystem noch weitere Arbeitslasten sequentiell ausführen soll (sogenannte *Batch*-Verarbeitung). Für das obige Beispiel sieht man, dass das erste Datenbanksystem in der Antwortzeit $t_2 = 2 \cdot t_1$ den Benchmark zwei Mal ausführen kann, also zwei Arbeitslasten. Das zweite Datenbanksystem kann in dieser Zeit nur eine Arbeitslast ausführen. Dieses Vorgehen hat aber den Nachteil, dass es zu Kosten des Energieverbrauches geht.
- Eine Höherpriorisierung des Energieverbrauches als Bestandteil der Energieeffizienz kann zum Beispiel dann vorkommen, wenn die Antwortzeit t nicht maßgeblich ist. Ein möglicher Grund kann es sein, dass es keinerlei Abhängigkeiten zum Ergebnis der Arbeitslast gibt. So kommt es in der Praxis häufig vor, dass als Arbeitslast für ein Datenbanksystem die Erstellung von tagesaktuellen Statistiken und Reports definiert wird. Solche Arbeitslasten werden dann meistens über Nacht ausgeführt, da hier die Antwortzeiten des Datenbanksystems weniger wichtig sind. Hinsichtlich des Energieverbrauches kann dann aber das Datenbanksystem auf einer technischen Plattform betrieben werden, die zur Bewältigung der Arbeitslast weniger elektrische Energie benötigt.

Optimierung der Energieeffizienz Wie bereits in Kapitel 1 beschrieben, kann aufgrund des steigenden, weltweiten Informationsaufkommens davon ausgegangen werden, dass bestehende Datenverarbeitungszentren weiter ausgebaut beziehungsweise neue entstehen werden, um die Informationen auch zukünftig zu speichern, zu verarbeiten und abrufbar zu gestalten. Dadurch ist zu erwarten, dass auch der weltweite Energieverbrauch dieser Datenverarbeitungszentren steigen wird. Dazu zählt nicht nur der Betrieb der Server in diesen Zentren, sondern auch andere Aufwendungen, etwa die Kühlung der Server. Der steigende Energieverbrauch führt dazu, dass dieser mittlerweile ein nicht zu unterschätzender finanzieller Kostenfaktor geworden ist. Eine Optimierung der Energieeffizienz kann es ermöglichen, diese Kosten zu reduzieren.

Gleichung 3.3 auf Seite 64 für die Energieeffizienz von Datenbanksystemen impliziert zwei Möglichkeiten, sie zu optimieren:

1. Die Performance eines Datenbanksystems wird gesteigert, ohne den Energieverbrauch wesentlich zu erhöhen. Dieses Vorgehen ist in der Praxis recht häufig anzutreffen und verdankt ihre Popularität dem Umstand, dass Datenbanksysteme im allgemeinen als Computersoftware implementiert sind und Quelltextänderungen recht einfach möglich sind. Das eröffnet natürlich die Möglichkeit, neue Algorithmen und Techniken zu implementieren oder den bestehenden Quelltext zu optimieren, zum Beispiel um bestehende technische Komponenten besser zu nutzen.
   Ein Beispiel für die Modifikation eines bestehenden Datenbanksystems zur Erhöhung Energieeffizienz ist die Änderung der Komponente zur Erstellung des Ausführungsplanes (*query planner*). Nach [XTW10] wurde das relationale Datenbanksystem *Postgresql* so modifiziert, dass nach der Erstellung aller möglichen Ausführungspläne für eine Datenbankabfrage nicht der Ausführungsplan ausgewählt wird, der am meisten Performance verspricht, sondern der am energieeffizientesten ist. Insgesamt bedeutet diese Modifikation, dass neben der Performance auch der erwartete Energieverbrauch für die Ausführung einer Datenbankabfrage in Betracht gezogen wird.
   Ein anderes Beispiel ist die komplette Neuimplementation eines Datenbanksystems, das direkt auf Energieeffizienz ausgerichtet ist. Dabei sind die beiden relationalen Datenbanksysteme *F1* und *WattDB* zu nennen. Erstgenanntes wurde von *Google* für seine weltweit verteilten Datenverarbeitungszentren entworfen und implementiert, wobei die Energieeff-

fizienz ein erklärtes Architekturziel ist [SH11, SVSR13]. Das zweitgenannte Datenbanksystem entstammt einem Forschungsprojekt der *Technischen Universität Kaiserlautern*. Dieses Datenbanksystem nimmt für sich in Anspruch, nicht nur energieeffizient, sondern sogar energie-proportional zu sein [SHH10, SH13]. Diese Proportionalität kann also Analogie zur Auslastung von Servern angesehen werden, wie sie in Textabschnitt 1.2 beschrieben ist. Betrachtet man dazu Abbildung 3.1 auf Seite 63, so kann sich auf der x-Achse die Auslastung des Datenbanksystems und auf der y-Achse dessen Energieverbrauch in Relation zum Spitzenenergieverbrauch gedacht werden. Es ergibt sich eine Energieproportionalität, wenn die Energieverbrauchskurve des Datenbanksystems eine Diagonale vom Koordinatenursprung (0,0) zu den beiden Achsenmaxima (100,100) ergibt. Dass bedeutet zweierlei: wenn das Datenbanksystem aktuell keine Arbeitslasten ausführt, dann wird *keinerlei* elektrische Energie verbraucht. Steigert man die Auslastung des Datenbanksystems um einen gewissen Prozentsatz, so steigt der Energieverbrauch auch nur um diesen Prozentsatz an.

2. Der Energieverbrauch wird gesenkt, ohne dass die Einbußen in der Performance gravierend sind. Verglichen mit der Optimierung der Performance sind die Möglichkeiten hinsichtlich der Reduktion des Energieverbrauches eingeschränkt, da sie zumeist technische Komponenten betreffen.
   Eine Möglichkeit ist es, einfach technische Komponenten auszutauschen, die zum Beispiel eine höheren Datendurchsatz oder eine höhere Verarbeitungsgeschwindigkeit bieten. Hier sei das Beispiel aus Textabschnitt 3.2 genannt, in der die Energieeffizienz eines Datenbanksystems dadurch optimiert wurde, indem traditionelle Massenspeichers in Form von magnetischen Festplatten durch *Solid State Disks* (SSDs) ersetzt wurden. Eine andere Möglichkeit ist es, spezialisierte technische Komponenten zu nutzen, zum Beispiel moderne Grafikkarten. Dazu werden existierende Datenbanksysteme so modifiziert, dass sie die Programmierschnittstellen der Grafikkarten nutzen, um Datenbankoperationen wie das Sortieren von Datensätzen extrem zu beschleunigen [GGKM06, BS10, SEM+14]. Dadurch wird die CPU entlastet und Energie gespart.
   Eine zusätzliche Möglichkeit ist es, die Auslastung und damit die Effizienz der technischen Komponenten zu erhöhen, auf der ein Datenbanksystem betrieben wird. So konnte zum Beispiel in Rahmen einer Studie von *Lang und Patell* gezeigt werden, dass die Energieeffizienz eines verteilten Datenbanksystems dadurch gesteigert werden kann, indem die Verarbeitungsgeschwindigkeit der beteiligten Server künstlich begrenzt wurde [LP09]. Dafür wurde die Taktrate der CPUs auf ein Maß reduziert, in der die Antwortzeiten der gestellten Arbeitslast gerade noch akzeptabel war, aber Energie gespart werden konnte.

Es sei an dieser Stelle auf die vergleichende Studie von *Wang et al.* in [WFXS11] verwiesen, in der bis dato alle bisher publizierten Technologien zur Optimierung der Energieeffizienz von Datenbanksystemen aufgelistet sind, detailliert beschrieben und verglichen werden. Zudem sei auch angemerkt, dass natürlich auch die Optimierung der Energieeffizienz beide Richtungen möglich ist, also die Erhöhung der Performance bei gleichzeitiger Senkung des Energieverbrauches.

In der genannten vergleichenden Studie von *Wang et al.* in [WFXS11] bemerkten die Autoren bei ihren Recherchen, dass es im Bezug auf die Optimierung der Energieeffizienz von Datenbanksystemen einen Disput gibt. Im Originalen heisst es:

> *It is interesting to note two different opinions in recent research work:*
>
> - *[...] claimed that energy efficiency and performance are two different optimization goals, and there exists the tradeoff of energy efficiency and performance.*

- *[...] claimed that energy efficiency and performance are consistent, and it said that "within a single node system (intended for use in scale-out architectures), the most energy-efficient configuration is typically the highest performing one."*

Dabei gilt es zu beachten, das die Studie von *Wang et al.* dabei Publikationen analysiert haben, die das Skalierungsverhalten hinsichtlich der Energieeffizienz von verteilten Datenbanksystemen zum Gegenstand der Untersuchung hatten. Wenn man unter diesem Gesichtspunkt das obige Zitat ins Deutsche übersetzt, so ergeben sich zwei unterschiedliche Standpunkte, die Energieeffizienz von verteilten Datenbanksystemen zu optimieren:

1. Die Performance und der Energieverbrauch eines verteilten Datenbanksystems sind voneinander unterschiedliche Optimierungsziele. Das bedeutet, dass versucht wird, die Performance zu maximieren und den Energieverbrauch zu minimieren. Die energieeffizienteste Konfiguration des verteilten Datenbanksystems (zum Beispiel die Auswahl der technischen Komponenten oder die Clustergröße) zur Ausführung einer gegebenen Arbeitslast ist ein Kompromiss zwischen der performantesten und energiesparendsten Konfiguration. Dieser Ansatz ist bereits im obigen Textabschnitt "Vergleich von Energieeffizienzwerten" bereits beschrieben.

2. Die energieeffizienteste Konfiguration eines verteilten Datenbanksystems, das in einem Cluster mit mehreren Clusterknoten betrieben wird, ergibt sich daraus, dass man die energieeffizienteste Konfiguration für einen Minimal-Cluster ermittelt, also einem Cluster mit nur einem Clusterknoten. Dabei hat sich gezeigt, dass die performanteste Konfiguration hinsichtlich der Arbeitslast des Minimal-Clusters gleichzeitig die energieeffizienteste war. Man geht davon aus, dass sie dieses Verhalten auch für Cluster mit mehr als einem Clusterknoten zeigt.

Eine Antwort auf die Frage, welche der beiden Ansätze nun der erfolgversprechendste ist, ist bis dato nicht geklärt. Ein Grund ist, dass es kaum seriöse, unabhängige Publikationen gibt, die experimentelle gewonnene Ergebnisse im Bezug auf die Energieeffizienz von Datenbanksystemen beinhalten. Das mag daran liegen, dass es einen bedeutenden Aufwand in Zeit und unter Umständen Investitionen in technische Komponenten bedeutet, experimentelle Ergebnisse hinsichtlich der Energieeffizienz nach wissenschaftlichen Ansprüchen zu gewinnen. Auch die experimentellen Ergebnisse zur Energieeffizienz von Datenbanksystemen in Kapitel 4, die im Rahmen dieser Dissertation durchgeführt wurden, erlauben keine abschließende Antwort auf obige Frage, obwohl sie beide Ansätze untersuchen.

Es ist aber die Ansicht des Autors dieser Dissertation, dass der erste Standpunkt der plausiblere von den beiden beschriebenen ist. Der zweite Standpunkt, die energieeffizienteste Konfiguration eines Datenbanksystems aus der Evaluation eines Minimalclusters zu gewinnen und anzunehmen, dass diese Konfiguration auch für Cluster jenseits des Minimalclusters die energieeffizienteste ist, gilt nach Ansicht des Autors nur für verteilte Datenbanken, die eine zentrale Komponente einsetzen. Das gilt insbesondere für traditionelle, zentral betriebene Datenbanksysteme, die das relationale Datenbankschema und das *ACID*-Paradigma implementieren sowie eine *SQL*-Schnittstelle bieten. Hier kann es sinnvoll sein, einen Minimalcluster zu bilden und hinsichtlich der Energieeffizienz zu evaluieren, um die gewonnenen Ergebnisse zu nutzen, einen Cluster mit zentraler Komponente zu bilden, der durchaus ähnlich wenn nicht gleich energieeffizient ist.

Die Praxis zeigt, dass diese Form von verteilten Datenbanksystemen kaum mehr Bedeutung hat. Mittlerweile werden verteilte Datenbanksysteme eingesetzt, die keine zentrale Komponente zur Lastverteilung und zur Steigerung der Ausfallsicherheit mehr brauchen. Zusätzlich sind auch

verteilte Datenbanksysteme zu betrachten, die nicht mehr dem *ACID*-Paradigma folgen und keine standardisierte Schnittstelle für den Zugriff auf die Datenbanken mehr bieten, etwa verteilte *NoSQL*-Datenbanksysteme. Zudem zeigen die Ergebnisse der Experimente, die im Rahmen dieser Dissertation durchgeführt wurden und im nachfolgenden Textabschnitt erläutert werden, dass es bei verteilten Datenbanksystemen in einem Cluster Wechselwirkungen auftreten, die einen zum Teil immensen Einfluss auf die Performance und den Energieverbrauch haben können. Daher ist es die Überzeugung des Autors, dass der erste Standpunkt nachvollziehbarer ist, die Wechselwirkungen im Bezug auf die Energieeffizienz zu evaluieren. Diese Wechselwirkungen und deren Effekte können mit der Methode, die der zweite Standpunkt vertritt, nicht ergründet und demnach hinsichtlich der Evaluation und Optimierung der Energieeffizienz von verteilten Datenbanksystemen genutzt werden.

3.4 Kapitelzusammenfassung

Nachdem im vorherigen Kapitel 2 die Grundlagen der Architektur von Datenbanksystemen beschrieben wurde, verdeutlicht Kapitel 3, inwiefern Datenbanksysteme verglichen werden können.

Man kann konstatieren, dass Datenbanksysteme bereits seit ihrer kommerziellen Einführung hinsichtlich ihrer Leistungsfähigkeit verglichen wurden. Dabei wurden allerdings immer Metriken genutzt, die darauf ausgerichtet waren, die Leistungsfähigkeit eines Datenbanksystems nur nach der Antwortzeit oder dem Durchsatz zu bemessen. Allerdings hat sich gezeigt, dass diese beiden Vergleichsmetriken zwar immernoch primär wichtig sind, aber andere Vergleichsmetriken hinzugekommen sind und wachsende Bedeutung erfahren.

Der Hauptgrund dafür ist, dass *BigData*-Anwendungsszenarien zumeist damit einhergehen, dass die Datenbasis im TByte-Bereich oder sogar höher ist, die es zu speichern, zu verwalten und abzurufen gilt. Tradionelle Datenbanksysteme sind zunehmend aufgrund ihrer Zielstellung und ihrer Architektur nicht mehr in der Lage, große und sehr große Datenmenge effizient zu verwalten und zu verarbeiten. Das implementierte relationale Datenmodell und das *ACID*-Paradigma unterstützt zwar eine Vielzahl von Anwendungsfällen, zeigt in der Praxis aber ein schlechteres Skalierungsverhalten im Bezug auf die Performance, wenn die Datenbasis deutlich anwächst. Das führte zur Implementierung und Einführung von Datenbanksystemen, in denen nicht mehr das relationale Datenmodell genutzt wird, sondern solche, die für für spezielle Anwendungsfälle eine höhere Performance in Form der Antwortzeit oder dem Durchsatz bieten.

Insgesamt ist damit eine Situation entstanden, dass mittlerweile über 150 Datenbanksysteme bekannt sind, die alle ihre Vor- und Nachteile hinsichtlich ihrer Nutzung haben sowie auf einer Vielzahl von technischen Plattformen ausführbar sind. Endanwender sehen sich damit konfrontiert, aus einer fast unüberschaubaren Kombinationsvielfalt aus Datenbanksystem, technischen Komponenten sowie Nutzungsszenarien genau dasjenige zu ermitteln, dass für ihren Anwendungsfall am geeignetesten ist.

Der Term “am geeignetesten” ist in diesem Zusammenhang offensichtlich mehrdeutig. Möchte man die Eignung eines Datenbanksystems zur Unterstützung eines gegebenen Anwendungsfalles objektiv ermitteln, sind Vergleichmetriken unumgänglich. Wie der Name bereits suggeriert, müssen solche Metriken vergleichbar sein, wobei nach objektiven Gesichtspunkten agiert werden muss. Das bedeutet, dass die Metrikwerte entweder durch Recherche oder experimentell bestimmbar sein müssen. Mit anderen Worten heisst das, dass die Metrikwerte einem objektiven Vergleich standhalten müssen. Beispiele für solche Metriken, die per Recherche ermittelt werden können, sind etwa der Preis oder die Anzahl der unterstützten Betriebssysteme eines Datenbank-

systems. Experimentell bestimmbare Vergleichmetriken haben hingegen den Vorteil, dass durch die Variation des gegebenen Anwendungsfalles auch das Skalierungsverhalten von Datenbanksystemen bemessen werden kann. Das bedeutet, dass zum Beispiel die Größe der Datenbasis schrittweise erhöht wird und das Verhalten des Datenbanksystems hinsichtlich der Antwortzeiten oder dem Durchsatz beobachtet wird.

Allerdings existieren andere Vergleichsmetriken von Datenbanksystemen, die im Rahmen dieser Dissertation eine wichtigere Bedeutung haben. Dazu gehören neben der Performance der Energieverbrauch sowie die Energieeffizienz von Datenbanksystemen.

So wurde in Textabschnitt 3.1 zunächst der Begriff der *Performance* von Datenbanksystemen konkretisiert und Möglichkeiten evaluiert, sie zu bemessen. Dabei zeigte sich, dass im allgemeinen immer ein standardisierter und anerkannter Benchmark verwendet wird. Dabei wird ein Anwendungsfall und die dafür notwendige Datenbasis definiert. Zusätzlich werden auch Metriken zur Bewertung und Bemessung der Performance sowie Handlungsanweisungen zur Durchführung des Benchmarks angegeben. Der Vorteil der Nutzung von Benchmarks ist es, dass damit eine objektive Vergleichsbasis zur Bemessung der Performance von Datenbanken geschaffen wird. Wie in Textabschnitt 3.1 beschrieben, zeigt sich allerdings, dass nicht jeder Benchmark für jedes Datenbanksystem eingesetzt werden kann. Der Grund ist, dass Benchmarks darauf ausgerichtet sind, ein definiertes Daten- und Speichermodell zu evaluieren, die durch den konstruierten Anwendungsfall bestimmt werden. Somit ergibt sich der Umstand, das es keinen generellen Benchmark geben kann, der in der Lage ist, *alle bekannten* Datenbanksysteme *einheitlich* zu evaluieren.

Allerdings hat sich auch gezeigt, dass der Schwerpunkt nicht allein auf der Evaluierung der Performance liegt. So ist auch der Energieverbrauch von Datenbanksystemen in den Blickpunkt geraten, da der Energieverbrauch ein nicht zu unterschätzender Kostenfaktor beim Betrieb von Datenbanksystemen geworden ist. Das hat dazu geführt, dass Performance nicht allein die Bemessungsgrundlage für die Güte beziehungsweise die Eignung eines Datenbanksystems ist, sondern auch der Aufwand an Energie, um diese Performance zu erreichen. Diese sogenannte Energieeffizienz kann dadurch bestimmt werden, indem zusätzlich zur Ausführung eines Benchmarks auch der Energieverbrauch während der Ausführung gemessen wird, wodurch sie dann berechnet werden kann. Im Gegensatz zur Bemessung der Performance wird in Textabschnitt 3.2 erläutert, wie der Energieverbrauch von Datenbanksystemen evaluiert werden kann. Hier ist vom Vorteil, dass der Energieverbrauch eine physikalische Größe ist und damit bereits bewährte und anerkannte Verfahren und Metriken existieren. Allerdings muss im Umgang mit Datenbanksystemen geklärt werden, welche Faktoren zum Energieverbrauch beitragen. In Textabschnitt 3.2 wurden diese Faktoren evaluiert. Als Ergebnis wurden vier technische Kernkomponenten identifiziert und erläutert.

In Textabschnitt 3.3 wurde schließlich der Begriff der *Energieeffizienz* von Datenbanksystemen definiert. Die Energieeffizienz ist zusätzlich zum Energieverbrauch eine der aktuellsten Vergleichsmetriken im Bezug auf Datenbanksysteme. Daher wurde in Textabschnitt 3.3 evaluiert, wie die Energieeffizienz bemessen und Energieeffizienzwerte bewertet werden können. Hier stellt sich natürlich die Frage, welchen praktischen Nutzen die Evaluation und die Optimierung der Energieeffizienz eines Datenbanksystems hat. Hierfür wurden in Textabschnitt 3.3 mehrere Szenarien untersucht und die möglichen (positiven und negativen) Auswirkungen auf die Performance und den Energieverbrauch diskutiert.

Durch den Umstand, dass die Energieeffizienz von Datenbanksystem erst seit wenigen Jahren im wissenschaftlichen Fokus steht, liegen nur wenige Publikationen zu diesem Thema vor. Das äußert sich, dass es zwar viele Publikationen gibt, die sich mit der Evaluation und der Optimierung der Performance von Datenbanksystemen befassen, aber den Energieverbrauch und die

Energieeffizienz als vergleichende Metrik nicht betrachten. Wie in Textabschnitt 3.3 beschrieben, gibt es auch hinsichtlich der Optimierung der Energieeffizienz von Datenbanksystemen einen Disput, in welcher Weise vorgegangen werden soll. Einerseits werden die Performance und der Energieverbrauch als zwei unterschiedliche Optimierungsziele betrachtet und die daraus resultierende Energieeffizienz ist ein Kompromiss zwischen beiden. Andererseits wird argumentiert, dass das Optimum an Energieeffizienz im allgemeinen aus der performantesten Konfiguration eines Datenbanksystems abgeleitet werden kann. Das ist problematisch, denn beide Sichtweisen geben die Handlungsweise vor, wie die Energieeffizienz eines Datenbanksystem in der Praxis evaluiert und optimiert werden soll. Mit anderen Worten bedeutet es, dass es aufgrund der wissenschaftlichen Untersuchungen zwei praxisrelevante Optimierungsanweisungen gibt.

4 Experimentelle Ergebnisse

Um die Performance, den Energieverbrauch und schlussendlich die Energieeffizienz von Datenbanksystemen eingehender zu betrachten, wurden mehrere Experimente durchgeführt. Dabei wurden die Experimente so angelegt, dass erkennbar wird, welche Faktoren sich wie stark auf diese drei Aspekte auswirken. Zudem wurden sie so organisiert, um horizontale und vertikale *scale-up*-Szenarien näher zu untersuchen. Diese Szenarien dienen dazu, das Verhalten eines Datenbanksystems hinsichtlich der oben genannten drei Aspekte zu untersuchen, während ein Einflussparameter skaliert wird.

4.1 Untersuchung vertikaler *scale-up*-Szenarien

Im Kontext dieser Dissertation sind *vertikale scale-up-Szenarien* wie folgt definiert:

> **Definition 12: Vertikale *scale-up*-Szenarien**
> *Vertikale scale-up-Szenarien beschreiben das Verhalten und die Auswirkungen auf die Performance, den Energieverbrauch und der Energieeffizienz eines zentralen Datenbanksystems, wenn ein Einflussparameter skaliert wird. Dieser Einflussparameter ist im allgemeinen die Datenmenge, die das Datenbanksystem speichern, organisieren und abfragebereit halten soll.*

Um diese Szenarien detaillierter zu untersuchen, wurden insgesamt vier Experimente durchgeführt (Experiment A bis D), die aus Vergleichsgründen alle dieselbe technische Plattform nutzten. Die Charakteristiken dieser technischen Plattform sind in Tabelle 4.1 auf Seite 74 dargestellt. In dieser Tabelle ist auch ersichtlich, dass als Betriebssystem *Linux* und als Datenbanksystem *Postgresql* verwendet wurde. Die Gründe dafür sind, dass beide Softwarekomponenten bewährt, deren Architekturen sehr gut dokumentiert und beide als Quelltext einsehbar sind und damit spätere Analysen von experimentellen Ergebnissen gefördert werden.

4.1.1 Energieverbrauch der technischen Kernkomponenten

Das erste Experiment soll den Energieverbrauch der technischen Kernkomponenten eines zentralen Datenbanksystems untersuchen, wie sie in Textabschnitt 3.2 beschrieben sind. Der Hintergrund ist, dass im allgemeinen immer angenommen wurde, dass zum Beispiel die CPU für einen Großteil der insgesamt verbrauchten Energie verantwortlich ist. Ein zweites Beispiel ist, dass der Austausch des Massenspeichers in Form von magnetischen Festplatten durch SSDs zu einem geringeren Energieverbrauch führt.

Experiment A: Energieverbrauch der technischen Kernkomponenten
In diesem Experiment soll der Energieverbrauch der technischen Kernkomponenten Arbeitsspeicher, Massenspeicher, CPU und Mainboard individuell evaluiert werden, indem Stresstests für diese Komponenten durchgeführt werden. Das Ziel ist es, zu ermitteln, wie groß der Energieverbrauch der Komponenten relativ zum Gesamtenergieverbrauch ist.

Um dieses Experiment durchführen zu können, muss die interne Stromversorgung der technischen Plattform aus Tabelle 4.1 modifiziert werden. Diese Modifikation ist durch die Apparatur inspiriert, wie sie in [Ben11] publiziert wurde. Generell versorgt ein Netzteil die gesamten technische Plattform mit Energie, wobei drei unterschiedliche Spannungen übertragen werden: 3.3, 5 und 12 Volt. Diese drei Spannungen werden auch als "Stromschienen" bezeichnet. Durch diverse Spannungswandler werden dann die Spannungen erzeugt, die für andere Komponenten gebraucht werden. Ein Beispiel ist der Arbeitsspeicher, der im allgemeinen mit 1.5 Volt betrieben wird. Abbildung 4.1 zeigt unterschiedliche standardisierte Stecker und Buchsen, die genutzt werden, um im Inneren der technischen Plattform Energie zu den verschiedenen technischen Komponenten zu übertragen.

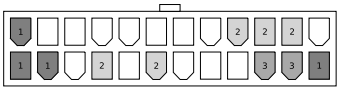
Komponente	Beschreibung
CPU	*Intel Core i7-860* mit 2.8 GHz Taktfrequenz
Hauptspeicher	3× *Samsung* DDR-2 2 GByte bei 800 MHz Taktfrequenz
Massenspeicher	3× *Hitachi* 1 TByte, 16 MByte Cache
Betriebssystem	*Ubuntu Server 11.10 x64*
Kernel	*Linux 3.0.0.17*
Datenbanksystem	*Postgresql 9.1*

Tabelle 4.1: Charakteristiken der technischen Plattform für die Experimente A bis D

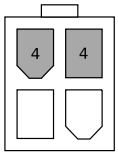
Die Modifikation umfasste das Zusammenfassen der Adern, die dieselbe Spannung führen, zu einer großen Ader. In Abbildung 4.1(b) und 4.1(c) sind diese Adern mit Nummern versehen. Ingesamt entstanden so vier große, spannungsführende Adern mit konstanter Länge, die danach wieder in Einzeladern aufgetrennt wurden (sogenannten Multiplexing und Demultiplexing der Adern). Durch Kenntnis der Spannung und Länge der großen Adern konnte dann mittels des Spannungsabfalls auf die momentane elektrische Leistung dieser "Stromschiene" geschlossen werden.

Nummer	Spannung in Volt
1	3.3
2 und 5	5
3,4 und 6	12

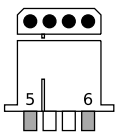
(a) Spannungsschienen



(b) ATX-Stecker



(c) ATX12



(d) Molex

Abbildung 4.1: Standardisierte Anschlüsse zur Strom- und Spannungsversorgung: die entsprechenden Pins aus den Anschlüssen in den Abbildungen 4.1(b) bis 4.1(d) sind nummeriert und können in Tabelle 4.1(a) nachgeschlagen werden.

Es ist auch bekannt, welche der Kernkomponenten CPU, Mainboard, Massen- und Hauptspeicher durch welche der Stromschienen versorgt werden. Durch die Modifikation ist es also möglich, den momentanen elektrischen Leistungsverbrauch pro Kernkomponente zu messen. Im Vergleich zu anderen Methoden zur elektrischen Leistungsbestimmung, zum Beispiel durch eine Strommesszange, ist die hier beschriebene Methode äußerst präzise und erlaubt auch Wechselwirkungen der Kernkomponenten untereinander exakt zu messen [Ahl15].

Solche Wechselwirkungen können entstehen, wenn die diversen Mechanismen genutzt werden, die unabhängig voneinander versuchen, den Energieverbrauch einzelner Komponenten einzuschränken. Dabei werden zwei Ansätze verfolgt:

1. Der erste Ansatz sieht vor, Teile einer technischen Komponente abzuschalten, sofern und sobald sie kaum oder gar nicht mehr genutzt wird. Der dadurch resultierende geringere Energieverbrauch richtet sich also nach einer Regelkurve mit mehreren Lastpunkten. Teilweise wird dieser Ansatz auch durch die Treiber im Betriebssystem beeinflusst; er ist also einstellbar.

   Prominente Beispiele sind moderne Prozessoren, Massenspeicher und spezielle Arbeitsspeicher im Serverumfeld. Bei ersterem wird je nach Auslastung Teile des Prozessors in einen Schlafmodus versetzt, die Versorgungsspannung gesenkt und die Taktung schrittweise heruntergeregelt. Bei Massenspeichern, zum Beispiel bei magnetischen Festplatten, wird nach einer einstellbaren Zeit der Inaktivität die Rotation der Speicherplatten eingestellt und die Steuerelektronik heruntergefahren. Arbeitsspeicherkomponenten, die ihre Auslastung nachverfolgen, können ähnlich wie Prozessoren ihre Taktung und ihre Kernversorgungsspannung senken.

   All diesen Technologien ist gemein, dass die Energiesparmechanismen recht schnell wieder aufgehoben werden können. Das bedeutet, dass ein Anwender relativ schnell wieder auf das volle Potential zurückgreifen können, wenn die Last ansteigt.

2. Der zweite Ansatz ist rein software-orientiert und wird genutzt, um eine technische Komponente besser auszunutzen. Das sind zum Beispiel alle Algorithmen, die den Zugriff auf zentrale Betriebsmittel verwalten, also sogenannte *scheduler*. Ein prominentes Beispiel sind diejenigen, die die Rechenzeit einer oder mehrerer CPU auf die Prozesse verteilen (sogenannte *Prozess-Scheduler*)

   Ein anderes Beispiel ist *Hyperthreading*: neben den physikalisch vorhandenen Prozessorkernen von *Intel*-Pozessoren mit dieser Technologie wird dieselbe Anzahl virtuell gegenüber dem Betriebssystem bekannt gegeben. Eingehende Instruktionen werden so umverteilt, dass die physikalischen Prozessorenkerne besser ausgelastet werden. Allerdings entfaltet diese Technologie nur mit einem geeigneten Prozess-Scheduler ihr volles Energiesparpotential.

Zur Durchführung des Experiments A wurden zunächst alle nicht notwendigen Dienste und Komponenten des Betriebssystems abgeschaltet. Das betrifft insbesondere das Datenbanksystem *Postgresql* der getesteten technischen Plattform nach Tabelle 4.1. Danach wurden individuelle Stresstests durchgeführt, um die technische Kernkomponenten einzeln zu belasten. Die experimentellen Ergebnisse dieser Stresstests sind in Abbildung 4.2 auf Seite 76 dargestellt.

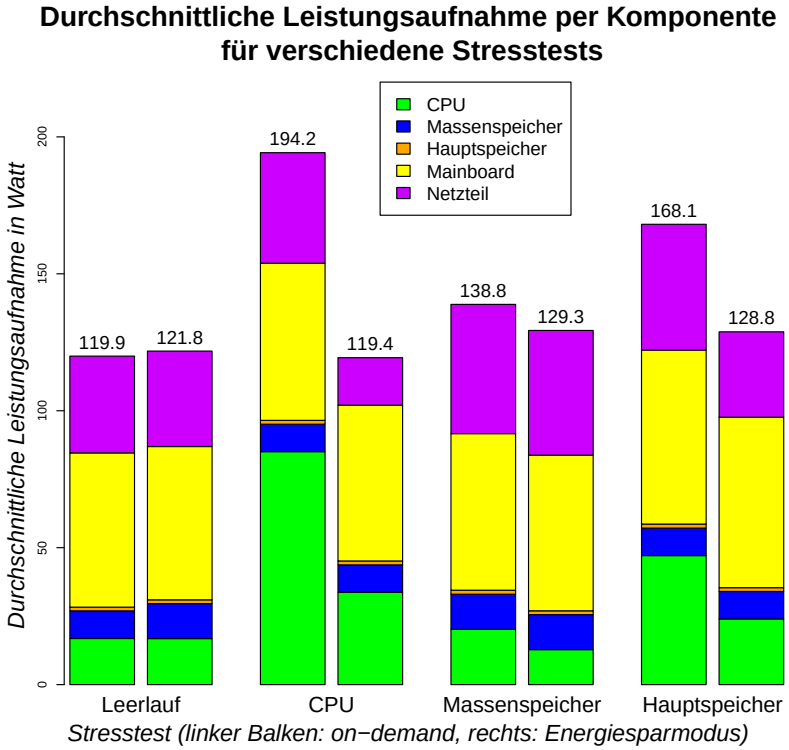


Abbildung 4.2: Mittlerer Leistungsverbrauch pro Kernkomponente für verschiedene Stresstests für Experiment A

In Abbildung 4.2 sind vier Balkenpaare erkennbar. Dabei stellt immer der linke Balken eines Balkenpaars den Energieverbrauch der technischen Komponenten mit deaktivierten Energiesparmechanismen und dem Standard-Prozess-Scheduler *on-demand* von *Linux* dar. Der rechte Balken eines Balkenpaars zeigt dann den Energieverbrauch bei aktivierten Energiesparmechanismen und dem *powersave*-Prozess-Scheduler von *Linux*. In Abbildung 4.2 wird auch der Energieverbrauch des Netzteils dargestellt, der sich aus der Differenz des gesamten zugeführten Energie und dem Energieverbrauch der gemessenen Kernkomponenten errechnet.

Um Vergleichswerte für die einzelnen Stresstests bereitzustellen, wurde der Energieverbrauch der Kernkomponenten gemessen, wenn die getestete technische Plattform nicht belastet wird (Leerlauf). Die Energieverbrauchswerte sind in Abbildung 4.2 als Balkenpaar mit der Bezeichnung “Leerlauf” dargestellt. Dabei zeigt sich, dass die Differenz der durchschnittliche elektrische Leistungsaufnahme sich kaum ändert, wenn die Energiesparmechanismen aktiviert werden. Im Gegenteil, sie erhöhen den Energieverbrauch sogar etwas.

Mittels des Kommandozeilenprogramms `burnMMX` wurde dann ein Stresstest der CPU durchgeführt. Dieses Programm lastet alle Teile der CPU aus, also auch selten genutzte Erweiterungen wie *AVX* oder *MMX*. Da die getestete technische Plattform eine CPU mit acht Prozessorkernen besitzt, wurde suksessiv die Anzahl der parallel ausgeführten `burnMMX`-Instanzen auf acht erhöht und die elektrische Leistungsaufnahme der Kernkomponenten gemessen. In Abbildung 4.2 sind die experimentellen Messergebnisse gemittelt als Balkenpaar mit der Bezeichnung “CPU” über alle acht Einzelmessungen dargestellt. Die Einzelmessungen pro Prozessorkern ist in Abbildung 4.3 auf Seite 77 dargestellt.

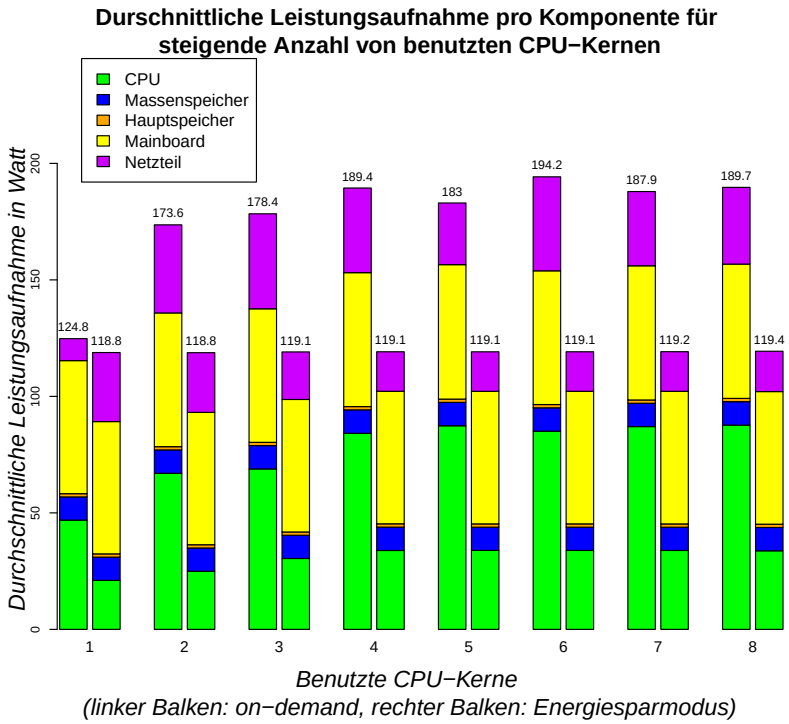


Abbildung 4.3: Leistungsverbrauch pro Kernkomponente für CPU-Stresstest

Auf Abbildung 4.3 ist erkennbar, dass der Energieverbrauch mit jedem genutzten physikalischen Prozessorkern ansteigt. Beginnend mit dem fünften Prozessorkern verbleibt der Energieverbrauch relativ konstant. Der Grund dafür ist, dass durch die *Hyperthreading*-Technologie die Prozessorkerne 5 bis 8 nur noch virtuell sind. Dieses Verhalten deckt sich mit den Messergebnisse aus [THS10].

Um den maximalen Energieverbrauch der Massenspeicher zu messen, wurde das Kommandozeilenprogramm `iozone` genutzt. Es führt eine Serie von Lasttests aus, die unterschiedliche Zugriffsstrategien, Puffer und Caches kombinieren, um den Datendurchsatz der Massenspeicher zu testen. Der Energieverbrauch während dieser Testserien wurde gemessen und ist in Abbildung 4.2 als Balkentupel mit der Bezeichnung “Massenspeicher” dargestellt.

Um den Arbeitsspeicher maximal zu belasten, wurde das Kommandozeilenprogramm `memtester` genutzt, dass durch unterschiedliche Zugriffsmuster prüft, ob Daten korrekt in den Arbeitsspeicher geschrieben und wieder gelesen werden. Analog zu dem vorherigen Experiment wurde auch hier der Energieverbrauch während der Arbeitsspeichertests gemessen. Er ist in Abbildung 4.2 als Balkentupel mit der Bezeichnung “Hauptspeicher” erkennbar. Im Vergleich zu den anderen Stresstests sind auch bei maximaler Auslastung des Arbeitsspeichers keine messbaren Auswirkungen auf den Energieverbrauch bestimmbar (vgl. dazu auch [ZZ10, SD95]).

4.1.2 Ergebnisse des *TPC-H*-Benchmarks

Das zweite Experiment, dass auf der technischen Plattform nach Tabelle 4.1 ausgeführt wurde, untersucht das Skalierungsverhalten des relationalen Datenbanksystems *Postgresql*. Dabei wird der *TPC-H*-Benchmark verwendet, wie er bereits in Textabschnitt 3.1 beschrieben wurde. Prinzipiell definiert dieser Benchmark 22 einzelne Datenbankabfragen, die analytisch auf die Datenbank zugreift (*OLAP*). Der Benchmark nutzt dabei die Antwortzeit des Datenbanksystems, um alle Einzelabfragen komplett zu verarbeiten und die Abfrageergebnisse zurückzuliefern.

Primär stellt sich zunächst die Frage, welche Faktoren beziehungsweise Konfigurationsmöglichkeiten hauptsächlich die Performance beeinflussen können. Dafür wurde das verwendete Datenbanksystem *Postgresql* eingehend analysiert. Die Analyse ergab folgende Faktoren:

1. der Anteil am Arbeitsspeicher, der exklusiv *Postgresql* zur Verfügung gestellt wird, um Datenbankoperationen durchzuführen
2. die Größe von diversen Puffern und Caches, zum Beispiel um Ergebnisse von Datenbankabfragen zu sortieren
3. Parameter für die Erstellung des Ausführungsplanes (sogenannter *query planner*)
4. die Verbindungsart (Einfach- oder Mehrfachverbindung)

Hinsichtlich der Verbindungsart wird zwischen einfachen und mehrfachen Verbindungen unterschieden (*single session* versus *multi session*). Bei ersterer wird eine einzige Verbindung zu *Postgresql* aufgebaut[1] und alle Datenbankoperationen werden sequentiell ausgeführt. Danach wird die Verbindung geschlossen. Bei dieser Verbindungsart können Zwischenergebnisse in Puffern und Caches gespeichert werden. Nachfolgende Datenbankoperationen können von ihnen profitieren, wodurch die Performance gesteigert werden kann. Bei mehrfachen Verbindungen wird für jede Datenbankoperation eine einzige Verbindung aufgebaut und nach Ausführung geschlossen. Damit wird die Nutzung von Zwischenergebnissen ausgeschlossen.

> **Experiment B: Skalierungsverhalten von *Postgresql* anhand des *TPC-H*-Benchmarks**
> Dieses Experiment soll das Skalierungsverhalten eines Datenbanksystems hinsichtlich der Performance, des Energieverbrauches und der Energieeffizienz mittels des *TPC-H*-Benchmarks evaluieren.
>
> Dabei soll insbesondere untersucht werden, welches Verhalten sich ergibt, wenn die folgenden Einflussfaktoren skaliert werden: Größe der Datenmenge, die durch das Datenbanksystem gespeichert und verwaltet werden soll; die Zugriffsart auf auf die Datenmenge (Einfach- oder Mehrfachverbindung) und die Größe des zugesicherten Arbeitsspeichers, der exklusiv für das Datenbanksystem reserviert wird.

Zur Durchführung von Experiment B wurde der Datengenerator des *TPC-H*-Benchmarks genutzt, um drei verschieden große Datenbanken[2] zu erstellen (1, 5 und 10 GByte Daten ohne Indizes). Diese Größen wurden so gewählt, dass die generierten Datensätze komplett, nahezu und unter keinen Umständen in den Anteil vom gesamten Arbeitsspeicher passen, den *Postgresql* zugewiesen wurde, um die 22 Datenbankabfragen des *TPC-H*-Benchmarks zu verarbeiten. Wie in Tabelle

[1] Es ist erwähnenswert, dass *Postgresql* jede Verbindung in einem eigenen Prozess ausführt und komplett isoliert. Das bedeutet, dass die Prozesse untereinander keinerlei Daten austauschen und nur auf einen kleinen gemeinsam genutzten Arbeitsspeicherbereich zur Synchronisierung und zur Wahrung des *ACID*-Paradigmas zugreifen.

[2] Die Tabellenstruktur und Datenbankanfragen sowie deren Semantik des *TPC-H*-Benchmarks ist in [KE11, S.714 ff.] dargestellt und beschrieben.

4.1 erkenntlich, verfügte die getestete technische Plattform über insgesamt 6 GByte Arbeitsspeicher, daher wurden der Anteil auf 1, 2.5 und 5 GByte Arbeitsspeicher festgelegt.

Somit entstanden 18 experimentelle Testreihen (3 Datenbankgrößen $\times$ 3 Arbeitsspeichergrößen $\times$ 2 Verbindungsarten), die jeweils dreimal durchgeführt wurden. Insgesamt sind somit 54 einzelne Experimente durchgeführt wurden. Die experimentellen Ergebnisse im Bezug auf die Antwortzeit und dem gemessenenen Energieverbrauch wurden dann gemittelt.

Betrachtung der Performance In Abbildung 4.4 auf Seite 80 ist die durchschnittliche Performance der 18 Testreihen von Experiment B dargestellt. Die Performance ergibt sich laut Gleichung 3.1 von Seite 54 als die Inverse der Antwortzeit, da nicht die Antwortzeiten der einzelnen Datenbankabfragen von Interesse sind, sondern die Antwortzeit des gesamten Benchmarks. Die durchschnittlichen Antwortzeiten sind als Abbildung im Anhang A.1.1 auf Seite 225 zu finden. Die dortige Abbildung folgt demselben Aufbau wie Abbildung 4.4, allerdings sind die durchschnittlichen Antwortzeiten numerisch ablesbar.

Abbildung 4.4 gliedert sich in drei Teile, die durch vertikale gestrichelte Linien getrennt sind. Diese Teile repräsentieren die Messergebnisse für den Anteil am gesamten Arbeitsspeicher, dem *Postgresql* exklusiv zur Verfügung stand (1, 2.5 und 5 GByte). Pro Teil sind dann die Performancewerte für jede getestete Datenbankgröße von 1, 5 und 10 GByte dargestellt, wobei sie für jede Verbindungsart untergliedert werden. In Abbildung 4.4 ist zu beachten, dass die y-Achse logarithmisch eingeteilt ist, um die unterschiedlichen Ergebnisse besser sichtbar zu gestalten.

Die Analyse von Abbildung 4.4 hinsichtlich der Performance zeigt zum einen, dass sich der Anteil des exklusiv reserviertem Arbeitsspeicher für das Datenbanksystem sehr stark auf die Performance auswirkt. So ist in Abbildung 4.4 klar erkennbar, dass die Performance umso höher ausfällt, je weniger Arbeitsspeicher exklusiv für *Postgresql* reserviert wurde. Der Grund war, dass *Postgresql* als ein Datenbanksystem charakterisiert werden kann, das stark Massenspeicherorientiert ist, da seine Ursprünge historisch in eine Zeit fallen, in der Arbeitsspeicher sehr teuer war und wenig Speicherkapazität besaß. Arbeitsspeicher wird von *Postgresql* primär dafür verwendet, die Zugriffe auf den Massenspeicher zu puffern, zum Beispiel für Datenbank- und Indexdateien. Die Annahme, dass sich die Performance mit exklusiv reserviertem Arbeitsspeicheranteil steigen sollte, hat sich nicht bewahrheitet. Im Gegenteil, Abbildung 4.4 zeigt eindeutig, dass sie sinkt. Dazu sei angemerkt, dass die Pufferung von Massenspeicherdaten zentral durch das zugrunde liegende Betriebssystem *Linux* der getesteten technischen Plattform vollzogen wird, wofür laut Standardkonfiguration circa 90 Prozent des verfügbaren Arbeitsspeichers verwendet werden. Durch die exklusive Reservierung von Arbeitsspeicher für *Postgresql* stand dieser dem Betriebssystem nicht mehr zur Verfügung. So zeigte sich, dass zum Beispiel für Datenbankabfrage 1, 9 und 21 des verwendeten *TPC-H*-Benchmarks das Betriebssystem durch heftige Pufferoperationen (sogenanntem *swapping*) die Ausführung dieser Datenbankabfragen für sehr lange Zeitperioden komplett ausgesetzt hat. Dies hat einen äußerst negativen Einfluss auf die Antwortzeit und damit auf die Performance des genutzten Benchmarks.

Um dieses Verhalten gründlicher zu analysieren, wurde auch das Pufferverhalten von *Postgresql* analysiert. Das Ergebnis dieser Analyse zeigt, dass dieses Datenbanksystem zwar Zugriffe auf den Massenspeicher puffert, aber diese Puffer nur selten nutzt. Das Ziel von Experiment B ist unter anderem, die Puffernutzung durch Nutzung von Einfach- und Mehrfachverbindungen explizit zu untersuchen. Da wie erwähnt die Puffer zwar existieren, aber kaum genutzt werden, so ist es nicht erstaunlich, dass sie die Performancewerte im Bezug auf die Verbindungsart nicht signifikant unterscheiden.

Das kann in Abbildung 4.4 daran erkannt werden, wenn man die Performancewerte einer Testreihe, also genau eines Balkenpaares, miteinander vergleicht. Hier sind kaum Unterschiede erkennbar.

Zudem zeigen die Ergebnisse der Analyse über das Pufferverhalten, dass vorhandene Datenbankindizes kaum genutzt wurden, um die einzelnen Datenbankabfragen des verwendeten Benchmarks zu beschleunigen. Es zeigte sich ein gegenteiliges Ergebnis: bei ungefähr der Hälfte der Datenbankabfragen wurden die Inhalte der Datenbankdateien sequentiell durchsucht (sogenannte *row scans*), was einen immensen Zugriff auf den Massenspeicher nach sich zieht. Durch das oben beschriebenen Pufferverhalten von Massenspeicherdaten relativiert sich jedoch die Auswirkungen auf die Performance. In Abbildung 4.4 ist dies zu erkennen, wenn man die Performanceergebnisse für jeden Arbeitsspeicherbereich getrennt betrachtet: bis auf eine Ausnahme für das Balkenpaar am rechten Rand ändern sich die Performancewerte innerhalb eines Bereiches kaum. Zudem relativiert sich auch die Zugriffsart: betrachtet man nur genau ein Balkenpaar auf einmal, so fällt auf, dass sich die Performance bis auf die erwähnte Ausnahme ebenfalls kaum ändert.

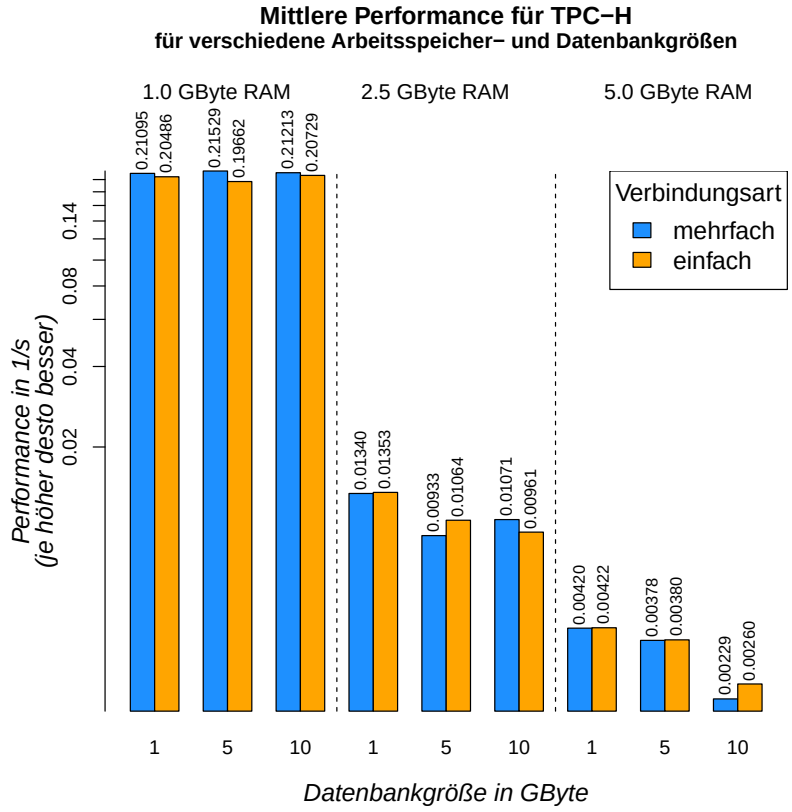


Abbildung 4.4: Performance des *TPC-H*-Benchmarks für verschiedene Datenbank- und Arbeitsspeichergrößen

Postgresql bietet Möglichkeiten, die Nutzung von Datenbankindizes und Puffern zu beeinflussen. Das bedeutet genauer, dass man prinzipiell *Postgresql* dazu zwingen kann, diese beiden Aspekte auch dann zu nutzen, wenn sie laut Zugriffsplan für eine Datenbankabfrage nicht sinnvoll sind. Von diesen Möglichkeiten wurde Gebrauch gemacht, um zu evaluieren, ob damit die experimen-

tell gewonnenen Performancewerte verbessern lassen. Daher wurden außerhalb der eigentlichen Zielsetzung von Experiment B zusätzliche Testreihen mit den geänderten Einstellungen durchgeführt. Ein Vergleich zeigt, dass dadurch die Performance im Mittel um vier Prozent sank.

Dazu sei bemerkt, dass die Komponenten von *Postgresql*, die für die Validierung und Optimierung der Datenbankabfragen und die Erstellung der Zugriffspläne (*query optimizer* und *query planner*) von den Entwicklern äußerst intelligent implementiert sind. Dazu wurde ein Verfahren implementiert, dass als *genetic query optimization* bezeichnet wird. Nach *Bennett et al.* sowie *Dong und Liang* werden bei diesem Verfahren genetische Algorithmen so angepasst, dass sie dafür genutzt werden können, aus *allen möglichen* Zugriffsplänen denjenigen auszuwählen, der den geringsten Aufwand bei gleichzeitiger Maximierung der Ausführungsgeschwindigkeit verspricht [BFI94, DL07]. Dazu nutzt das implementierte Verfahren einstellbare Kosten, zum Beispiel für die Abfrage eines Index oder das Lesen eines Datensatzes, zur Bewertung der Zugriffspläne. Diese Kosten wurde von den Entwicklern von *Postgresql* über die Jahre angepasst und sehr genau justiert. Im vorliegenden Fall wurden die Kosten für die Benutzung eines Index und von gepufferten Datenbankinhalten so stark gesenkt, dass sie in jeder Situation bevorzugt werden. Das ist in der Theorie plausibel, hatte aber laut den experimentellen Ergebnissen den gegenteiligen Effekt.

Betrachtung des Energieverbrauches Da die technische Plattform, auf der die Testreihen für Experiment B durchgeführt wurden, auch die Messung der individuellen elektrische Leistungsaufnahme pro Kernkomponente (Haupt- und Massenspeicher, Mainboard, CPU) erlaubte, konnten detaillierte detaillierte Verbrauchswerte gewonnen werden. Für alle 18 Testreihen sind sie in Abbildung 4.5 auf Seite 82 dargestellt.

In Abbildung 4.5 ist erkennbar, dass die Testreihen, die nur eine einfache Verbindung verwendeten, eine größere elektrische Leistungsaufnahme haben als diejenigen, die eine mehrfache Verbindung nutzten. Betrachtet man hingegen die gesamte mittlere elektrische Leistungsaufnahme für jede Verbindungsart getrennt, so sind kaum Unterschiede feststellbar. Dazu vergleicht man in Abbildung 4.5 nur diejenigen Balken, die gleich bezeichnet sind, also nur die jeweiligen mit der Bezeichnung “Einfachverb.” beziehungsweise “Mehrfachverb.”. Zudem fällt in Abbildung 4.5 auf, dass die technische Kernkomponente, die die größte elektrische Leistungsaufnahme hatte, das Mainboard ist und nicht die CPU, wie generell vermutet wird [THS10]. Dazu trug auch das beschriebene Pufferverhalten von *Postgresql* bei. Man kann davon ausgehen, dass die Datenbankabfragen eine höhere CPU-Auslastung verursacht hätten, wenn sie nicht von den erwähnten *swapping*-Operationen des Betriebssystems unterbrochen worden wären. Zudem sind diese Operationen generell nicht sehr CPU-lastig, wodurch sie dessen geringer Anteil am gemessen gesamten Energieverbrauch erklärt.

Betrachtung der Energieeffizienz Anhand von Gleichung 3.3 auf Seite 64 für die Energieeffizienz ist es möglich, mittels der experimentellen Ergebnisse für die Performance und dem Energieverbrauch die mittlere Energieeffizienz für alle 18 Testreihen von Experiment B zu errechnen. Sie sind in Abbildung 4.6 auf Seite 83 dargestellt. In Abbildung 4.6 ist zu beachten, dass die x-Achse logarithmisch eingeteilt ist, um die Unterschiede der Energieeffizienzwerte besser erkennen zu können.

Die Annahme, dass die Testreihen, die eine einzelne Verbindung verwendeten, aufgrund besserer Puffernutzung und damit besserer Performance eine bessere Energieeffizienz als dejenigen Testreihen mit Mehrfachverbindungen haben, hat sich nicht bestätigt. Aufgrund des Pufferverhaltens von *Postgresql*, das bereits beschrieben wurde, hat sich gezeigt, dass die Performance sich nicht signifikant unterschied, aber der Energieverbrauch dennoch gestiegen ist. Die Differenz beim

Energieverbrauch betrug im Mittel sieben Prozent. Das führte dazu, dass sich die errechneten Werte für die Energieeffizienz im Mittel nur um drei Prozent unterschieden. Dieser Sachverhalt ist in Abbildung 4.6 erkennbar, wenn man die Energieeffizienzwerte eines Balkenpaares vergleicht. Hier zeigt sich, dass es im Bezug auf die Energieeffizienz unerheblich ist, welche Verbindungsart man nutzt.

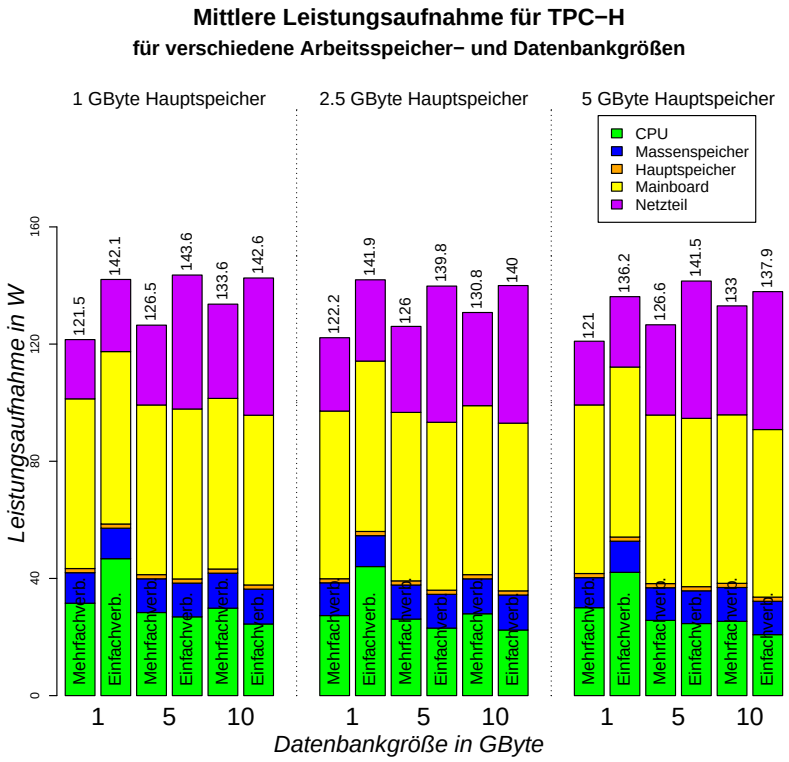


Abbildung 4.5: Energieverbrauch pro Kernkomponente für *TPC-H*

Es ist auch interessant, wie sich die Energieeffizienz gegenüber der Performance verhält. Dazu normalisiert man die errechneten Energieeffizienzwerte der 18 Testreihen von Experiment B in im Bereich zwischen Null und Eins, wobei die Null den kleinsten und die Eins den größten Wert repräsentiert. Nach demselben Verfahren werden auch die dazugehörigen Performancewerte normalisiert. Beide Normalisierungen können dann grafisch wie in Abbildung 4.7 auf Seite 83 dargestellt werden und werden als *Energieeffizienz-vs.-Performance*-Diagramme bezeichnet. In Abbildung 4.7 sind zwei Teilgrafiken dargestellt, wobei Teilgrafik 4.7(a) die Normalisierungen für alle 18 Testreihen darstellt, während Teilgrafik 4.7(b) die Normalisierungen nach Verbindsart unterscheidet.

Betrachtet man Abbildung 4.7(a) so fällt auf, dass sich ein Großteil der dargestellten Werte im unteren linken Sektor nahe am Koordinatenursprung befinden. Das bedeutet, dass einerseits die Mehrheit der Testreihen recht unperformant gegenüber der performantesten Testreihe sind und zudem gleichzeitig auch eine vergleichsweise schlechtere Energieeffizienz besitzen. Zusätzlich ist auch erkennbar, dass die performanteste Testreihe gleichzeitig die energieeffizienteste ist, wenn man den Wert auf Koordinate (1,1) betrachtet.

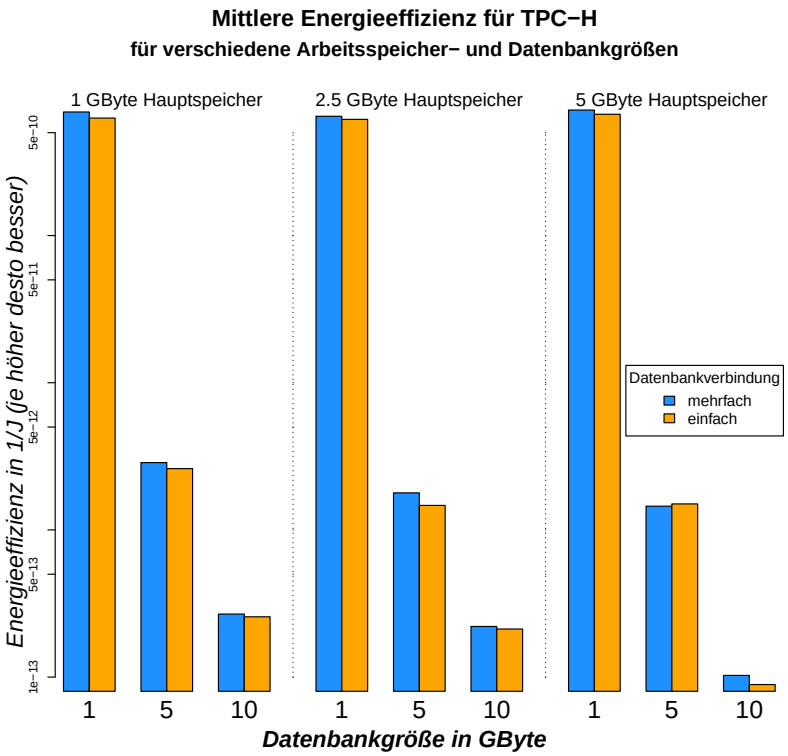
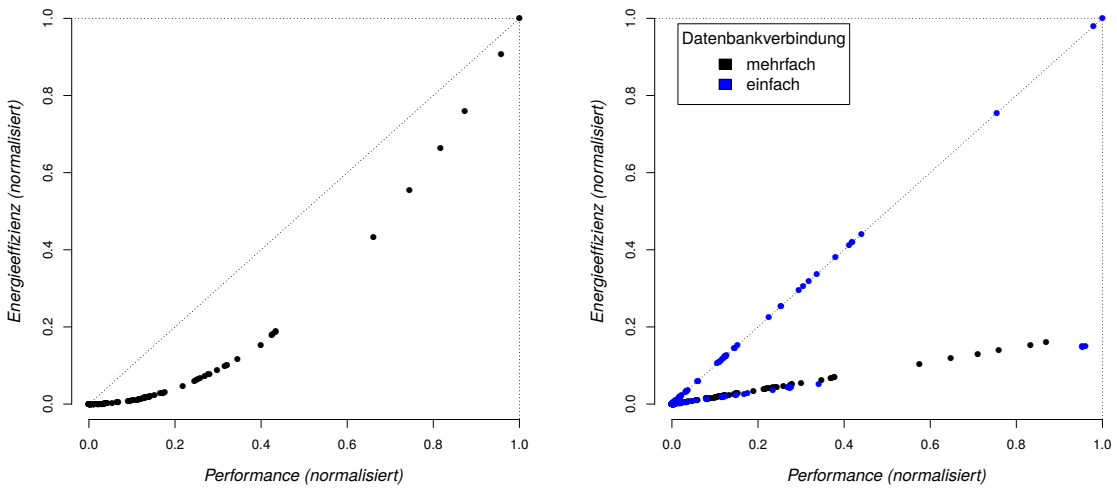


Abbildung 4.6: Mittlere Energieeffizienz für *TPC-H*[3]



(a) für alle experimentellen Ergebnisse

(b) nach Verbindungsart gruppiert

Abbildung 4.7: Energieeffizienz versus Performance für *TPC-H*

[3] Es sei zu Abbildung 4.6 bemerkt, dass auf eine Annotation der numerische Werte aufgrund der Darstellbarkeit verzichtet wurde. Die numerischen Werte sind im Anhang A.1.1 verzeichnet.

Wie bereits erwähnt, können die experimentellen Ergebnisse für die Performance und die Energieeffizenz jeweils getrennt normalisiert werden und in einem gemeinsamen Diagramm dargestellt werden. In Abbildung 4.7(b) werden beide Normalisierungen gewissermaßen überlagert dargestellt. Das erlaubt einen Vergleich, wie sich die Energieeffizenz gegenüber der Performance verhält, aber im Gegensatz zu Abbildung 4.7(a) nach Zugriffsart differenziert. In Abbildung 4.7(b) ist deutlich erkennbar, dass bei beiden Verbindungsarten die Energieeffizienz mit der Performance skaliert. Zudem ist auch sichtbar, das bei vielen Testreihen mit Einzelverbindung eine höhere Energieeffizienz zu erkennen ist als bei denjenigen Testreihen mit Mehrfachverbindungen. Das bedeutet nicht, dass letztgenannte unperformant sind, sondern dass sie einen höheren Energieverbrauch zur Erreichung der Performance hatten. Analog zu Abbildung 4.7(a) zeigt sich auch in Abbildung 4.7(b), dass die performanteste Testreihe gleichzeitig die energieeffizienteste ist.

Insgesamt kann damit das experimentelle Ergebnis aus der Studie von *Tsirogiannis et al.* bestätigt werden [THS10]: "*the most energy-efficient configuration is typically the highest performing one.*" Ins Deutsche übersetzt zeigen die experimentelle Ergebnisse von Experiment B, dass diejenige Konfiguration des Datenbanksystems (= Testreihe), die am performantesten ist, auch die energieeffizienteste ist.

4.1.3 Ergebnisse des *StarSchema*-Benchmarks

Experiment B, das mitsamt dessen experimentellen Ergebnissen im vorherigen Textabschnitt beschrieben ist, hat den *TCP-H*-Benchmark genutzt. Dieser Benchmark steht aber in der Diskussion, nicht mehr realitätsnah zu sein. Nach [OOC09] betrifft die Kritik vor allem die Datenbankstrukturen in Form von Tabellen, die durch den Datengenerator des *TPC-H*-Benchmarks erzeugt werden. Das äußert sich darin, dass die Strukturen nicht so angelegt sind, wie sie moderne Datenbankentwurfsverfahren anlegen würden. Zudem sind die im *TPC-H*-Benchmark definierten Datenbankabfragen zu konservativ. Das bedeutet, sie sind dergestalt definiert wurden, um auf möglichst vielen Datenbanksystem ausgeführt zu werden.

Aufgrund der Kritik wurde der *StarSchema*-Benchmark (*SSB*) auf Basis von *TPC-H* konzipiert, um die Performance in Form der Antwortzeit eines relationalen Datenbanksystem auf eine Serie von Datenbankabfragen zu evaluieren. *SSB* nutzt dieselbe Datenbankstruktur wie *TPC-H*, aber nach [OOC09] wurden einige Tabellen entkoppelt, um Datenbanksystemen die Vereinigung von Tabellendatensätzen zu erleichtern (sogenannte *joins*). Zudem wurden die Datenbankabfragen so gestaltet, dass sie realistischer sind und unterschiedliche Überdeckungsgrade sowie die Nutzung von Indizes testen.

Insbesondere die Überdeckungsgrade sind recht interessant. Die 13 Datenbankabfragen von *SSB* sind in vier Gruppen eingeteilt, wobei mit jeder Gruppe der Überdeckungsgrad der Datenbankabfrage steigt. Das bedeutet, dass bei der ersten Gruppe nur eine Tabelle genutzt wird. Die Abfragen der zweiten Gruppe nutzen als Basis bereits die Vereinigung der Einträge aus zwei Tabellen. Dieses Vorgehen setzt sich bis zur vierten Gruppe fort. Die Abfragen der Gruppen sind zusätzlich so angelegt, dass sie sich aber nicht überdecken.

> **Experiment C: Skalierungsverhalten von *Postgresql* anhand des *Starschema*-Benchmarks**
> Dieses Experiment ist analog zu Experiment B, allerdings wird der *StarSchema*-Benchmark (*SSB*) genutzt. Die Zielstellung ist, das Skalierungsverhalten des relationalen Datenbanksystems *Postgresql* hinsichtlich der Performance, des Energieverbrauches und der Energieeffizienz zu evaluieren.

Aus Vergleichsgründen mit den experimentellen Ergebnissen von Experiment B sind dieselben 18 Testreihen auch für Experiment C durchgeführt worden. Das bedeutet, dass mit dem Datengenerator von *SSB* drei Datenbanken mit einer Größe von 1, 5 und 10 GByte ohne Datenbankindizes erstellt worden sind. Für diese drei Datenbankgrößen wurde dann jeweils der *SSB* mit Einzel- und Mehrfachverbindung durchgeführt, wobei dem verwendeten Datenbanksystem *Postgresql* jeweils 1, 2.5 und 5 GByte an Arbeitsspeicher exklusiv zugesichert wurde. Zudem wurde auch dieselbe technische Plattform für alle 18 Testserien genutzt, wie sie in Tabelle 4.1 auf Seite 74 dargestellt ist. Insgesamt korrespondiert die Durchführung von Experiment C mit der von Experiment B, mit der Ausnahme, dass ein anderer Benchmark verwendet wurde.

Betrachtung der Performance In Abbildung 4.8 auf Seite 86 sind die mittleren Performancewerte für die 18 durchgeführten Testreihen dargestellt. Wie bereits erwähnt, definiert *SSB* die Performance als die Inverse der Antwortzeit. Die gemittelten Antwortzeiten sind im Anhang A.1.2 grafisch dargestellt, wobei die Antwortzeiten numerisch ablesbar sind. Auch Abbildung 4.8 gliedert sich in drei Teile, die durch vertikale gestrichelte Linien getrennt sind. Diese Teile repräsentieren die Messergebnisse für den Anteil am gesamten Arbeitsspeicher, dem *Postgresql* exklusiv zur Verfügung stand (1, 2.5 und 5 GByte). Pro Teil sind dann die Performancewerte für jede getestete Datenbankgröße von 1, 5 und 10 GByte dargestellt, wobei sie für jede Verbindungsart untergliedert werden. Auch in Abbildung 4.8 ist zu beachten, dass die y-Achse logarithmisch eingeteilt ist, um die unterschiedlichen Ergebnisse besser sichtbar zu gestalten.

Da der *SSB* im Grunde dieselben Datenbankstrukturen wie der *TCP-H*-Benchmark benutzt, ist es vom Interesse, die experimentellen Performancewerte zu vergleichen. Dieses Vorgehen ist auch deshalb möglich, da dieselbe technische Plattform und dieselben Testverfahren im Sinne der 18 Testreihen genutzt wurden. Vergleicht man also die mittleren Performancewerte von Experiment B (Abbildung 4.4 auf Seite 80) mit denjenigen von Experiment C in Abbildung 4.8, so fällt auf, dass letzterer performanter ausgeführt wurde. Dabei sind nicht die numerischen Performancewerte vom Belang, sondern der Trend, der sich durch die dargestellten Balkenpaare entwickelt. Beim *TPC-H*-Benchmark sanken die Performancewerte deutlich ab, je mehr Arbeitsspeicher exklusiv für *Postgresql* reserviert wurde. Dieses Verhalten ist in Abbildung 4.8 auch erkennbar, allerdings in abgeschwächter Form. Zudem fällt auf, dass es hinsichtlich der Zugriffsart (Einzel- gegenüber Mehrfachverbindung) kaum Unterschiede feststellbar sind.

Aufgrund der Erfahrungen im Umgang mit *Postgresql*, die in Experiment B gemacht wurden, wurden auch die Ausführungspläne von Experiment C detailliert analysiert, um das Performanceverhalten aus Abbildung 4.8 zu erklären. So ergab die Analyse eine bessere Puffernutzung seitens *Postgresql*, wodurch sich eine bessere Performance im Vergleich zu Experiment B (*TCP-H*) ergab. Allerdings traten auch die Puffereffekte auf, wie sie bereits im vorherigen Textabschnitt 4.1.2 beschrieben sind. Das bedeutet, dass mit steigendem exklusiven Arbeitsspeicheranteil von *Postgresql* dem Betriebssystem weniger Arbeitsspeicher verbleibt, um Zugriffe auf den Massenspeicher zu puffern, auf dem die Datenbank- und Indexdateien von Experiment C gespeichert wurden. Dies führte zu heftigen Auslagerungsoperationen des Betriebssystems mit der Folge, dass die Ausführung des *SSB* ausgesetzt wurde. Die Analyse ergab, dass durch die bessere interne Puffernutzung von *Postgresql* bei *SSB* dieser Effekt erst bei den Testreihen einsetzten, in denen maximale Grenzen getestet wurden, zum Beispiel einen Arbeitsspeicheranteil von 5 GByte für *Postgresql* und eine Datenbankgröße von 10 GByte.

Die Analyse der Ausführungspläne zeigt auch ein überraschendes Verhalten im Bezug auf die Nutzung von Datenbankindizes. Der Datengenerator von *SSB* hat neben den Tabellenstrukturen und deren Datensätzen auch geeignete Datenbankindizes für die Tabellen erstellt. Die analysier-

ten Ausführungspläne zeigten aber, dass diese generell nicht genutzt wurden. Faktisch wurden alle Datenbankabfragen, die im Rahmen von Experiment C durchgeführt wurden, mit Hilfe von *row scans* ausgeführt. Im Bezug zu den Testreihen von Experiment C ist dementsprechend die Performance allein von der Leserate beziehungsweise vom Datendurchsatz der zugrunde liegenden Tabellendateien abhängig. Anhand der Ausführungspläne sind die mittleren Lese- beziehungsweise Durchsatzraten für alle Testreihen dieses Experimentes in Abbildung 4.9 auf Seite 87 dargestellt.

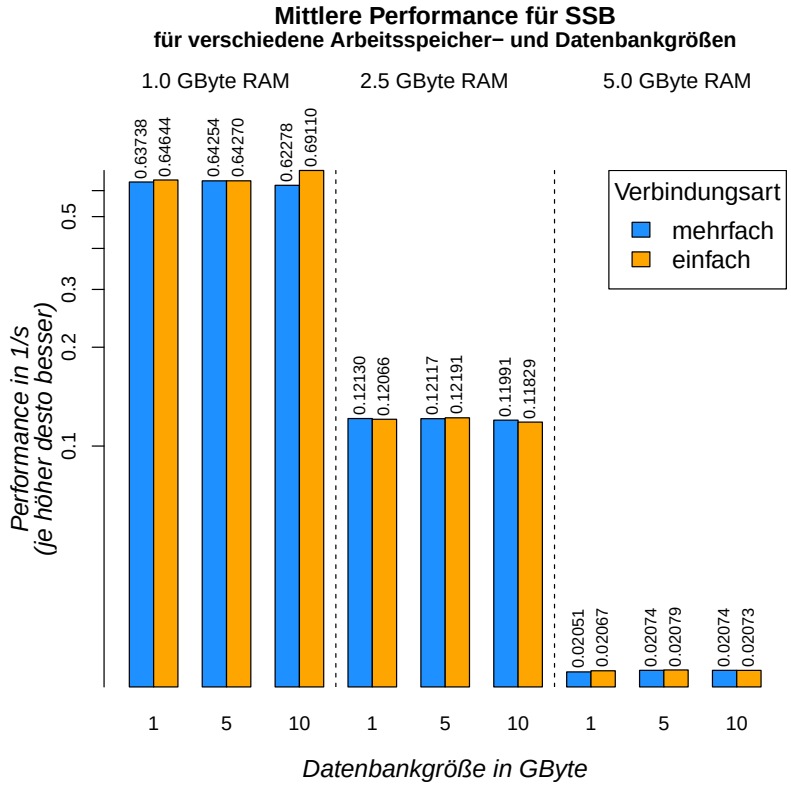


Abbildung 4.8: Performance des *StarSchema*-Benchmarks für verschiedene Datenbank- und Arbeitsspeichergrößen

Zur Erklärung von Abbildung 4.9 muss nochmals auf die eingangs erwähnten vier *SSB*-Gruppen eingegangen werden. Wie gesagt gibt die Gruppennummer an, aus wievielen Tabellen deren Inhalte vereinigt werden sollen. Für diese Vereinigung werden dabei die Tabellen genutzt, die in absteigender Reihenfolge die meisten Datensätze beinhalten. Zum Beispiel definiert *SSB*-Gruppe 4 als Basis die Vereinigung aus den vier größten *SSB*-Tabellen. Dieses Vorgehen wurde so gewählt, um mehrere Überdeckungsgrade hinsichtlich der Antwortzeit testen zu können. Innerhalb einer Gruppe sind dann die Datenbankabfragen so gestaltet, dass die zugrunde liegende Vereinigung durch eine ansteigende Anzahl von Bedingungen eingeschränkt wird.

Wie bereits erläutert, hängt die Performance der Testreihen von Experiment C durch generelle Nutzung von *rows scans* stark vom Datendurchsatz vom und zum Massenspeicher ab. In Abbildung 4.9 sind die Durchsatzraten für die 18 Testreihen dargestellt, wobei der Durchsatz als gelesene Datensätze pro Sekunde (“row scans/s”) auf der x-Achse abgebildet ist. Bei Betrachtung

dieser Abbildung ist erkennbar, dass der Durchsatz nicht von der Größe des Arbeitsspeichers abhängt, der für *Postgresql* exklusiv reserviert wurde. Zudem zeigt sich auch der beschriebene Puffereffekt ab einer getesteten Datenbankgröße von 10 GByte, wodurch sich die Durchsatzraten unabhängig von der Arbeitsspeicheranteilsgröße drastisch reduzieren. Insgesamt kann festgestellt werden, dass die dargestellten Performancewerte in Abbildung 4.8 mit den Durchsatzraten in Abbildung 4.9 korrelieren. Zudem kann auch konstatiert werden, dass aufgrund der generellen Nutzung der *row scans* die Art der Verbindung (Einfach- und Mehrfachverbindung) keinerlei Vorteile für die Performance bietet, wie sich in Abbildung 4.8 zeigt.

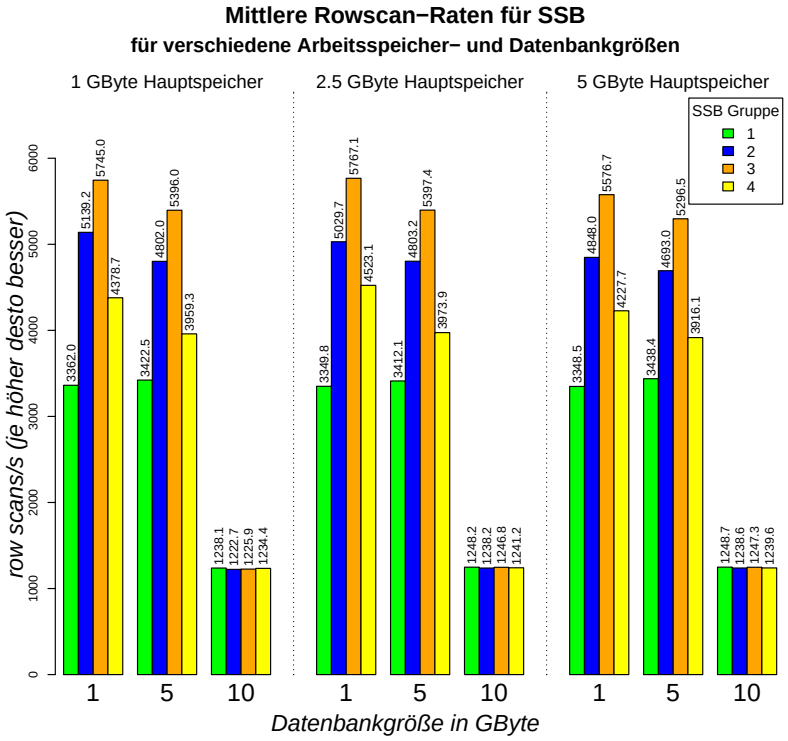


Abbildung 4.9: *SSB*-rowscans für verschiedene Datenbank- und Arbeitsspeichergrößen

Betrachtung des Energieverbrauches Das beschriebene Verhalten zeigt sich allerdings nicht beim Energieverbrauch der technischen Komponente der genutzten technischen Plattform für Experiment C. Da dieselbe technische Plattform wie in den Experimenten A und B genutzt wurde, konnte der elektrische Leistungsverbrauch der Kernkomponenten Arbeits- und Massenspeicher, CPU und Mainboard während der 18 Testreihen von Experiment C gemessen werden. Die durchschnittlichen Verbrauchswerte dieser Testreihen sind in Abbildung 4.10 auf Seite 88 dargestellt.

Wie in Abbildung 4.10 ersichtlich, unterscheiden sich die gemessenen und gemittelten Energieverbrauchswerte kaum. Die einzigen Ausnahmen bilden die Testreihen, in denen der *SSB* für eine Datenbankgröße von 5 und 10 GByte mit einer einzelnen Verbindung zum *Postgresql*-Datenbanksystem durchgeführt wurden. Hier weichen insbesondere die Werte des Gesamtenergieverbrauchs der technischen Plattform von denen der übrigen Testreihen deutlich ab. Aber auch hinsichtlich der Verbindungsart gibt es Unterschiede. Betrachtet man die elektrische Leis-

tungsaufnahme der CPU (der grün markierte Anteil in den Balken von Abbildung 4.10), so bemerkt man, dass sie höher bei den Testserien mit Einzelverbindung ausfallen, wenn man sie mit denjenigen mit Mehrfachverbindung vergleicht. Im Gegensatz dazu sind bei den anderen gemessenen technischen Kernkomponenten Massen- und Hauptspeicher sowie Mainboard keine Unterschiede erkennbar.

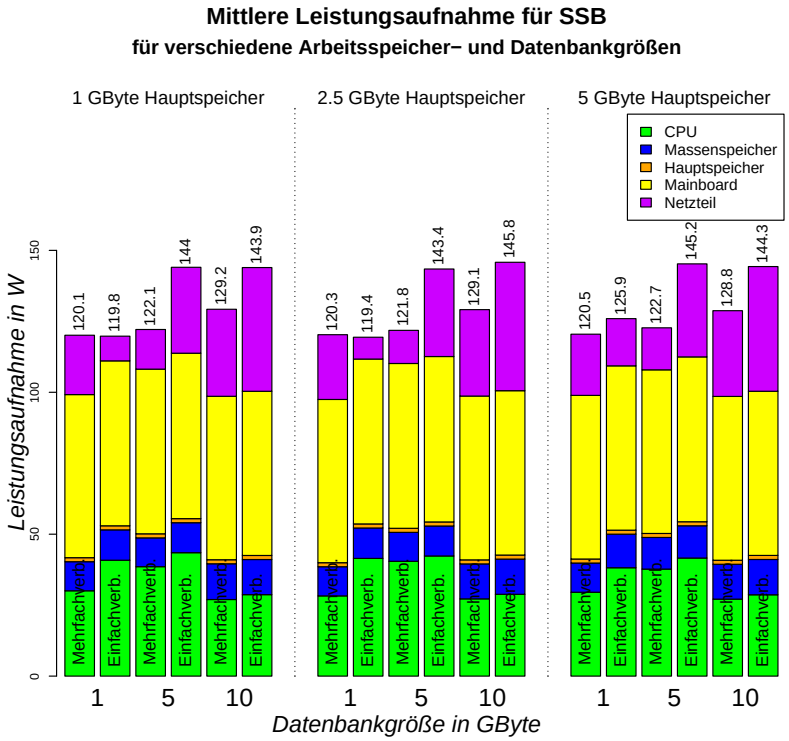


Abbildung 4.10: Energieverbrauch pro Kernkomponente für *SSB*

Betrachtung der Energieeffizienz Da experimentelle Ergebnisse sowohl für die Performance als auch für den Energieverbrauch für alle Testserien von Experiment C verfügbar waren, konnte auch die Energieeffizienz anhand von Gleichung 3.3 auf Seite 64 errechnet werden. Die durchschnittlichen Energieeffizienzwerte für alle 18 Testserien des Experiments C sind in Abbildung 4.11 auf Seite 89 dargestellt.

Im Vergleich mit den durchschnittlichen Energieeffizienzwerten von Experiment B (*TPC-H*, Abbildung 4.6 auf Seite 83) mit denjenigen von Experiment C in Abbildung 4.11 fällt ein deutlich positiverer Trend auf. Das betrifft nicht die numerischen Werte, sondern wie sich das Energieeffizienzverhalten mit steigender Datenbankgröße und steigendem Arbeitsspeicheranteil für *Postgresql* entwickelt. Der positivere Trend ist darauf zurückzuführen, dass die Performance der *SSB*-Testreihen gegenüber den *TPC-H*-Testreihen höher ausfällt. Vergleicht man zusätzlich die Energieverbrauchswerte der Testserien aus beiden Experimenten, so erkennt man kaum Unterschiede. Dazu muss man die numerischen Werte des Gesamtenergieverbrauches in Abbildung 4.5 auf Seite 82 mit denen auf Seite 88 vergleichen. Beide Aspekte, die höhere Performance und der gleichwertige Energieverbrauch, führen dazu, dass die Energieeffizienzwerte der *SSB*-Testreihen wie bereits erwähnt höher ausfallen.

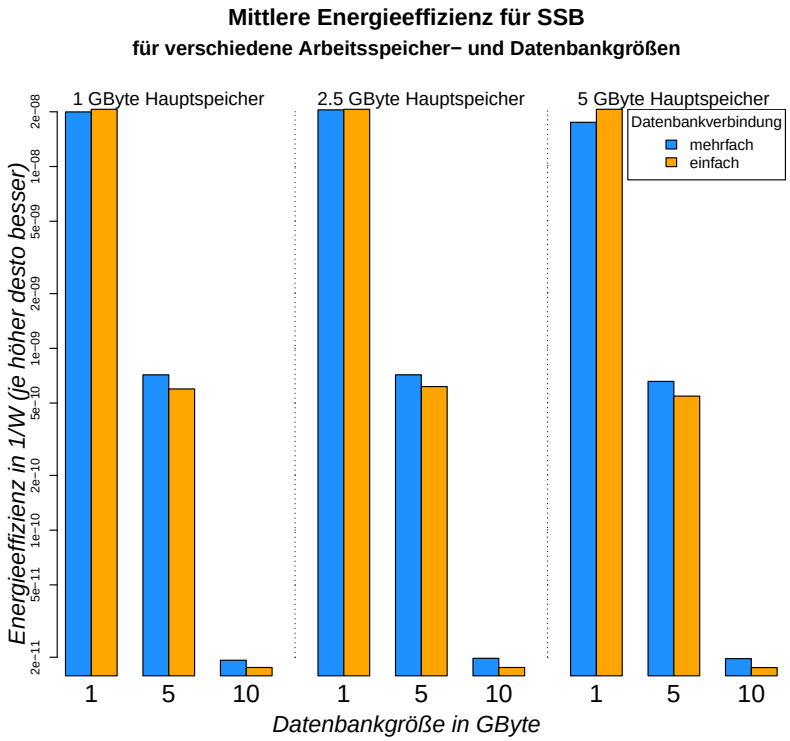


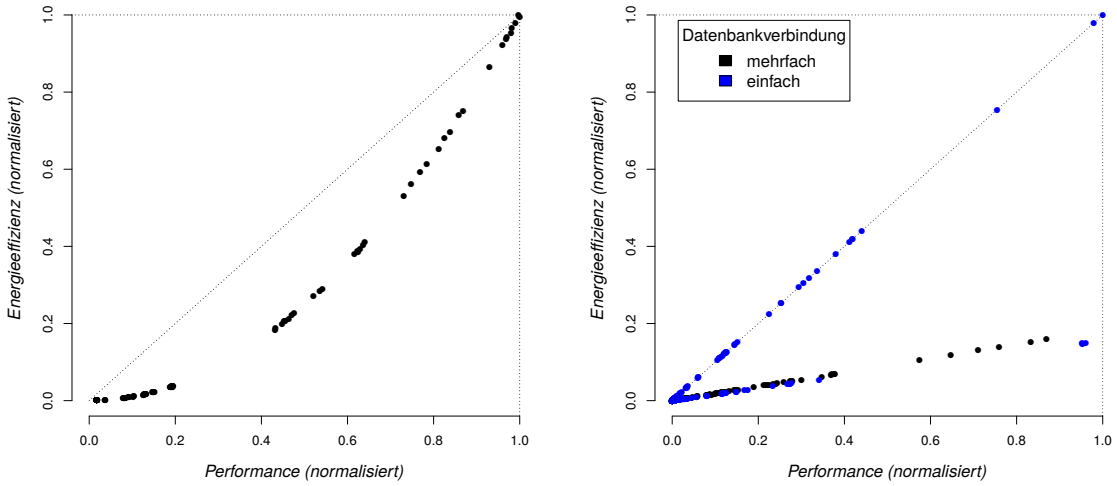
Abbildung 4.11: Mittlere Energieeffizienz für *SSB*[4]

Analog zu den Energieeffizienz- und Performancewerten von Experiment B ist es auch für Experiment C interessant, welches Verhalten sich zwischen der Performance und der Energieeffizienz der 18 *SSB*-Testreihen ergibt. Dazu wurden dieselben Normalisierungen vorgenommen, wie sie am Ende des vorherigen Textabschnittes 4.1.2 beschrieben sind. Dazu werden die durchschnittlichen Performance- und dazugehörigen Energieeffizienzwerte jeweils im Bereich Null und Eins normalisiert. Die Null repräsentiert den kleinsten und die Eins den größten beobachteten beziehungsweise errechneten Wert. Das Ergebnis ist in Abbildung 4.12 auf Seite 90 als Energieeffizienz-vs.-Performance-Diagramm dargestellt. Sie ist in zwei Teildiagramme unterteilt, wobei Abbildung 4.12(a) die Normalisierungen aller experimentellen Ergebnisse darstellt und Abbildung 4.12(b) die Normalisierungen nach Verbindungsart (Einfach- und Mehrfachverbindung) unterscheidet.

In Abbildung 4.12(a) erkennt man, dass sich nur ein kleiner Teil der 18 Testserien im unteren, linken Sektor nahe am Koordinatenursprung befinden. Der Großteil befindet sich nahe der Mitte. Insgesamt zeigt sich, dass mit steigender Performance auch die Energieeffizienz steigt. Allerdings zeigt Abbildung 4.12(b), dass dabei die Testserien, die eine Einzelverbindung nutzten, eine bessere Energieeffizienz mit steigender Performance hatten.

[4] Es sei zu Abbildung 4.11 bemerkt, dass auf eine Annotation der numerische Werte aufgrund der Darstellbarkeit verzichtet wurde. Die numerischen Werte sind im Anhang A.1.2 verzeichnet.

Beide Teildiagramme von Abbildung 4.12 zeigen aber auch, dass eine Testserie existiert, die sowohl am performantesen und als auch gleichzeitig am energieeffizientesten ist [THS10]. Dieselbe Aussage konnte auch für Experiment B getroffen werden. Damit gilt auch für Experiment C, dass die performanteste Testreihe für den Zugriff eine Datenbank mittels *SSB* auch gleichzeitig die energieeffizienteste ist.



(a) für alle experimentellen Ergebnisse

(b) nach Verbindungsart gruppiert

Abbildung 4.12: Energieeffizienz versus Performance für *SSB*

4.1.4 Ergebnisse des W22-Benchmarks

Das letzte Experiment, dass horizontale *scale-up*-Szenarien näher untersuchen soll, nutzt den *W22*-Benchmark. Im Gegensatz zu dem *TPC-H*- und dem *StarSchema*-Benchmark, die in den vorherigen Experimenten B und C verwendet wurden, ist bei *W22* nicht das Ziel, die Performance anhand von *OLAP*-Datenbankabfragen zu untersuchen, sondern die Auswirkungen der verschiedenen Operationen des *SQL*-Standards. Die Datenbankabfragen, die innerhalb von W22 definiert wurden, sind durch die Arbeiten von *Gontermann et al.* sowie *Tsirogiannis* ([GHR11, THS10]) inspiriert worden. Beide Studien befassten sich mit den Auswirkungen spezifischer *SQL*-Operatoren auf die Performance und zeigten, dass einzelne *SQL*-Operatoren sowie Kombinationen aus diesen durchaus einen beträchtlichen, messbaren Effekt haben können.

Experiment D: Skalierungsverhalten von *Postgresql* anhand des W22-Benchmarks
In diesem Experiment sollen die Auswirkungen auf die Performance, dem Energieverbrauch und die Energieeffizienz der einzelnen SQL-Operationen evaluiert werden. Als Performancemetrik wird die Antwortzeit von insgesamt 22 SQL-Datenbankabfragen verwendet, die in sechs Gruppen kategorisiert werden können:

1. Aggregatfunktionen (`count`, `avg` und `sum`)
2. Gruppierungsfunktionen (`GROUP BY`)
3. Sortierfunktionen (`ORDER BY`)
4. Nutzung verschiedener Basisdatentypen, zum Beispiel `int`, `varchar`, `text` und `date`

5. Duplikatserkennung (`DISTINCT`)
6. Vereinigungen (bedingt oder unbedingt, `JOIN`)

Wie bereits beschrieben, sind die Datenbankabfragen des *TPC-H*- und *StarSchema*-Benchmarks, die in den beiden vorhergehenden Textabschnitt erläutert wurden, darauf ausgelegt, ein *OLAP*-Szenario zu evaluieren. Dafür benutzen die Datenbankabfragen eine Mischung aus *SQL*-Operationen der oben genannten sechs Gruppen. Im Gegensatz dazu ist der *W22*-Benchmark speziell entworfen worden, die Auswirkungen von *SQL*-Anweisungen jeder Gruppe in sich zu evaluieren. Zudem erlaubt dieser Benchmark, weitergehende Betrachtungen hinsichtlich des Ausführungsplaners und -optimierers (*query planner*, *query optimizer*) von *Postgresql* durchzuführen sowie die Interaktionen mit dem darunterliegenden Betriebssystem zu analysieren.

Die 22 Datenbankabfragen von *W22* nutzen dieselben Tabellenstrukturen und Datensätze, wie sie auch auch der Datengenerator des *TPC-H*-Benchmarks generiert. Zur Ausführung des W22-Benchmarks wurden dieselben Datenmengen und Arbeitsspeicheranteile sowie Verbindungsarten genutzt, wie sie für die Messergebnisse von Experiment B und C definiert wurden. Das bedeutet, es wurden Datenbanken mit einer Größe von 1, 5 und 10 GByte bei jeweils einem Arbeitsspeicheranteil für *Postgresql* von 1, 2.5 und 5 GByte für beiden Verbindungsarten getestet. Somit entstanden dieselben 18 Testreihen, wie sich für die beiden letztgenannten Experimenten definiert wurden. Zudem wurde dieselbe technische Plattform für die Durchführung von Experiment D genutzt, wie sie in Tabelle 4.1 auf Seite 74 beschrieben ist.

Aggregatfunktionen Die Ergebnisse aller 18 Testserien von Experiment D in Hinblick auf die Performance der getesteten Aggregatfunktionen zeigen denselben Puffereffekt, der bereits bei den vorherigen Experimenten B und C beobachtet werden konnte. Dieser Effekt beschreibt, dass mit steigendem exklusiven Arbeitsspeicheranteil von *Postgresql* dem Betriebssystem weniger Arbeitsspeicher verbleibt, um Zugriffe auf den Massenspeicher zu puffern, auf dem die Datenbank- und Indexdateien von Experiment D gespeichert wurden. Dies führte zu heftigen Auslagerungsoperationen des Betriebssystems mit der Folge, dass die Ausführung des Benchmarks ausgesetzt wurde.

Eine Analyse der Ausführungspläne ergab, dass Datenbankanfragen an *Postgresql*, die speziell die Aggregatfunktionen `avg()`, `sum()` und `count()` beinhalten, *immer* das sequentiellen Lesen (*row scan*) der Datensätze der beteiligten Tabellen zur Folge hatten[5]. Das sequentielle Lesen von Inhalten der Datenbankdateien ist stark vom Datendurchsatz zum Massenspeicher abhängig, da sie dort gespeichert sind. Dementsprechend wirkt sich der eingangs genannte Puffereffekt aus. So konnte beobachtet werden, dass die Performance steigt, je weniger Arbeitsspeicher *Postgresql* für seine internen Puffer zur Verfügung stand und dadurch das Betriebssystem den Zugriff auf die Datenbankdateien besser puffern konnte.

Allerdings zeigt die Analyse der Ausführungspläne, dass sich der Puffereffekt nicht so stark auswirkt. So konnte beobachtet werden, dass die Antwortzeit für die erste Datenbankabfrage, die die Aggregatfunktionen `count(*)` testet, wesentlich größer war als die nachfolgenden Datenbankabfragen, die auf denselben Tabellen die Aggregatfunktionen `avg()` und `sum()` testeten. Dabei zeigte sich, dass die Tabelleninhalte durch die erste Datenbankabfrage durch die internen Puffer von *Postgresql* vorgehalten wurden. Das beschleunigte die Ausführung der nachfolgenden

[5] Dieses Verhalten gilt in der Theorie auch für jedes andere Datenbanksystem. Es gibt aber in der Praxis auch Ausnahmen. Bei Verwendung des *MyISAM*-Tabellenlayouts von *MySQL* etwa wird ein separates Feld vorgehalten und aktualisiert, das die Anzahl der Datensätze speichert. Damit werden zum Beispiel Datenbankabfragen, die die Aggregatfunktion `count(*)` nutzen, massiv beschleunigt, da keine *row scans* notwendig sind.

beiden Datenbankabfragen. Es konnte auch beobachtet werden, dass durch diese Pufferung die Art der Verbindung zu *Postgresql* unerheblich geworden ist.

Durch die Nutzung der internen Puffer für den Zugriff auf die Datenbankdateien von *Postgresql* zeigte sich auch, dass der Energieverbrauch messbar um circa 10 Prozent sank. Dies wirkt insgesamt auf die Energieeffizienz der getesteten Aggregatfunktionen, wie in Abbildung 4.13 dargestellt. In dieser Abbildung ist erkennbar, dass die höhere Performance durch die Pufferung auch zu einer höheren Energieeffizienz führt.

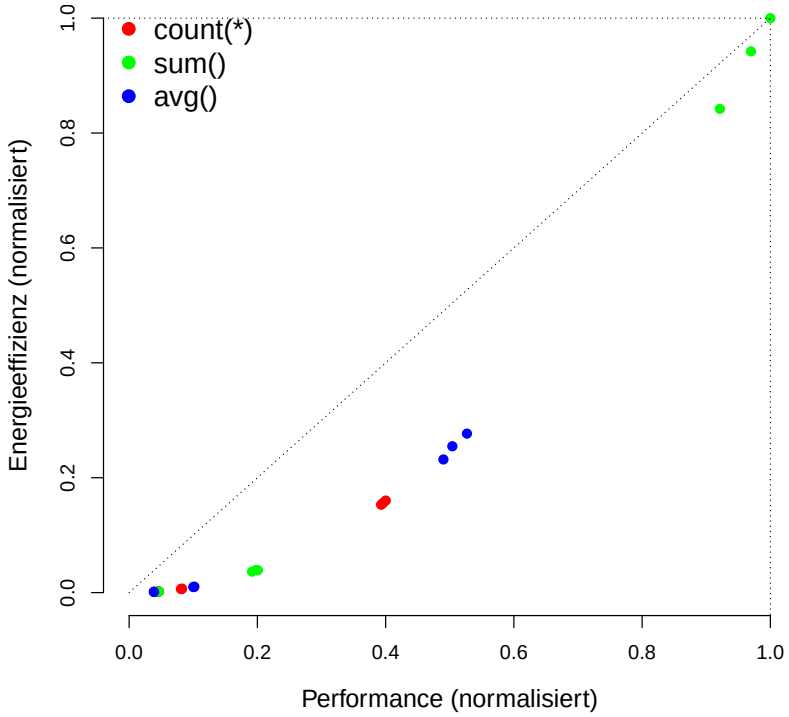


Abbildung 4.13: Performance versus Energieeffizienz für Gruppe 1 des *W22*-Benchmarks

Zusätzlich wurde experimentell geprüft, ob und welche Auswirkungen es hat, die Anzahl der Datensätze einer Tabelle unbestimmt oder durch die Angabe einer indizierten Spalte bestimmen zu lassen. Das bedeutet praktisch, dass anstatt der Aggregatfunktion `count(*)` ein `count(<spalte>)` geprüft wurde. Die Auswertung der experimentellen Ergebnisse zeigten keine Unterschiede hinsichtlich der Antwortzeit oder dem Energieverbrauch der gemessenen Kernkomponenten.

Gruppierungs- und Sortierfunktionen Die Datenbankabfragen der zweiten und dritten Gruppe des *W22*-Benchmarks (Gruppierungs- und Sortierfunktionen) nutzen als Basis die Datenbankabfragen der ersten Gruppe. Das sind diejenigen, die bereits im vorhergehenden Textabschnitt beschrieben wurden, also die Aggregatfunktionen `count()`, `avg()` und `sum()`. Sie wurden aber so modifiziert, dass das Ergebnis der Datenbankabfrage durch `GROUP BY`-Operatoren gruppiert beziehungsweise durch `ORDER BY`-Operatoren sortiert wurden.

Durch den Fakt, dass die Datenbankabfragen der zweiten und dritten Gruppe auf denen der ersten basieren, können die experimentellen Ergebnisse im Bezug auf die Performance, dem Energieverbrauch und der Energieeffizienz direkt verglichen werden. Der Vergleich der experimentellen Ergebnisse zeigt aber nur sehr kleine Unterschiede, die als Messungenauigkeit betrachtet werden können. Dementsprechend kann konstatiert werden, dass die getesteten Gruppierungs- und Sortieroperatoren keine Auswirkungen auf Performance, Energieverbrauch und Energieeffizienz gezeigt haben.

Nutzung verschiedener Basisdatentypen Die Datenbankabfragen, die für die vierten Gruppe des *W22*-Benchmarks definiert wurden, verfolgen zwei Ziele. Sie sollen die Auswirkungen

- der Nutzung spezieller Basisdatentypen wie etwa `INTEGER`, `VARCHAR` sowie `DATE` und
- die Operatoren dieser Basistypen, zum Beispiel die Vergleichsoperatoren für den numerischen Basisdatentyp `INTEGER`

hinsichtlich der Performance, des Energieverbrauches der technischen Kernkomponenten und der Energieeffizienz evaluieren.

Die experimentellen Ergebnisse für die Datenbankabfragen der vierten Gruppe zeigen, dass die Verwendung der verschiedenen Basisdatentypen keinen messbaren Einfluss auf die drei Aspekte Performance, Energieverbrauch und -effizienz hat. Das bedeutet zum Beispiel, dass unerheblich ist, anstatt `CHAR` den Datentyp `VARCHAR[1]` zu nutzen, um die Performance zu steigern.

Dementgegen offenbarten die experimentellen Ergebnisse zur Evaluierung der Operatoren der Basistypen einen deutlichen Einfluss auf die Performance. Dazu sei angemerkt, dass alle Datentypen innerhalb von *Postgresql* als abstrakter Datentyp definiert und implementiert sind. Das heisst, für einen Datentypen wird nicht nur sein Name, sondern auch der Typ, der Wertebereich und die möglichen Operatoren explizit angegeben und als Softwarebibliothek implementiert. Diese Bibliothek wird zur Laufzeit von *Postgresql* geladen und zum Beispiel dafür genutzt, um Werte einer Tabellenspalte mit dem entsprechenden Datentypen auf Einhaltung des Wertebereiches zu prüfen. Zudem ist es möglich, zusätzliche Operatoren zu einem späteren Zeitpunkt zu definieren und zu implementieren. Anhand der Informationen zum Datentypen kann zum Beispiel der Abfrageplaner (*query planner*) eingehende Datenbankabfragen dahingehend gestalten, dass unter Umständen ein Index genutzt wird.

Dieser Sachverhalt kann anhand des Basisdatentypen `INTEGER` erklärt werden, der innerhalb *Postgresql* implementiert und fest integriert ist. Dieser Basisdatentyp definiert einen numerischen Datentypen für vorzeichenbehaftete ganze Zahlen, dessen Wertebereich zwischen -2^{32} und $2^{32} - 1$ ist. Zudem werden drei Operatoren definiert und implementiert, um zwei Werte dieses Basisdatentypen vergleichen zu können: die größer-gleich-, kleiner-gleich- und ist-gleich-Operation (<,>,=). Die Nutzung dieser Operatoren kann innerhalb einer Datenbankabfrage an *Postgresql* wirksam durch einen Index, zum Beispiel durch einen B-Baum-Index, unterstützt werden. Es existieren auch noch andere Indexarten, zum Beispiel ein invertierter Index, die Operatoren für andere Basisdatentypen von *Postgresql* unterstützt. Es sei sich nun eine Tabelle gedacht, wobei die Werte der ersten Spalte alle vom Typ `INTEGER` sein sollen. Ferner sei nun eine Datenbankabfrage gedacht, die alle Werte dieser Spalte abfragt, die kleiner als eine bestimmte Zahl sein sollen. Sofern kein Index diese Abfrage unterstützt, bleibt keine andere Option als die gesamte Tabellenspalte sequentiell zu durchsuchen. Das bedeutet, dass die Datenbankdatei der Tabelle sequentiell gelesen werden muss. Die Existenz eines Index für diese Art der Datenbankabfrage, zum Beispiel durch einen B-Baum, würde sie erheblich beschleunigen.

Die experimentellen Ergebnisse zeigen sehr deutlich, dass die Performance immens gesteigert werden kann, wenn ein Index für die Datenbankabfrage existiert und genutzt wird. So zeigte sich für die getesteten Datenbankgrößen von 1, 5 und 10 GByte ohne Index eine reziproke Proportionalität: verfünffacht man zum Beispiel die Datenbankgröße für dieselbe Datenbankabfrage, so sank die Performance auf ein Fünftel des ursprünglichen Performancewertes. Selbiges gilt auch bei der Verzehnfachung der Datenbankgröße, wobei eine Performance von einem Zehntel des originalen Performancewertes gemessen wurde. Bei Existenz eines Index war zu beobachten, dass die Performance mit steigender Datenbankgröße zwar auch sank, aber nahe am Optimum des eingesetzten Index lag (logarithmische Performance in Relation zur Datenbankgröße, B-Baum-Indexstruktur).

Hinsichtlich des Energieverbrauches zeigten die experimentellen Ergebnisse, dass entgegen der Annahme die Existenz eines Index nicht zu einer Verringerung des Energieverbrauches der technische Komponenten führt. Im Gegenteil, die Existenz und die Nutzung eines Indexes bei einer Datenbankabfrage zeigte keine messbaren Unterschiede gegenüber denjenigen Datenbankabfragen, wo kein Index zur Verfügung stand. Wie aber beschrieben, hat die Existenz und Nutzung eines Indexes entscheidenden Einfluss auf die Performance. Anhand von Gleichung 3.3 für die Energieeffizienz zeigt sich also, dass Datenbankabfragen im Rahmen des *W22*-Benchmarks bei Nutzung eines Index eine signifikant höhere Energieeffizienz haben als diejenigen ohne Nutzung eines Index.

Vereinigungen und Duplikatserkennung Die beiden letzten Gruppen von Datenbankabfragen des *W22*-Benchmark haben den Zweck, die Performance, den Energieverbrauch der technischen Kernkomponenten und die Energieeffizienz bei der Duplikatserkennung sowie der Vereinigungen von Tabelleninhalten zu evaluieren. Da *Postgresql* ein Datenbanksystem ist, dass das relationale Datenmodell implementiert, ist die Vereinigung von Tabelleninhalten prinzipiell die Vereinigung von Relationen, wobei die Vereinigung bedingt oder unbedingt sein kann [Vos94, S.234 ff.]. Diese Vereinigung basiert auf der relationalen Algebra, die die Basis für das relationale Datenmodell darstellt [KE11, S.87]. Neben der relationalen Algebra und dem Relationkalkül ist *SQL* eine Sprache, die stark an erstgenannten angelehnt ist, aber ein weitaus praktischere Bedeutung im Umgang mit relationalen Datenbanken hat [Vos94, S.221]. In *SQL* werden Vereinigungen von Tabelleninhalten durch das Schlüsselwort `JOIN` ausgedrückt.

Innerhalb der letzten Gruppe des *W22*-Benchmarks wurden vier Datenbankabfragen definiert, die das Verhalten hinsichtlich der drei Aspekte Performance, Energieverbrauch und -effizienz untersuchen soll. Die erste Datenbankabfrage wurde in der Art definiert, dass ein unbedingtes `JOIN` untersucht werden soll. Die zweite und dritte Datenbankabfrage behandeln bedingte `JOIN`s, wobei zwischen inneren[6] und einem Equi-`JOIN`[7] unterschieden wurde. Die vierte und letzte Datenbankabfrage untersucht die Duplikatserkennung.

Die erste Datenbankabfrage der Gruppe war so gestaltet, dass die beiden größten *TPC-H*-Tabellen (Tabellen `lineitem` und `orders`, siehe Tabelle 4.2) vereinigt wurden, wobei diese Vereinigung dann durch zwei nachgelagerte Bedingungen eingeschränkt wurde (eine Bedingung pro Tabelle in der `WHERE`-Klausel). Die Annahme, dass die Ausführung dieser Datenbankabfrage durch die Vereinigung unperformant sein wird, hat sich nicht bestätigt. Die Analyse des Ausführungsplanes zeigte den (unbeabsichtigten) Einsatz eines Index für eine Tabelle und das sequentielle Lesen der

[6] Der *SQL*-Standard definiert einen *inner join* für zwei Tabelle `a` und `b` als `<table a> [INNER] JOIN <table b> WHERE <a.xyz> = <b.xyz>`.

[7] Ein *equi-join* hat die Form `<table a> JOIN <table b> ON <a.xyz> = <b.xyz>`. Der *SQL*-Standard erlaubt auch, die Bedingung durch die Nutzung des Schlüsselwortes `USING` zu verkürzen.

Einträge der anderen Tabelle. Hinsichtlich der Performance, dem Energieverbrauch und -effizienz zeigten die Messergebnisse kaum Unterschiede zu den Ergebnissen, die im letzten Textabschnitt zur Evaluierung der Basisdatentypen beschrieben wurden, obwohl das sequentielle Lesen einer Tabelle involviert war.

Die Auswertung der Messergebnisse der beiden nachfolgenden Datenbankabfragen hatte das Ergebnis, dass es hinsichtlich der Performance, dem Energieverbrauch und der Energieeffizienz keine Unterschiede gab. Daher wurden die Ausführungspläne beider Datenbankabfragen analysiert. Die Analyse ergab, dass beide Ausführungspläne identisch waren. Mit anderen Worten ausgedrückt: für *Postgresql* ist es unerheblich, wo die Bedingung für ein bedingten `JOIN` in der Datenbankabfrage platziert wird. Damit wird seitens *Postgresql* nicht zwischem einem expliziten, inneren und Equi-`JOIN` unterschieden.

	← *Datenbankgröße* →		
Tabelle	1 Gbyte	5 GByte	10 GByte
`lineitem`	6.001.215	299.999.795	599.986.052
`orders`	1.500.000	7.500.000	15.000.000
`part`	200.000	1.000.000	2.000.000
`customer`	150.000	750.000	1.500.000

Tabelle 4.2: Anzahl der Datensätze von ausgewählten *TPC-H*-Tabellen für verschiedene Datenbankgrößen

Um die Performance zu evaluieren, mit der `JOIN`-Operationen den Inhalt zweier Tabellen vereinigen, wurden dieselben drei genutzten Datenbankabfragen so modifiziert, dass sie die Anzahl der vereinigten Datensätze zurückgaben. Dafür wurde die `COUNT(*)`-Aggregatfunktionen genutzt und zusätzliche Testreihen durchgeführt. Wie in Textabschnitt zur Evaluierung der Aggregatfunktionen beschrieben, führt diese Aggregatfunktion immer dazu, dass *Postgresql* keinerlei Indizes nutzt, sondern den Inhalt der beteiligten Tabellen sequentiell ausliest. Dies gilt sowohl für unbedingte als bedingte `JOIN`s. Die Messergebnisse zeigen sehr deutlich, dass die Performance sehr stark vom Datendurchsatz zu den Datenbankdateien auf dem Massenspeicher abhängen. Insofern ähneln die Messergebnisse denjenigen aus Experiment B und C, in denen der *TPC-H*- und *SSB*-Benchmark verwendet wurden. Das bedeutet, dass sich die Messergebnisse hinsichtlich den Energieverbrauches nicht stark unterscheiden, aber die Performance- und Energieeffizienzwerte stark variieren. Es ist nicht überraschend, dass die Performance bei der Nutzung eines `JOIN`s von der Anzahl der Datensätze abhängt: je weniger Datensätze vorhanden sind, desto höher war die Performance und desto höher war auch die Energieeffizienz.

Die letzte Datenbankabfrage dieser Gruppe hatte das Ziel, die Duplikationserkennung zu untersuchen. Die Datenbankabfrage fragt die Werte einer Spalte der größten *TPC-H*-Tabelle `lineitem` (vgl. Tabelle 4.2) ab, wobei diese Spalte keinen unterstützenden Index hatte. Diese Spalte wurde gewählt, da sichergestellt war, dass sie Duplikate enthielt. Zur Erkennung und Eliminierung von Duplikaten wurde die `DISTINCT()`-Funktion genutzt. Die Auswertung der experimentellen Ergebnisse zeigte dasselbe Verhalten hinsichtlich der Performance, Energieverbrauch und -effizienz wie es bereits im Textabschnitt zu den Aggregatfunktion beschrieben ist. Daher wurden die Ausführungspläne analysiert, um den Grund zu erfahren.

Es zeigte sich, das die `DISTINCT()`-Funktion das sequentielle Lesen der Datenbankdatei für die Tabelle `lineitem` ausgelöst hat, wobei zur Duplikatserkennung die gelesenen Werte gehasht und anhand des Hashwertes verglichen wurden.

Um auszuschließen, dass die Nutzung der `DISTINCT()`-Funktion *immer* zum sequentiellen Lesen einer Datenbankdatei führt, wurden zusätzliche Testserien durchgeführt. Dabei wurde für die Spalte, die durch die genutzte Datenbank abgefragt wurde, ein Index erstellt. Der Vergleich der zusätzlichen Messergebnisse mit den originalen zeigt aber, dass sich die Performancewerte nur in Nuancen unterschieden. Die Analyse der Ausführungspläne zeigte auch bei Existenz des Index, dass die Datenbankdatei für die Tabelle `lineitem` sequentiell gelesen wurde. Damit kann festgestellt werden, dass die Nutzung der `DISTINCT()`-Funktion zusammen mit Aggregatfunktionen wie `count()`, `avg()` und `sum()` *immer* zu einem sequentiellen Lesevorgang der beteiligten Datenbankdateien von *Postgresql* führen.

4.2 Untersuchung horizontaler *scale-up*-Szenarien

Wie im vorhergehenden Textabschnitt 4.1 beschrieben, betrachten vertikale *scale-up*-Szenarien die Auswirkungen auf die Performance eines Datenbanksystems, das zentral betrieben wird (Einzelplatz-PC beziehungsweise -Server). Dabei wird ein potentieller Einflussparameter variiert, während die übrigen konstant verbleiben. Im allgemeinen wird die Datenmenge, die zu es verarbeiten gilt, nach oben skaliert, um die Auswirkungen zu bemessen. Es kann aber auch die technische Plattform, auf die das Datenbanksystem ausgeführt wird, variiert werden, um deren Auswirkungen zu evaluieren, zum Beispiel der Austausch von magnetischen Festplatten zu SSDs.

Horizontale *scale-up*-Szenarien befassen sich dagegen mit den Auswirkungen auf die Performance eines Datenbanksystems, dass dezentral betrieben wird. Das bedeutet, dass der Gegenstand der Untersuchung verteilte Datenbanksysteme sind, die auf einem Cluster ausgeführt werden (vgl. hierzu Textabschnitt 2.4). Hierbei ist vom größeren Interesse, wie sich ein verteiltes Datenbanksystem gegenüber der Änderung des zugrunde liegenden Clusters verhält.

Es stellt sich natürlich die Frage, warum überhaupt horizontale *scale-up*-Szenarien betrachtet werden. Der Grund ist der, dass zentral betriebene Datenbanksysteme mittlerweile mit sehr großen Datenmengen konfrontiert sind, die es zu speichern und zu verarbeiten gilt. Stößt ein solches Datenbanksystem an seine Leistungsgrenzen, zum Beispiel im Bezug zur Speicherkapazität oder Antwortzeiten, können nur vertikale Optimierungen vorgenommen werden. Das bedeutet zumeist den Einsatz eines anderen Datenbanksystems, das eine höhere Performance verspricht oder den Austausch von technischen Komponenten, die performanter arbeiten oder mehr Speicherplatz bieten. Wenn selbst diese Maßnahmen nicht mehr ausreichen, ist es ein probates und komfortables Verfahren, von einem zentralen zu einem verteilten Datenbanksystem zu migrieren. Diese bieten gewichtige Vorteile wie etwa Lastverteilung und eine höhere Ausfallsicherheit. Dieses Verfahren ist bewährt und zeigt sich auch in anderen Bereichen. Zum Beispiel wurden zur Leistungssteigerung Hauptplatinen eingeführt, die anstatt einem mehrere Prozessoren unterstützen. Zudem sind mittlerweile Einkern-Prozessoren von Mehrkernprozessoren verdrängt worden, die dieselbe Zielstellung verfolgen.

Wird also von einem zentralen zu einem verteilten Datenbanksystem gewechselt, das in einem Cluster betrieben wird, so können auch andere Vorteile genutzt werden, zum Beispiel eine Partitionierung der zu speichernden Daten gemäß den technischen Spezifikationen der jeweiligen Clusterknoten. Wenn zum Beispiel in einem Cluster mehrere hochperformante mit weniger performanten Clusterknoten kombiniert werden, so können diejenigen Daten primär auf ersteren

gespeichert und verarbeitet werden, die am meisten abgefragt oder modifiziert werden. Es sind aber auch andere Partitionierungen möglich, zum Beispiel die redundante Speicherung auf mehreren Clusterknoten, um die Ausfallsicherheit zu gewährleisten.

Diese genannten Vorteile haben dazu geführt, dass in der Zwischenzeit verteilte Datenbanksysteme in großen und sehr großen Datenverarbeitungszentren vorherrschen. Allerdings müssen auch die Nachteile genannt werden. Ein verteiltes Datenbanksystem beziehungsweise der zugrunde liegende Cluster kann nicht beliebig vergrößert werden, wenn es an seine Leistungsgrenzen stößt. Die Gründe dafür sind, dass jeder zusätzliche Clusterknoten Platz, Strom, Kühlung und Wartung benötigt, wobei meist bauliche Grenzen gesetzt sind.

Allerdings muss auch davon ausgegangen werden, dass ein möglicher Performancegewinn nicht notwendigerweise mit den eingesetzten Resourcen skaliert. Wenn zum Beispiel die Anzahl der Clusterknoten, auf denen ein verteiltes Datenbanksystem ausgeführt wird, verdoppelt wird, so sollte sich auch die Performance verdoppeln. Das äußert sich zum Beispiel darin, dass sich durch die Lastverteilung die Antwortzeiten im besten Fall halbieren. Dieser Zusammenhang wird formal als *speedup* definiert, also der zu erwartende Gewinn in Abhängigkeit zu den eingesetzten Resourcen. Zur Erläuterung kann sich ein Cluster gedacht werden, auf dem ein verteiltes Datenbanksystem ausgeführt wird. Dieser Cluster besitzt zuerst n Clusterknoten, wobei dann die Größe auf m Clusterknoten erhöht wird. Formal können diese Größen als nicht-leere Teilmenge Z_c der positiven ganzen Zahlen $\mathbb{Z}$ definiert werden:

$$Z_c = \{z \in \mathbb{Z} | \exists\, n, m \in \mathbb{Z} : 1 \leq z \leq n < m\} \tag{4.1}$$

Sei nun eine Metrik angenommen, die für den Cluster evaluiert werden kann, zum Beispiel die Antwortzeit für die Performance oder dem Energieverbrauch. Die Messwerte für diese Metrik werden durch die Funktion *metrik* den Clustergrößen zugewiesen:

$$\mathit{metrik} : Z_c \to \mathbb{R}$$

Nun können zwei Clustergrößen $z_1 \in Z_c$ und $z_2 \in Z_c$ betrachtet werden, wobei $z_1 < z_2$ ist. Der *speedup* s ist demnach:

$$s(z_1, z_2 \in Z_c | z_1 < z_2) = \frac{\mathit{metrik}(z_2)}{\frac{z_2}{z_1} \cdot \mathit{metrik}(z_1)} = \frac{z_1 \cdot \mathit{metrik}(z_2)}{z_2 \cdot \mathit{metrik}(z_1)} \tag{4.2}$$

In Gleichung 4.2 gibt z_2/z_1 den Anstiegsfaktor der Clustergröße an. Ist zum Beispiel $z_1 = 3$ und $z_2 = 6$, dann ist dieser Faktor 2. Er wird mit dem Messwert der Metrik der Clustergröße z_1 multipliziert (*metrik*(z_1)). Dadurch erhält man einen Erwartungsmesswert der Metrik für die Clustergröße z_2. Dividiert man den eigentlichen Messwert der Metrik für die Clustergröße z_2 (*metrik*(z_2)) durch den Erwartungswert, so erhält man den eigentlichen *speedup*. Für obiges Beispiel mit $z_1 = 3$ und $z_2 = 6$ sei *metrik*$(z_1) = 1$ und *metrik*$(z_1) = 1.5$. Damit ergibt sich nach Gleichung 4.2 ein *speedup* $s(z_1, z_2) = 0.7$. Das heisst, wenn die Clustergröße verdoppelt wurde, ergab sich nur ein Zuwachs von 50 Prozent der gemessenen Metrik.

Das wesentlich Interessantere an Gleichung 4.2 ist die Betrachtung des Wertebereiches. Je nach errechnetem *speedup* s für zwei $z_1, z_2 \in Z_c$ wird der Wert als *sublinear*, *linear* oder *superlinear* bezeichnet:

$$s(z_1, z_2 \in Z_c | z_1 < z_2) = \frac{z_1 \cdot \mathit{metrik}(z_2)}{z_2 \cdot \mathit{metrik}(z_1)} \begin{cases} > 1 & \textit{superlinear} \\ = 1 & \textit{linear} \\ < 1 & \textit{sublinear} \end{cases}$$

Diese Bezeichnungen werden oft in Verbindung mit *Amdahl's Gesetz* genannt. In seinen Untersuchungen zum Leistungsgewinn von parallelisierten gegenüber seriell ausgeführten Anwendungsinstruktionen von 1967 kam *Amdahl* in [Amd67] zu der Erkenntnis, dass der maximale *speedup* bezüglich der Performance lediglich linear sein kann und in Realität eher sublinear ist. Bezogen auf das oben gezeigte Beispiel bezeichnete *metrik* die experimentellen Ergebnisse für die Performance. Bei $z_1 = 3$, $z_2 = 6$ und *metrik*$(z_1) = 1$ kann nach *Amdahl metrik*$(z_2) \leq 2$ sein, also maximal 2 und in Realität eher kleiner. Das bedeutet, dass sich im besten Fall die Performance verdoppelt, wenn die Clustergröße verdoppelt wird.

Auch im Falle von verteilten Datenbanksystemen können dieselben Benchmarks genutzt werden wie sie auch zur Evaluierung der Performance von zentralen Datenbanksystem zum Einsatz kommen. Allerdings wird in diesen Fall nicht nur das Skalierungsverhalten für eine hochskalierte Datenmenge untersucht, sondern auch die Charakteristiken des eingesetzten Clusters werden mit einbezogen. Generell wird hier als Einflussgröße die Anzahl der beteiligten Clusterknoten genutzt.

Zur Untersuchung des Skalierungsverhaltens von verteilten Datenbanksystemen wurden im Rahmen dieser Dissertation zwei Experimente durchgeführt. In beiden Experimenten wurde das verteilte Datenbanksystem *Cassandra* genutzt, wobei zwei typische Anwendungsfälle betrachtet wurden, mit denen ein Datenbanksystemadministrator in der Realität konfrontiert wird. In den nächsten zwei Textabschnitten werden beide Experimente im Bezug auf Zielstellung, Messaufbau und -durchführung sowie die Ergebnisse detailliert beschrieben.

4.2.1 Experimentelle Ergebnisse für einen 7-Knoten-*Cassandra*-Cluster

Das erste Experiment nutzt einen *Cassandra*-Cluster, der aus sieben Clusterknoten besteht. Wie bereits im vorherigen Textabschnitt beschrieben, werden zwei täglich anzutreffende Anwendungsfälle evaluiert.

Experiment E: Cassandra-Cluster mit 7 Clusterknoten und *YCSB* als Benchmark
Dieses Experiment untersucht zwei Anwendungsfälle, mit denen ein Administrator von einem verteilten Datenbanksystem konfrontiert wird. Zur besseren Unterscheidung werden die beiden Anwendungsfälle mit A und B bezeichnet:

- Anwendungsfall A sieht eine feste Datenmenge vor, die ein verteiltes Datenbanksystem und der zugrunde liegende Cluster speichern soll. Der Administrator möchte herausfinden, wieviele Clusterknoten notwendig sind, um eine optimale Performance zu erzielen.
- Anwendungsfall B sieht eine treppenartig ansteigende Datenmenge für ein verteiltes Datenbanksystem vor. Dies ist zum Beispiel der Fall, wenn einem bestehenden Cluster ein neuer Clusterknoten hinzugefügt wird, um beispielsweise die Speicherkapazität zu erhöhen oder um die Performance im Sinne der Lastverteilung zu steigern.

Das Experiment wird auf einem Cluster ausgeführt, der insgesamt aus acht Clusterknoten besteht. Als Benchmark wird der *Yahoo Cloud Serving Benchmark* (*YCSB*) genutzt, für den vier Lastfälle definiert wurden: `load`, `read`, `write` und `mixed`. Einer der acht Clusterknoten dient exklusiv zur Ausführung dieser Lastfälle, wobei auf einen *Cassandra*-Cluster zugriffen wird, der aus den übrigen Clusterknoten gebildet wird (minimal drei und maximal sieben Clusterknoten).

Wie aus der Beschreibung von Experiment E hervorgeht, wird als verteiltes Datenbanksystem *Cassandra* genutzt, dessen Performance mit dem *Yahoo Cloud Serving Benchmark* (*YCSB*, definiert in [CST+10]) für beiden Anwendungsfälle anhand von vier Lastfällen evaluiert wird. Die Gründe für die *Cassandra/YCSB*-Kombination liegen darin, dass dieses verteilte Datenbanksystem vielfach eingesetzt wird, dessen Architektur sehr gut dokumentiert ist und bereits viele Publikationen vorliegen, die dessen Performance mit *YCSB* evaluieren und analysieren (zum Beispiel [RGVS+12, PPR+11, KKR14, DSK+13]). Es sei an dieser Stelle noch auf die Studie von *Rosselli et al.* in [RNI+15] hingewiesen, die dieselbe Testmethodik und denselben Versuchsaufbau wie Experiment E genutzt hat, um experimentell das Verhalten von *Cassandra* im Bezug auf die Ausfallsicherheit zu untersuchen.

YCSB ist sehr flexibel im Bezug auf die Definition der Arbeitslast für ein verteiltes Datenbanksystem. Generell werden aber zwei Typen unterschieden:

1. Einfügen von Datensätzen: der *YCSB*-Client verbindet sich mit dem *Cassandra*-Cluster und fügt eine festgelegte Menge von Datensätzen ein. Die Größe eines Datensatzes ist einstellbar. Für Experiment E wurde eine Datensatzgröße von 1 kByte genutzt. Die eingefügten Datensätze werden durch interne Mechanismen von *Cassandra* gleichmäßig auf alle beteiligten Clusterknoten verteilt[8]. Diese Arbeitslast wird im folgenden als `load` referenziert.
2. Modifizieren und Abfragen von Datensätzen: der *YCSB*-Client greift auf die vormals eingefügten Datensätze zu. Dabei wird zwischen einem nur-lesenden, nur-modifizierenden Zugriff oder einer Kombination aus beiden unterschieden. Im folgenden werden diese drei Arbeitslasten als `read`, `write` und `mixed` bezeichnet.

Aus Vergleichsgründen wurden die vier Arbeitslasten `load`, `read`, `write` und `mixed` für beide Anwendungsfälle A und B von Experiment E ausgeführt. Die Ausführung einer *YCSB*-Arbeitslast resultiert immer mit der Angabe von *operations* (kurz *ops*) als Metrik. Das bedeutet zum Beispiel, dass 350 Millionen Datensätze in den *Cassandra*-Cluster eingefügt werden, wenn für die Arbeitslast `load` 350 Millionen *ops* konfiguriert werden. Im Falle der Arbeitslasten `read`, `write` und `mixed` bedeutet es, wieviele Datensätze gelesen oder modifiziert werden. In Tabelle 4.3 auf Seite 100 sind die Anzahlen an *ops* für alle Arbeitslasten dargestellt.

Der Ablauf einer Testreihe für Experiment E verlief immer nach folgenden Schritten:

1. Zuerst wurde die Arbeitslast `load` ausgeführt. Für Anwendungsfall A werden 350 Millionen *ops* ausgeführten, was ungefähr 350 GByte an Datensätzen entspricht. Für Anwendungsfall B war die Anzahl an auszuführenden *ops* ein Vielfaches der Anzahl an beteiligten *Cassandra*-Clusterknoten. Die Gesamtanzahl an *ops* entsprach immer der Anzahl an Clusterknoten multipliziert mit 50 Millionen. Die Gesamtanzahlen sind in Tabelle 4.3(b) in der Tabellenzeile “load” ablesbar.
2. Danach wurden die Arbeitslasten `read`, `write` und `mixed` ausgeführt, wobei eine diskrete Gleichverteilung verwendet wurde. Nach *Cooper et al.* [CST+10] werden die zu lesenden oder zu modifizierenden Datensätze per Zufall ausgewählt, wobei jeder Datensatz dieselbe Wahrscheinlichkeit dafür besitzt.
   Für beide Anwendungsfälle A und B ist die Anzahl an *ops* auf 10 Millionen fixiert worden. Das bedeutet, dass für die Arbeitslast `read` 10 Millionen Datensätze gelesen werden, die

[8] Die gleichmäßige Verteilung nutzt den Umstand, dass nach [Hew11] der Primärschlüsselraum von Datensätzen durch *Cassandra* durch die Anzahl der Clusterknoten geteilt wird. Jeder Clusterknoten ist damit für einen Teil des Primärschlüsselraumes zuständig, womit sich auch der Speicherort eines Datensatzes anhand seines Primärschlüssels ergibt.

vormals durch die Arbeitslast `load` eingefügt worden sind. Die Arbeislast `write` modifiziert dann 10 Millionen Datensätze. Die Arbeitslast `mixed` liest zuerst 5 Millionen Datensätze, bevor dann 5 Millionen Datensätze modifiziert werden.

3. Dieselben Arbeitslasten `read`, `write` und `mixed` werden nochmals ausgeführt, aber anstatt einer diskreten Gleichverteilung wird eine *Zipf*-Verteilung[9] genutzt. Sie ist darüber charakterisiert, dass die Wahrscheinlichkeit, dass ein Datensatz gelesen oder modifiziert wird, für manche größer als für die übrigen ist.

	← *Knoten im Cassandra-Cluster* →				
Arbeitslast	**3**	**4**	**5**	**6**	**7**
`load`	350	350	350	350	350
Diskrete Gleichverteilung					
`read`	10	10	10	10	10
`write`	10	10	10	10	10
`mixed`	5+5	5+5	5+5	5+5	5+5
Zipf-Verteilung					
`read`	10	10	10	10	10
`write`	10	10	10	10	10
`mixed`	5+5	5+5	5+5	5+5	5+5

(a) Anwendungsfall A (feste Datenmenge)

	← *Knoten im Cassandra-Cluster* →				
Arbeitslast	**3**	**4**	**5**	**6**	**7**
`load`	150	200	250	300	350
Diskrete Gleichverteilung					
`read`	10	10	10	10	10
`write`	10	10	10	10	10
`mixed`	5+5	5+5	5+5	5+5	5+5
Zipf-Verteilung					
`read`	10	10	10	10	10
`write`	10	10	10	10	10
`mixed`	5+5	5+5	5+5	5+5	5+5

(b) Anwendungsfall B (ansteigende Datenmenge)

Tabelle 4.3: Anzahl der verarbeiteten Datensätze in Millionen für Experiment E (1 Million Datensätze entspricht ungefähr einer Datenmenge von 1 GByte)

Jede Testreihe wurde dreimal wiederholt, um Nebeneffekte auszuschließen. Zwischen jeder Wiederholung wurde der Testcluster komplett neugestartet und die Datenbankstrukturen, die von *YCSB* automatisch angelegt werden, gelöscht. Somit wurden insgesamt 210 Einzelexperimente durchgeführt (2 Anwendungsfälle $\times$ 7 Arbeitslasten $\times$ 5 *Cassandra*-Clustergrößen $\times$ 3 Wiederholungen). Für jedes Einzelexperiment wurde die Antwortzeit in Sekunden gemessen.

Für die Testreihen wurde ein *Fujitsu BX600 S3* Bladecenter[10] genutzt. Die technischen Spezifikationen des genutzten Bladecenters im Bezug auf Hard- und Software sind in Tabelle 4.4 auf Seite 101 aufgelistet. Bladecenter-Knoten B1 wurde exklusiv dafür genutzt, den *YCSB*-Client auszuführen, der auf den *Cassandra*-Cluster zugriff. Die übrigen Bladecenter-Knoten B2 bis B8 wurden dazu genutzt, einen *Cassandra*-Cluster zu bilden, wobei die Clustergröße zwischen drei und sieben Clusterknoten betrug. Bei jeden Clusterknoten wurden die lokalen Festplatten (Massenspeicher) zu einem RAID-0 verbunden.

In Abbildung 4.14 auf Seite 101 ist der Testaufbau schematisch dargestellt. Die Knoten des Bladecenters waren mit dessen interner Backplane verbunden, die sie mit Strom und Netzwerkzugang mit einer Bandbreite von 1 GBit/s versorgte. Wie bereits erwähnt, wurden die Bladecenter-

[9] Die *Zipf*-Verteilung ist ein Spezialfall der *Pareto*-Verteilung und wird oft im Zusammenhang mit der Verteilung von Worten in einem Text genannt. Ein anschauliches Beispiel wurde von *Axtel* in [Axt01] veröffentlicht. Dabei untersuchte er die Anzahl an US-amerikanischen Firmen in Relation zu ihrer Mitarbeiterzahl. Er fand heraus, dass sie dabei einer *Zipf*-Verteilung unterliegen: verzehnfacht man die Anzahl der Mitarbeiter, dann sinkt die Anzahl der Firmen mit dieser Mitarbeiterzahl auf ein Zehntel.

[10] Ein Bladecenter ist eine Server-Baugruppe, die es erlaubt, mehrere unabhängige, nicht eigenständig funktionierende Server (Blades) äußerst kompakt zu organisieren. Das Bladecenter versorgt jedes Blades mit Strom, Netzwerkzugang und anderen Schnittstellen.

Knoten B2 bis B8 dafür genutzt, einen *Cassandra*-Cluster mit ansteigender Knotenanzahl zu bilden. In Abbildung 4.14 ist ein minimaler *Cassandra*-Cluster mit drei Clusterknoten dargestellt.

Komponente	**Clusterknoten B1**	**Clusterknoten B2 bis B8**
CPU	2×AMD Opteron 890	2×AMD Opteron 870
Hauptspeicher	32 GByte DDR-2 reg.	16 GByte DDR-2 reg.
Massenspeicher	2×300 GByte	2×146 GByte
Betriebssystem	*Ubuntu Server* 12.04 LTS x64	
Java Runtime	*Oracle* JRE 1.7.0.60-b19	
Cassandra	*DataStax Enterprise Server* 4.5.0	

Tabelle 4.4: Spezifikationen des *Cassandra*-Clusters für Experiment E

Zur Messung des Energieverbrauches wurde ein geeichtes Leistungsmessgerät vom Typ *EMH DMTZ-XC* genutzt. Im Gegensatz zu den Experimenten B, C und D war es bei dem vorliegenden Testaufbau nicht möglich, die elektrische Leistungsaufnahme der technischen Kernkomponenten (CPU, Arbeits- und Massenspeicher sowie Mainboard) eines jeden Clusterknotens zu messen. Der Grund dafür ist, dass die internen Stromschienen im Inneren der Clusterknoten proprietär sind und keinerlei Dokumentation existiert. Stattdessen wurde der elektrische Leistungsverbrauch des gesamten Testaufbaus gemessen. Genauer gesagt wurde der Energieverbrauch des gesamten genutzten Bladecenters gemessen. Das Leistungsmessgerät wurde alle 15 Sekunden durch einen externen Server nach der verbrauchten elektrischen Leistung abgefragt und zur späteren Analyse in eine Datenbank gespeichert. Eine kürzere zeitliche Abfrageperiode war leider technisch nicht möglich, da die Abfrageschnittstelle des Leistungsmessgerätes nur eine sehr schmale Datenübertragungsbandbreite besaß. Um zu gewährleisten, dass die Zeitstempel der abgefragten Messwerte übereinstimmen, wurden alle Komponenten des Testaufbaus (das Bladecenter, alle Clusterknoten, das Leistungsmessgerät sowie der externe Server zur Speicherung der Messwerte) periodisch zeitlich synchronisiert.

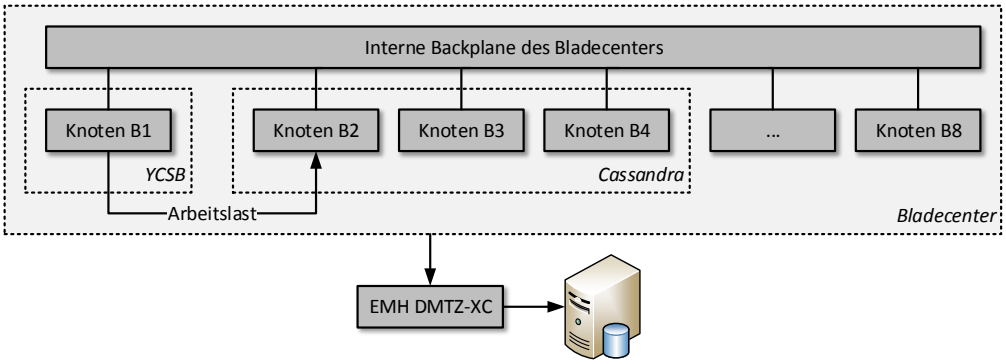


Abbildung 4.14: Schematischer Testaufbau für Experiment E auf einem Bladecenter mit 8 Knoten. Knoten B1 dient als *YCSB*-Client, der auf einen *Cassandra*-Cluster mit variabler Knotenanzahl zugreift (Knoten B2 bis B8). Dargestellt ist ein minimaler *Cassandra*-Cluster mit drei Clusterknoten.

Experimentelle Ergebnisse zu Anwendungsfall A Wie bereits in der Beschreibung zu Experiment E auf Seite 98 erläutert, sieht Anwendungsfall A die Verarbeitung einer festen Datenmenge von circa 350 GByte durch einen *Cassandra*-Cluster mit ansteigender Anzahl von Clusterknoten vor. Die experimentellen Ergebnisse hinsichtlich der Antwortzeit, dem Energieverbrauch und der errechneten Energieeffizienz für die Arbeitslast `load` ist exemplarisch in Abbildung 4.15 auf Seite 104 dargestellt.

In Abbildung 4.15 sind drei einzelne Diagramme dargestellt, die die durchschnittliche Antwortzeit, den durchschnittlichen Energieverbrauch und die errechnete Energieeffizienz für die getesteten Clustergrößen zeigen. Dabei ist zu beachten, dass sich die Performance anhand Gleichung 3.1 auf Seite 54 errechnet. Hierfür wird die Anzahl der Operationen durch die Antwortzeit dividiert. Die Anzahl an Operationen für jede Arbeitslast kann in Tabelle 4.3(a) auf Seite 100 eingesehen werden. Das bedeutet für die Arbeitslast `load` des Anwendungsfalles A von Experiment E, dass sich die Performance aus der Division von 350 Millionen *ops* und der in Abbildung 4.15 dargestellten Antwortzeiten ergibt. Die errechneten Performancewerte können dann genutzt werden, um die Energieeffizienz zu errechnen. Dafür wird Gleichung 3.3 von Seite 64 genutzt. Somit ergibt sich die in Abbildung 4.15 dargestellten Energieeffizienzwerte aus der Division der Performancewerte mit den Energieverbrauchswerten, wie sie numerisch in Abbildung 4.15 abgelesen werden können.

Die experimentellen Ergebnisse für die übrigen Arbeitslasten `read`, `write` und `mixed` des Anwendungsfalles A sind als Diagramme im Anhang A.1.3 ab Seite 228 dargestellt. Sie folgenden derselben Struktur wie Abbildung 4.15, dass heisst, es werden die Antwortzeiten, der Gesamtenergieverbrauch und die daraus resultierende Energieeffizienz als Teildiagramme dargestellt.

Hinsichtlich der Performance ist in Abbildung 4.15 erkennbar, dass mit steigender Anzahl an *Cassandra*-Clusterknoten die durchschnittlichen Antwortzeiten sinken. Insofern steigt damit auch die Performance mit jedem hinzugefügten *Cassandra*-Clusterknoten. Allerdings ist auch ein Trend erkennbar. Die Antwortzeiten sinken in demselben Maße, wie die Anzahl an *Cassandra*-Clusterknoten steigt. Es besteht also die Annahme, dass es eine *Cassandra*-Clustergröße geben kann, bei der kaum noch ein Performancegewinn möglich ist.

Im Bezug auf den Energieverbrauch ist in Abbildung 4.15 ersichtlich, dass auch der Energieverbrauch mit steigender Größe des *Cassandra*-Clusters sinkt. Dazu muss angemerkt werden, dass zum Zeitpunkt der Durchführung der Experimente die administrativen Funktionen des genutzten Bladecenters eingeschränkt waren. Somit war es leider nicht möglich, ungenutzte Clusterknoten abzuschalten oder in einen Energiesparmodus zu versetzen. Dementsprechend umfasste die Messung des Energieverbrauches immer alle Clusterknoten des Bladecenters. Das führte dazu, dass immer ein gewisser Grundverbrauch an Energie vorhanden war und gemessen wurde.

Eine nachgelagerte Betrachtung des Versuchsaufbaus von Experiment E zeigte folgenden Sachverhalt: das verwendete Bladecenter wurde mit ingesamt neun Knoten B1 bis B9 betrieben. Aufgrund des eingeschränkten administrativen Zuganges waren alle neun Knoten permanent eingeschaltet. Insbesondere Knoten B9 wurde im Leerlauf betrieben, obwohl er während der Experimente nicht genutzt wurde. Je nach getesteter *Cassandra*-Clustergröße wurden bis zu vier Knoten im Leerlauf betrieben. Allerdings wurden bei allen Einzelexperimenten die Auslastung der Bladecenter-Knoten gemessen und gespeichert. Insgesamt kann der gemessene Gesamtenergieverbrauch und die gemessene Auslastung genutzt werden, um den Energieverbrauch jedes Bladecenter-Knotens als Gleichung darzustellen:

$$W_c = \underbrace{W_0}_{\textit{const.}} + \underbrace{(l_1 \cdot W_1 + W_1)}_{\textit{Knoten 1}} + \underbrace{(l_2 \cdot W_2 + W_2)}_{\textit{Knoten 2}} + \cdots + \underbrace{(l_8 \cdot W_8 + W_8)}_{\textit{Knoten 8}} + \underbrace{W_9}_{\textit{const.}} \tag{4.3}$$

Hierbei bezeichnet W_c den gemessenen Gesamtenergieverbrauch und W_0 den Energieverbrauch des Bladecenters an sich ohne die Knoten. Der Energieverbrauch eines Knotens ergibt sich aus einem grundlegenden Energieverbrauch im Leerlauf W_1 bis W_9 plus einen anteiligen, auslastungsabhängigen zusätzlichen Energieverbrauch. Die Auslastung wird über die Faktoren l_1 bis l_8 angegeben, wobei der Wertebereich reell ist und zwischen Null (Leerlauf) und Eins (volle Auslastung) liegt. In Gleichung 4.3 ist zu beachten, dass für Knoten B9 kein Auslastungsfaktor l_9 angegeben wurde, da er permanent im Leerlauf betrieben wurde. Dementsprechend ergibt sich ein konstanter grundlegender Energieverbrauch W_9.

Das Ziel von Gleichung 4.3 ist es, den Wert des gemessenen Gesamtenergieverbrauch W_c in einer Weise zu korrigieren, dass nur der Energieverbrauch derjenigen Summanden in Gleichung 4.3 aufsummiert wird, die tatsächlich zum eigentlichen Gesamtenergieverbrauch beigetragen haben. Das bedeutet, dass nur der Energieverbrauch derjenigen Bladecenter-Knoten aufsummiert werden muss, die auch faktisch während einer Testserie genutzt und nicht im Leerlauf betrieben wurden. Diese Korrektur ist deswegen essentiell, da die Energieeffizienz gemäß Gleichung 3.3 auf Seite 64 darauf basiert, die Performance mit dem Energieverbrauch zu dividieren. Am Beispiel eines Einzelexperimentes für einen *Cassandra*-Cluster mit drei *Cassandra*-Clusterknoten sind in Gleichung 4.3 insgesamt fünf Summanden aufzuaddieren: W_0 sowie die vier Summanden für die Knoten B1 bis B4 (B1: *YCSB*-Client, B2 bis B4: *Cassandra*-Cluster). Die restlichen Summanden in Gleichung 4.3 sind Störglieder, die es in Hinblick auf die Berechnung der Energieeffizienz zu eliminieren gilt.

Dafür kann Gleichung 4.3 genutzt werden, um mit den experimentellen Ergebnissen zum Gesamtenergieverbrauch und den Auslastungen ein lineares Gleichungssystem zu formen. Da Einzelexperimente für insgesamt fünf *Cassandra*-Clustergrößen durchgeführt wurden, entstanden fünf lineare Gleichungssysteme, deren Struktur mit Gleichung 4.3 identisch sind, sowie 10 Unbekannten (W_0 bis W_9). Jedes dieser linearen Gleichungssysteme besteht aus 42 linearen Gleichungen (210 Einzelexperimente $\div$ 5 *Cassandra*-Clustergrößen). Nach *Bronstein et al.* [BSMM99, S.271] kann jedes der fünf linearen Gleichungssysteme als $A\vec{x} = \vec{a}$ dargestellt werden. A bezeichne hierbei eine Koeffizientenmatrix mit 10 Spalten und 42 Zeilen. Die Werte dieser Matrix ergeben sich aus den experimentell gemessenen Werten der Einzelexperimente. Der Vektor $\vec{x}$ besteht aus 10 Komponenten x_0 bis x_9 und entspricht den 10 Unbekannten. Der Vektor $\vec{a}$ besteht aus 42 Komponenten und entspricht den gemessenen Gesamtenergieverbrauch eines Teilexperimentes für eine getestete *Cassandra*-Clustergröße.

Der Vorteil der entstandenen linearen Gleichungssysteme, dargestellt durch $A\vec{x} = \vec{a}$, ist deren Lösung nach dem Eliminationsverfahren von *Gauss* [BSMM99, S.275]. Zur Lösung wurde die Applikation *RStudio* in der Version 0.99.902 als grafische Oberfläche für die Programmiersprache *R* in der Version 3.2.2 eingesetzt. Diese Programmiersprache wird dafür eingesetzt, statistische Berechnungen durchzuführen und kann durch mittlerweile zahlreiche Zusatzpakete erweitert werden. Die Lösungen der fünf linearen Gleichungssysteme wurden genutzt, um wie oben beschrieben den Gesamtenergieverbrauch zu korrigieren. Mit den korrigierten Werten wurden dann auch die Werte für die Energieeffizienz berechnet. Die Grafiken 4.15, 4.16 sowie die Grafiken A.3 und A.4 im Anhang zeigen die korrigierten Werte.

Betrachtet man den durchschnittlichen Gesamtenergieverbrauch in Abbildung 4.15, so ist erkennbar, dass er mit steigender *Cassandra*-Clustergröße sinkt. Dieses Verhalten konnte auch im Experiment F beobachtet werden, dessen experimentellen Ergebnisse im nachfolgenden Textabschnitt 4.2.2 erläutert werden. Die Analyse der gemessenen Energieverbräuche von Experiment F treffen auch für diejenigen zu, die in Abbildung 4.15 dargestellt werden. Um textuelle Redundanzen zu vermeiden, wird auf die Erläuterungen in nachfolgenden Textabschnitt 4.2.2 verwiesen.

Die steigenden Performancewerte bei gleichzeitigem Sinken des Energieverbrauches führte dazu, dass bei steigender Anzahl an *Cassandra*-Clusterknoten auch die Energieeffizienz stieg. Dieses Verhalten kann in Abbildung 4.15 erkannt werden. Insgesamt kann also für die Arbeitslast `load` des Anwendungsfalles A von Experiment E konstatiert werden, dass es aufgrund der experimentellen Ergebnisse sinnvoll ist, eine konstante Datenmenge mit einer höheren Anzahl von Clusterknoten für einen *Cassandra*-Cluster verarbeiten zu lassen.

Die experimentellen Ergebnisse der übrigen Arbeitslasten `read`, `write` und `mixed` für Anwendungsfall A des Experimentes E bestätigen vollumfänglich die Ergebnissen anderer Studien, zum Beispiel [PPR+11], [KKR14], [CST+10] und [INI+15]. Das bedeutet, dass die Performance bei Schreib- und Leseoperationen nahezu linear mit der Anzahl der beteiligten *Cassandra*-Clusterknoten wächst. Dieses Verhalten kann auf den Abbildungen zu diesen Arbeitslasten beobachtet werden, die aus Platzgründen im Anhang dieser Dissertation zu finden sind (Textabschnitt A.1.3 auf Seite 228). Die experimentellen Ergebnisse der genannten Arbeitslasten zeigen auch dasselbe Verhalten im Bezug auf den Energieverbrauch und die Energieeffizienz, wie es bereits für die Arbeitslast `load` beschrieben ist.

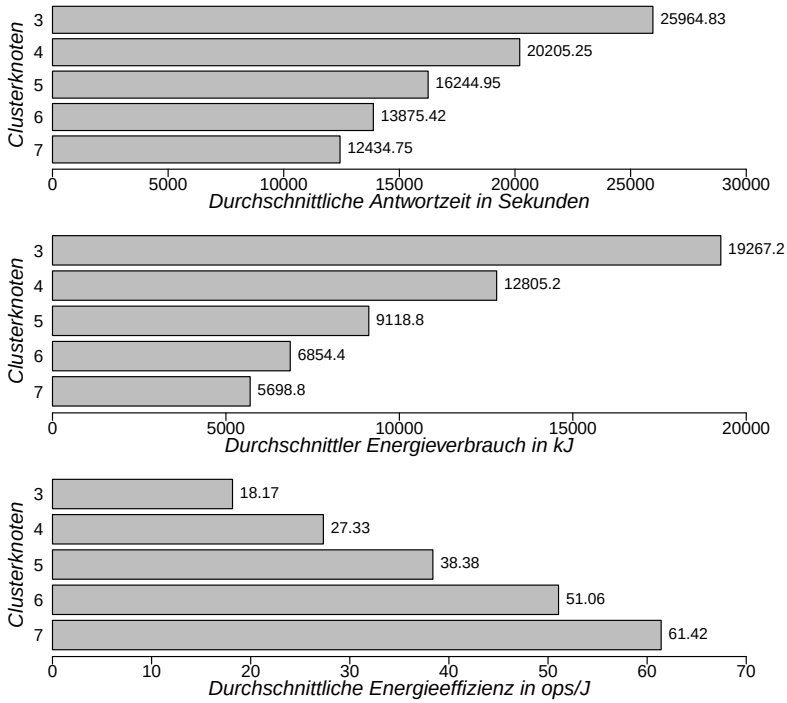


Abbildung 4.15: Experimentelle Ergebnisse für die Arbeitslast `load` für Anwendungsfall A des Experimentes E

Vergleich man die Antwortzeiten und den Energieverbrauch der Arbeitslasten `read`, `write` und `mixed` im Bezug auf die getesteten Verteilungen (diskrete Gleichverteilung und *Zipf*-Verteilung) in Abbildung A.3 auf Seite 229, so fällt auf, dass die diskrete Gleichverteilung performanter bei Lastfällen ist, die Leseoperationen beinhalten, also `read` und `mixed`. Bei reinen Schreiboperationen wie im Lastfall `write`, ergeben sich kaum Unterschiede hinsichtlich der Performance. Der Grund dafür ist, dass die Architektur von *Cassandra* extensiv Gebrauch von Puffern macht, um den Datendurchsatz zum Massenspeicher massiv zu beschleunigen. Nach [Hew11, S.87 ff.]

werden Schreiboperationen auf Datensätze zunächst im Arbeitsspeicher gepuffert und gesammelt, bevor sie asynchron und sequentiell in die Datenbankdateien auf dem Massenspeicher geschrieben werden. Bei Leseoperationen auf Datensätze werden diese zuerst in den Puffer geschrieben, sofern sie noch nicht enthalten sind. Bei einer diskreten Gleichverteilung der Leseoperationen werden alle Clusterknoten eines *Cassandra*-Clusters gleichmäßig belastet. Bei einer *Zipf*-Verteilung der Leseoperationen werden hingegen wenige Clusterknoten stärker belastet als die übrigen. Angesichts der unterschiedlichen Lastverteilung ist es also verständlich, dass die Performance der Lastfälle `read` und `mixed` bei Nutzung einer diskreten Gleichverteilung der Leseoperationen höher ausfällt. Dieses Verhalten relativiert sich aber bei Schreiboperationen aufgrund der beschriebenen Schreibstrategie von Datensätzen innerhalb von *Cassandra*.

Experimentelle Ergebnisse zu Anwendungsfall B Der Anwendungsfall B von Experiment E soll die Auswirkungen hinsichtlich der Performance, des Energieverbrauches und -effizienz untersuchen, wenn ein zusätzlicher Clusterknoten in einen *Cassandra*-Cluster aufgenommen wird. Das ist zumeist der Fall, wenn beispielsweise die Speicherkapazität des bestehenden *Cassandra*-Cluster nicht mehr ausreicht und erweitert wird. Ein anderer Fall ist es, wenn der *Cassandra*-Cluster an seine Performancegrenzen stößt und die Performance durch einen zusätzlichen Knoten mittels Lastenausgleich Lastausgleich gesteigert werden soll. Beide Fälle können natürlich kombiniert werden, da sich Synergieeffekte ergeben.

Die experimentellen Ergebnisse hinsichtlich der durchschnittlichen Antwortzeit, dem Energieverbrauch und der errechneten Energieeffizienz für die Arbeitslast `load` für Anwendungsfall B des Experimentes E sind in Abbildung 4.16 kombiniert dargestellt. Im Gegensatz zu Anwendungsfall A ergibt sich aber für Anwendungsfall B eine andere Berechnung der Performance. Erstgenannter Anwendungsfall sieht die Speicherung einer konstanten Datenmenge von cica 350 GByte in einem *Cassandra*-Cluster mit steigender Clusterknotenanzahl vor. Anwendungsfall B sieht dagegen die Speicherung von 50 GByte Daten für jeden *Cassandra*-Clusterknoten vor, sodass die Datenmenge insgesamt mit jedem hinzugefügten Clusterknoten wächst.

In Abbildung 4.16 erkennt man diesen Sachverhalt daran, dass die durchschnittliche Antwortzeit mit steigender *Cassandra*-Clustergröße steigt. Das bedeutet, dass das Einfügen der anwachsenden Datenmenge mit jedem hinzugefügten *Cassandra*-Clusterknoten treppenartig mit ansteigt. Im Vergleich mit den durchschnittlichen Antwortzeiten von Szenario A in Abbildung 4.15 auf Seite 104 fallen diejenigen von Szenario B in Abbildung 4.16 aber deutlich geringer aus. Allerdings ist widersprüchlich, dass die Energieverbrauchswerte in Abbildung 4.16 mit ansteigender *Cassandra*-Clustergröße sinken. Die Gründe dafür sind ausführlich in nachfolgenden Textabschnitt 4.2.2 beschrieben. Dies hat denselben Effekt auf die errechnete Energieeffizienz wie sie bereits für Szenario A beschrieben wurde.

Analog zu Szenario A können auch die Antwortzeiten, die Energieverbrauchswerte und die errechneten Werte für die Energieeffizienz der einzelnen Arbeitslasten vergleichen werden. Dazu betrachtet man Abbildung A.4 auf Seite 231. Vergleicht man dabei obige Metriken für die diskrete Gleichverteilung mit denjenigen der *Zipf*-Verteilung, so ist dasselbe Verhalten erkennbar, wie es bereits für Szenario A beschrieben wurde. Das bedeutet, dass die Testreihen, die die diskrete Gleichverteilung genutzt haben, performanter gegenüber denjenigen Testreihen ausgeführt wurden, die die *Zipf*-Verteilung nutzten. Das gilt aber nur für die Arbeitslasten `read` und `mixed`, also Arbeitslasten mit Leseoperationen. Bei der Arbeitslast `write` ergeben sich kaum Unterschiede.

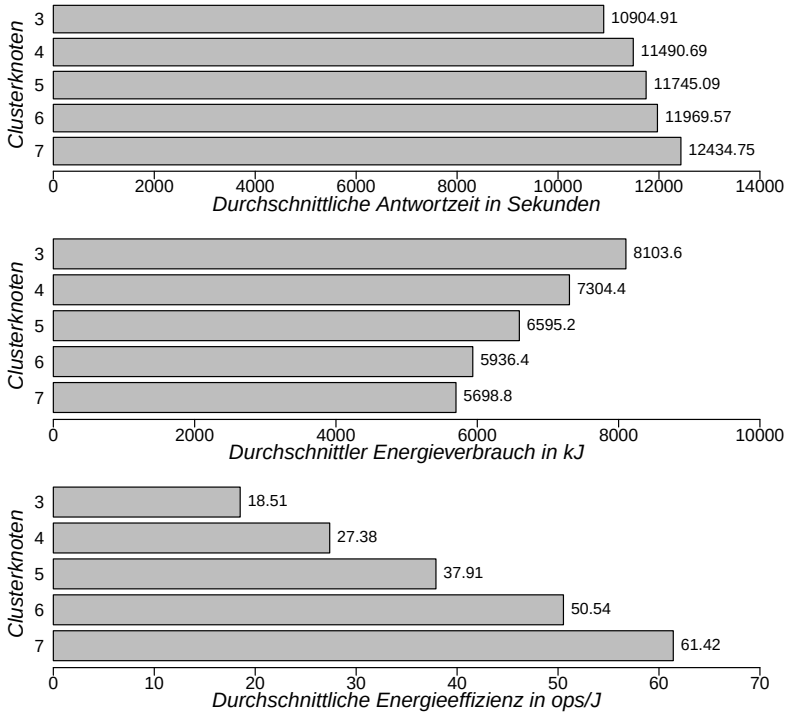


Abbildung 4.16: Experimentelle Ergebnisse für die Arbeitslast `load` für Anwendungsfall B des Experimentes E

Zusätzliche Ergebnisse Neben der Analyse der experimentellen Ergebnisse der beiden Szenarien A und B von Experiment E wurden die errechneten Werte für die Energieeffizienz beider Szenarien einer weitergehenden Analyse unterzogen. Dieses Vorgehen basiert darauf, dass in Abbildung 4.15 für die Antwortzeit und für den Energieverbrauch ein Trend zu erkennen ist. Vergleicht man in Abbildung 4.15 zum Beispiel die durchschnittlichen Antwortzeiten für eine steigende Anzahl an *Cassandra*-Clusterknoten, so wird die Differenz zwei aufeinanderfolgender *Cassandra*-Clustergrößen kleiner. Es kann also angenommen werden, dass die Antwortzeiten für *Cassandra*-Cluster jenseits der getesteten Größe von sieben *Cassandra*-Knoten zwar sinken wird, aber nicht mehr in dem Maße, wie es in Abbildung 4.15 sichtbar ist. Mit anderen Worten ausgedrückt bedeutet es, dass der "Gewinn" an Performance durch Reduzierung der Antwortzeit mittels zusätzlicher *Cassandra*-Clusterknoten geringer werden wird.

Dieselbe Annahme kann auch für den Energieverbrauch und schlussendlich für die Energieeffizienz getroffen werden. Besonders für den letztgenannten Aspekt ist es interessant, bei welcher *Cassandra*-Clustergröße sich das Optimum hinsichtlich der Energieeffizienz im Bezug auf die Arbeitslasten von Experiment E einstellt. Präziser formuliert wird also diejenige Anzahl an *Cassandra*-Clusterknoten gesucht, in der die Energieeffizienz optimal ist. Wie in Textabschnitt 3.3 beschrieben, bezeichnet dieses Optimum den Kompromiss zwischen der Performance und dem dafür aufgewandten Verbrauch an elektrischer Energie.

Dafür wurden jeweils die durchschnittlichen Antwortzeiten und Energieverbrauchswerte der beiden Szenarien A und B, wie sie in den Abbildungen 4.15, 4.16, A.3 und A.4 dargestellt sind, einer linearen Regression unterzogen. Nach *Bronstein et al.* bezeichnet die lineare Regression eine

Form der Regressionsanalyse, um einen funktionalen Zusammenhang zwischen zwei Merkmalen einer Grundgesamtheit anhand von Messwerten zu bestimmen [BSMM99, S.772 ff.]. Auch hier wurde die Applikation *RStudio* verwendet, um zunächst zu bestimmen, ob generell ein funktionaler Zusammenhang im Sinne der Korrelationsanalyse besteht. Es zeigte sich, dass die notwendigen Korrelationskoeffizienten berechnet werden konnten. Die entstandene Regressionsgerade konnte dann genutzt werden, um weitere Werte hinsichtlich der Antwortzeit und des Energieverbrauches zu extrapolieren. Sie wurden genutzt, um die Energieeffizienz für *Cassandra*-Cluster zu berechnen, die außerhalb der getesteten Größen lagen. Es zeigte sich, dass die Energieeffizienz weiterhin ansteigt, aber der Effizienzgewinn mit jedem zusätzlichen *Cassandra*-Clusterknoten sinkt.

Durch diesen Umstand wurde die Festlegung getroffen, dass die energieeffizienteste *Cassandra*-Clustergröße diejenige sein soll, bei der der Energieeffizienzgewinn unter 10 Prozent im Vergleich zur Vorgänger-Clustergröße sein soll. Diese Bedingung tritt laut den extrapolierten Energieeffizienzwerten bei einem *Cassandra*-Cluster mit 10 Knoten auf. Genauer gesagt, diese Clustergröße stellt das Optimum im Bezug auf die Energieeffizienz dar, sofern obige Bedingung beachtet wird. In diesem Zusammenhang muss aber darauf hingewiesen werden, dass dies nur ein rechnerisches Optimum ist. Im Rahmen von Experiment E konnte diese Annahme nicht experimentell belegt werden. Um die Annahme zu bestätigen, wurde ein neues Experiment durchgeführt, dessen experimentelle Ergebnisse im nachfolgenden Textabschnitt beschrieben sind.

4.2.2 Experimentelle Ergebnisse für einen 13-Knoten-*Cassandra*-Cluster

Im Experiment E, dessen experimentelle Ergebnisse im vorherigen Textabschnitt 4.2.1 beschrieben sind, wurde die Performance, der Energieverbrauch und die errechnete Energieeffizienz eines *Cassandra*-Clusters mit sukzessiv ansteigender Anzahl an Clusterknoten zwischen drei und sieben betrachtet. Dabei wurden Experimente zu zwei Anwendungsfällen durchgeführt. Das Ziel eines dieser Anwendungsfälle war es, die Anzahl der notwendigen *Cassandra*-Clusterknoten zu bestimmen, für die die Energieeffizienz optimal ist.

Durch lineare Regressionsanalyse der experimentellen Werte für die Performance und dem Energieverbrauch wurde errechnet, dass ein *Cassandra*-Cluster mit 10 Clusterknoten im Bezug auf den Anwendungsfall am energieeffizientesten ist. Diese Annahme konnte im Rahmen von Experiment E jedoch nicht belegt werden, da nur Clustergröße bis sieben *Cassandra*-Clusterknoten getestet wurden. Außerdem bot der technische Versuchsaufbau, genauer gesagt das verwendete Bladecenter, nicht genug Knoten, um einen *Cassandra*-Cluster mit 10 Knoten im Rahmen von Experiment E zu testen. Zudem war der technische Testaufbau administrativ beschränkt. Das bedeutet, die verwendeten technischen Komponenten konnten nicht ausgeschaltet beziehungsweise in einen Energiesparmodus versetzt werden. Das resultierte in einer Korrektur der experimentell gemessenen Energieverbrauchswerte.

Um die angenommene optimale, energieeffizientesten *Cassandra*-Clustergröße auch zu belegen, wurde ein weiteres Experiment durchgeführt. Prinzipiell wird dabei Anwendungsfall A von Experiment E auf Seite 98 mit einer größeren Anzahl an *Cassandra*-Clusterknoten wiederholt.

> **Experiment F: Cassandra-Cluster mit 13 Clusterknoten und *YCSB* als Benchmark**
> Dieses Experiment sieht die Wiederholung von Anwendungsfall A des Experimentes E vor. Das bedeutet, dass für eine feste Datenmenge die Performance und der Energieverbrauch eines *Cassandra*-Clusters experimentell ermittelt werden soll, wobei die Clustergröße suksessiv von drei auf 13 *Cassandra*-Clusterknoten erhöht wird.

Das Ziel ist es, experimentell zu belegen, dass die rechnerisch ermittelte Clustergröße von 10 Clusterknoten die energieeffizienteste hinsichtlich der feste Datenmenge ist.

Zur experimentellen Bestimmung der Performance und des Energieverbrauches wird wie in Experiment E *YCSB* benutzt, wobei die in Experiment E definierten Arbeitslasten `load`, `read`, `write` und `mixed` ausgeführt werden.

Wie bereits eingangs erwähnt, lässt sich die Notwendigkeit von Experiment F gegenüber Experiment E damit erklären, dass der technische Versuchsaufbau nicht die Möglichkeit bot, die gestiegene, notwendige Anzahl an *Cassandra*-Clusterknoten bereitzustellen. Der Grund ist, dass das im Experiment E verwendete Bladecenter nur eine maximale Kapazität von 10 Einschüben in Form von Blades hat. Es war also für die Durchführung von Experiment F notwendig, den technischen Versuchsaufbau zu modifizieren, um Experimente mit einem *Cassandra*-Cluster durchzuführen, die bis zu 13 Clusterknoten umfassen.

Der technische Testaufbau für Experiment F ist in Abbildung 4.17 auf Seite 109 schematisch dargestellt. Im Gegensatz zum technischen Testaufbau von Experiment E, der in Abbildung 4.14 auf Seite 101 dargestellt ist, wurden anstatt einem zwei Bladecenter vom Typ *Fujitsu BX600 S3* verwendet, die damit ein Bladecenter-Verbund bildeten. Um den Datenaustausch zwischen beiden benutzten Bladecentern zu gewährleisten, wurde ein Switch vom Typ *Linksys SRW2048* benutzt, an dem beide Bladecenter angeschlossen wurden. Da beide Bladecenter mehrere Netzwerkschnittstellen bereitstellten und auch die Möglichkeit boten, sie zu bündeln, konnte eine Bandbreite von 6 GBit pro Sekunde zwischen beiden Bladecentern erreicht werden. Die Knoten der Bladecenter sind mit deren internen Backplanes verbunden, die sie mit Strom und Netzwerkzugang mit einer Bandbreite von 1 GBit/s versorgt. Die technischen Spezifikationen der genutzten Bladecenter im Bezug auf Hard- und Software sind in Tabelle 4.5 aufgelistet. Knoten B1 des Bladecenters 1 wurde exklusiv dafür genutzt, den *YCSB*-Client auszuführen, der auf den *Cassandra*-Cluster zugriff. Die übrigen Bladecenter-Knoten B2 bis B14 wurden dazu genutzt, einen *Cassandra*-Cluster zu bilden, wobei die Clustergröße zwischen drei und 13 Clusterknoten betrug. Bei jeden Clusterknoten wurden die lokalen Festplatten (Massenspeicher) zu einem RAID-0 verbunden.

Komponente	**Clusterknoten B1**	**Clusterknoten B2 bis B14**
CPU	2×AMD Opteron 890	2×AMD Opteron 870
Hauptspeicher	32 GByte DDR-2 reg.	16 GByte DDR-2 reg.
Massenspeicher	2×300 GByte	2×146 GByte
Betriebssystem	*Ubuntu Server* 12.04 LTS x64	
Java Runtime	*Oracle* JRE 1.7.0.60-b19	
Cassandra	*DataStax Enterprise Server* 4.5.0	

Tabelle 4.5: Spezifikationen des *Cassandra*-Clusters für Experiment F

Wie bereits erwähnt, wurden die Bladecenter-Knoten B2 bis B14 dafür genutzt, einen *Cassandra*-Cluster mit ansteigender Knotenanzahl zu bilden. In Abbildung 4.17 ist ein minimaler *Cassandra*-Cluster mit drei Clusterknoten dargestellt. Zur Messung des Energieverbrauches wurde ein geeichtes Leistungsmessgerät vom Typ *EMH DMTZ-XC* genutzt. Es maß die elektrische Leistungsaufnahme beider Bladecenter sowie des Switches. Wie auch in Experiment E wurde das Leistungsmessgerät alle 15 Sekunden durch einen externen Server nach der verbrauchten elektrischen Leistung abgefragt und zur späteren Analyse in eine Datenbank gespeichert. Um zu

gewährleisten, dass die Zeitstempel der abgefragten Messwerte übereinstimmen, wurden alle Komponenten des Testaufbaus (die Bladecenter, alle Clusterknoten, das Leistungsmessgerät sowie der externe Server zur Speicherung der Messwerte) periodisch zeitlich synchronisiert.

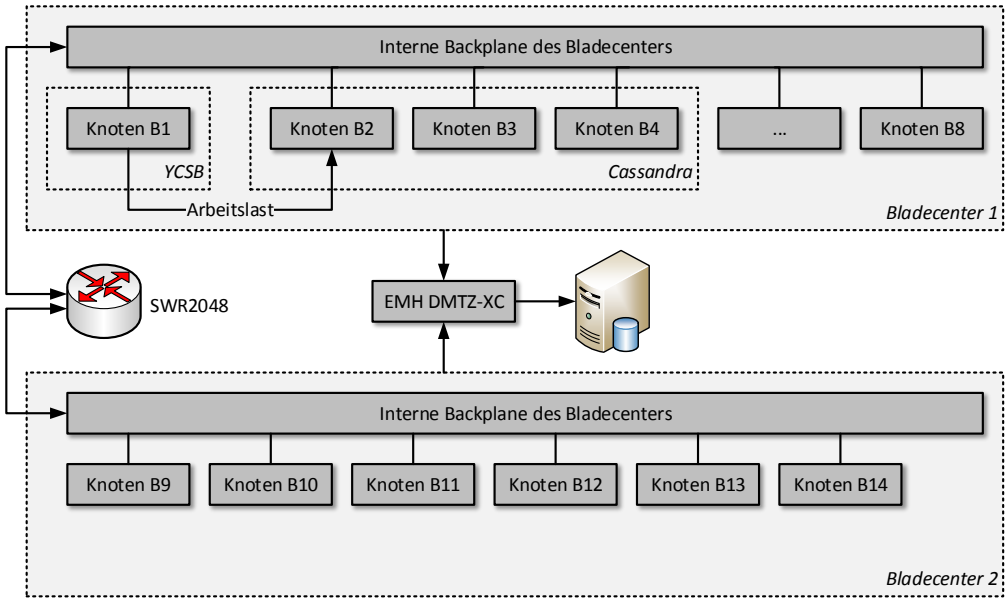


Abbildung 4.17: Schematischer Testaufbau für Experiment F auf einem Bladecenter-Verbund mit 14 Knoten. Knoten B1 dient als *YCSB*-Client, der auf einen *Cassandra*-Cluster mit variabler Knotenanzahl zugreift (Knoten B2 bis B14). Dargestellt ist ein minimaler *Cassandra*-Cluster mit drei Clusterknoten.

Im Bezug auf die Testmethodik von Experiment F wurde aus Gründen der Vergleichbarkeit auf dieselbe von Experiment E zurückgegriffen, wie sie in Textabschnitt 4.2.1 für Anwendungsfall A beschrieben ist. Das bedeutet, dass als Benchmark der *Yahoo Cloud Serving Benchmark* (*YCSB*) genutzt wurde, wobei vier Arbeitslasten in einer Testserie zusammengefasst wurden: in der Arbeitslast `load` wurden 350 Millionen Einfügeoperationen ausgeführt, wodurch die vormals leere *YCSB*-Datenbank mit rund 350 Millionen Datensätzen befüllt wurde. Da jeder Datensatz circa 1 kByte groß ist, ergab das eine Datenbank mit rund 350 GByte Daten. Durch die anschließend durchgeführte Arbeitslast `read` wurden 10 Millionen Leseoperationen ausgeführt. Danach wurden durch die Arbeitslast `write` 10 Millionen Schreiboperationen auf die *YCSB*-Datenbank ausgeführt. Zum Schluss wurden durch die Arbeitslast `mixed` zunächst 5 Millionen Schreib- und dann 5 Millionen Leseoperationen ausgeführt. Für die drei letztgenannten Arbeitslasten entsprachen die 10 Millionen Operationen einer Datenmenge von rund 10 GByte. Zudem wurden diese Arbeitslasten so konfiguriert, dass die ausgeführten Operationen diskret gleichverteilt waren. Das heisst, dass jeder Datensatz, der gelesen oder modifiziert wurde, per Zufall ausgewählt wurde und dabei jeder Datensatz dieselbe Wahrscheinlichkeit besitzt, ausgewählt zu werden. Insgesamt wurden 132 Einzelexperimente durchgeführt (4 Arbeitslasten $\times$ 11 *Cassandra*-Clustergrößen $\times$ 3 Wiederholungen).

Experimentelle Ergebnisse zur Performance Die Analyse der experimentellen Ergebnisse zeigt, dass sich die Performance in Form der Antwortzeiten superlinear verhält. Um diesen Sachverhalt zu illustrieren, kann zunächst Gleichung 4.2 für den *speedup* auf Seite 97 betrachtet werden. In dieser Gleichung wird die Menge Z_c genutzt, wie sie in Gleichung 4.1 definiert ist. Welche Elemente in dieser Menge enthalten sind, hängt von den beiden Variablen n und m ab. Im Bezug auf Gleichung 4.2 geben sie die untere und obere Anzahl an Knoten im einen Cluster an. Im Fall von Experiment F ist also $n = 3$ und $m = 13$. Somit ergibt sich die Menge $Z_c = \{3, 4, 5, \ldots, 13\}$.

In Gleichung 4.2 kann die Funktion *metrik* als Funktion aufgefasst werden, die den getesteten *Cassandra*-Clustergrößen von Experiment F die gemessene Antwortzeit zuweist. Es wurde festgestellt, dass für zwei beliebige $z_1, z_2 \in Z_c$ folgende Ungleichung gilt, wobei $z_1 < z_2$ ist:

$$s(z_1, z_2 \in Z_c | z_1 < z_2) > 1$$

Nach der Definition des Wertebereiches für den *speedup* in Gleichung 4.2 verhält sich also die Performance für alle getesteten *Cassandra*-Clustergrößen von Experiment F superlinear.

Mit anderen Worten ausgedrückt, bedeutet das beobachtete superlineare Verhalten der Performance, dass sie sich für dieselbe Arbeitslast mehr als verdoppelt, wenn man die Anzahl der *Cassandra*-Clusterknoten verdoppelt. Bezogen auf die Antwortzeit t, die es gebraucht hat, um eine Arbeitslast in Form einer Testserie anhand von *YCSB* durchzuführen, bedeutet es, dass die Antwortzeit weniger als die Hälfte betrug, wenn man die Anzahl der *Cassandra*-Clusterknoten verdoppelt. Nach *Amdahl*'s Gesetz, dass auf Seite 97 beschrieben ist, kann der Performance-*speedup* aber nicht mehr als linear sein, also $s(z_1, z_2) \leq 1$. In der Realität ist dieser *speedup* aber sublinear ($s(z_1, z_2) < 1$). Die experimentellen Ergebnisse stehen also im Widerspruch zu *Amdahl*'s Gesetz. Dazu muss aber bemerkt werden, dass dieses Gesetz viele Diskussionen auf sich gezogen hat. Ein Argument gegen dieses Gesetz ist, dass es die Parallelverarbeitung mittels eines Clusters sowie Puffer und Caches nicht berücksichtigt [SM08].

Allerdings ist aus der Analyse der experimentellen Ergebnisse von Experiment E bekannt, dass das eingesetzte verteilte Datenbanksystem *Cassandra* genau diese Konzepte nutzt, um die Performance zu steigern. Zudem zeigen auch andere Experimente, dass verteilte Datenbanksysteme durchaus einen superlinearen Performance-*speedup* haben können. Zum Beispiel wurde in der Studie von *Lang et al.* das verteilte, relationale Datenbanksystem *Vertica* in einem Cluster mit bis zu 16 Clusterknoten anhand des *TPC-H*-Benchmark untersucht [LHP+12]. Die Analyse der experimentellen Performancewerte zeigte auch einen superlinearen *speedup*, der durch Puffereffekte erklärt werden konnte.

Eine detaillierte Analyse der experimentellen Ergebnisse von Experiment F zeigt aber, dass der superlineare *speedup* hinsichtlich der Performance nur schwach ausgeprägt ist. So betrug die Differenz zwischen der Superlinearität und der Linearität nur zwischen zwei und acht Prozent. Zudem zeigt sich, dass sich auch der Performance-*speedup* bei den durchgeführten Testserien unterschied. Dabei war er bei den Testserien `load` und `write` (den Testserien mit Fokus auf Schreiboperationen) höher als bei den Testserien `read` und `mixed` (den Testserien mit dem Schwerpunkt auf Leseoperationen). Diese Beobachtung bestätigt das Verhalten von *Cassandra*, dass Schreiboperationen durch Puffer unter Umständen performanter ausgeführt werden als Leseoperationen (siehe experimentelle Ergebnisse von Experiment E auf Seite 98). Zur massiven Puffernutzung trug auch bei, dass die genutzten Clusterknoten nach Tabelle 4.5 auch eine große Arbeitsspeichergröße mit 16 GByte hatten, wodurch *Cassandra* in der Lage war, große Puffer anzulegen und zu nutzen.

Zusätzlich profitierten die experimentell gewonnenen Performancewerte von einem recht kleinen Abdeckungsgrad[11] von 2.8 Prozent. Das bedeutet praktisch, dass im Prinzip alle Zugriffe auf den Massenspeicher im Rahmen der durchgeführten *YCSB*-Testserien gepuffert werden konnten.

Zusammengefasst kann konstatiert werden, dass durch die Architektur von *Cassandra*, dem Abdeckungsgrad, dem technischen Versuchsaufbau und die Testmethodik ein superlinearer *speedup* der Performance möglich ist und auch beobachtet werden konnte.

Experimentelle Ergebnisse zum Energieverbrauch Die Auswertung der experimentellen Ergebnisse zum Energieverbrauch hat ein sublineares Verhalten aufgezeigt. Zur Illustration kann analog zum Performance-*speedup*, der im vorhergehenden Textabschnitt beschrieben wurde, der Energieverbrauchs-*speedup* errechnet und analysiert werden. Dafür kann die Funktion *metrik* in Gleichung 4.2 auf Seite 97 als eine Funktion aufgefasst werden, die den getesteten *Cassandra*-Clustergrößen den gemessenen Energieverbrauch zuweist. Insgesamt hat sich für alle Clustergrößen $z_1, z_2 \in Z_c$ die folgende Ungleichung ergeben, wobei $z_1 < z_2$ ist:

$$s(z_1, z_2 \in Z_c | z_1 < z_2) < 1$$

Damit kann festgestellt werden, dass der Energieverbrauchs-*speedup* für alle getesteten *Cassandra*-Clustergrößen von Experiment F sublinear sind.

Im allgemeinen kann immer angenommen werden, dass sich der Energieverbrauch eines Clusters mindestens verdoppelt, wenn die Anzahl der Clusterknoten verdoppelt wird. Das bedeutet, dass der Energieverbrauchs-*speedup* mindestens linear, in Realität aber superlinear ist. Die experimentellen Ergebnisse zeigen aber genau das Gegenteil: verdoppelt man die Anzahl der *Cassandra*-Clusterknoten, so betrug der experimentell gemessene Energieverbrauch weniger als das Doppelte. Der Energieverbrauchs-*speedup* ist somit sublinear. Überträgt man *Amdahl*'s Gesetz auf den Energieverbrauch, so stehen die experimentellen Messergebnisse im genauen Widerspruch dazu. Allerdings ist *Amdahl*'s Gesetz neben der Performancebetrachtung auch bei der Betrachtung des Energieverbrauches umstritten. In diesem Punkt ist zum Beispiel nach *Cho und Melhem* das Hauptgegenargument, dass *Amdahl*'s Gesetz Mechanismen und Technologien außer Acht lässt, die sowohl den Energieverbrauch als auch die Performance beeinflussen [SM08]. Dazu zählen zum Beispiel alle Technologien, die Energiesparmechanismen etablieren, etwa die Senkung der CPU-Taktrate in Abhängigkeit zur Auslastung (siehe dazu Textabschnitt 3.2 auf Seite 58).

Daher wurde eine genaue Analyse der Energieverbrauchswerte in Abhängigkeit zur Auslastung der Clusterknoten vorgenommen. Wie in Textabschnitt 3.2 auf Seite 58 beschrieben, sollte sich der Energieverbrauch idealerweise proportional zur Auslastung verhalten. Das bedeutet, dass der Energieverbrauch linear mit der Auslastung ansteigt. Die Analyse ergab jedoch, dass sich der Energieverbrauch weder proportional noch linear in Abhängigkeit zur Auslastung verhält. In Abbildung 4.18 auf Seite 112 ist der Energieverbrauch eines Clusterknoten im Rahmen der Testserien von Experiment F für verschieden Lastpunkte dargestellt. In dieser Abbildung sind der ideale energie-proportionale sowie der lineare Energieverbrauch als gestrichelte Linien dargestellt. Der gemessene Energieverbrauch für ausgewählte Lastpunkte ist als verbundene Linie gezeigt. Dabei wird er als Prozentsatz zur gemessenen maximalen Energieverbrauch angegeben. In Abbildung 4.18 erkennt man, dass der Energieverbrauch zwischen 52 und 76 Prozent Auslastung unterhalb des linearen Energieverbrauches ist. Mit anderen Worten bedeutet es, dass

[11] Der Abdeckungsgrad gibt den Anteil der tatsächlich durchgeführten Operationen relativ zu den theoretisch möglichen wider. Bei den durchgeführten *YCSB*-Testserien ist der Abdeckungsgrad immer 10 Millionen *ops* ÷ 350 Millionen *ops* · 100 = 2.8 Prozent.

im Bezug auf den Energieverbrauch die Clusterknoten am effizientesten arbeiteten, wenn sie zwischen 52 und 76 Prozent ausgelastet waren.

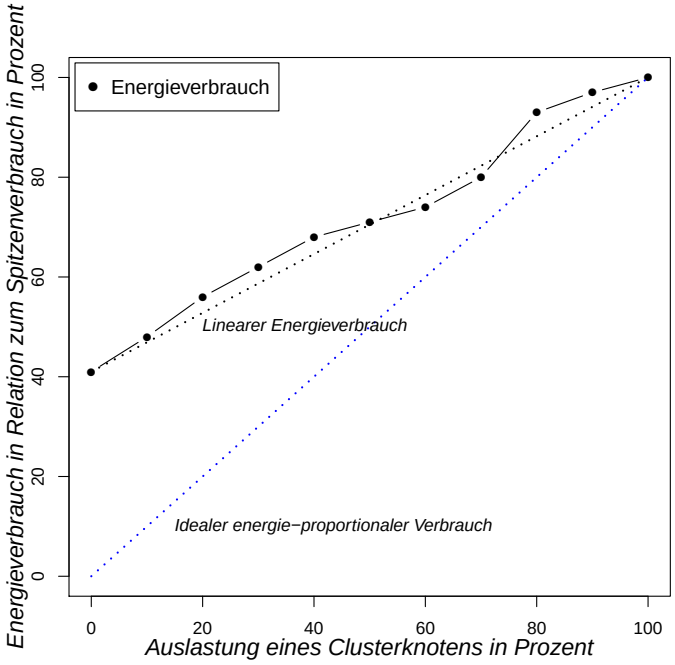


Abbildung 4.18: Energieverbrauch eines Clusterknotens in Relation zum Spitzenenergieverbrauch für verschiedene Lastpunkte

Zudem ergab die Analyse, dass die durchschnittliche Auslastung der *Cassandra*-Clusterknoten sank, je größer die getesteten *Cassandra*-Cluster hinsichtlich der Anzahl an Clusterknoten wurde. Das ist insofern verständlich, da die Arbeitslast in Form der *YCSB*-Testserien mit steigender *Cassandra*-Clustergröße auf mehr Clusterknoten verteilt wird. Dementsprechend sank die Auslastung für jeden beteiligten *Cassandra*-Clusterknoten und desto mehr Clusterknoten wurden in den beschriebenen Auslastungsbereich verschoben, in dem sie am effizientesten arbeiten.

Zusammengefasst erklärt sich der sublineare *speedup* im Bezug auf den Energieverbrauch mit dem superlinearen *speedup* der Performance in Verbindung mit der Auslastung der Clusterknoten. Wie bereits beschrieben, führte der superlineare *speedup* der Performance dazu, dass die Antwortzeit einer *YCSB*-Arbeitslast weniger als das Doppelte beträgt, wenn man die Anzahl der *Cassandra*-Clusterknoten verdoppelt. Daraus resultierte, dass der Zeitrahmen für die Messung des Energieverbrauches mit steigender *Cassandra*-Clustergröße sank. Mit steigender *Cassandra*-Clustergröße sank aber die Auslastung der beteiligten Clusterknoten, wodurch sie in den Auslastungsbereich gelangten, in dem sie am effizientesten arbeiteten. Diese genannten Aspekte führten dazu, dass der Energieverbrauch schlussendlich ein sublineares *speedup*-Verhalten aufwies.

Ergebnisse zur Energieeffizienz Basierend auf den experimentellen Messergebnissen zur Performance und dem Energieverbrauch für alle Testserien von Experiment F war es möglich, die Energieeffizienz nach Gleichung 3.3 auf Seite 64 zu errechnen.

Es soll an dieser Stelle darauf hingewiesen werden, dass der technische Versuchsaufbau von Experiment F nicht dieselben administrativen Einschränkungen besaß wie der Versuchsaufbau von Experiment E. Dementsprechend war es nicht notwendig, die Energieverbrauchswerte zu korrigieren und die Energieeffizienzwerte neu zu berechnen.

In Textabschnitt 3.3, in der oben genannte Gleichung beschrieben wird, wird auch die Interpretation von Gleichung 3.3 zur Optimierung der Energieeffizienz betrachtet. Sie impliziert, dass die Energieeffizienz dahingehend optimiert werden kann, indem die Performance auf Kosten des Energieverbrauches erhöht oder der Energieverbrauch auf Kosten der Performance gesenkt wird. Allerdings zeigen die experimentellen Ergebnisse der Performance und dem Energieverbrauch von Experiment F, dass beide Optimierungsziele gleichzeitig erreicht werden. Die errechnete Energieeffizienz für alle getestete *Cassandra*-Clustergrößen ist in Abbildung 4.19 durch graue Balken dargestellt. In dieser Abbildung ist zu erkennen, dass aufgrund der doppelten Optimierung die Energieeffizienz mit der Anzahl der beteiligten *Cassandra*-Clusterknoten steigt. Vom praktischen Standpunkt aus betrachtet, ist es also sinnvoller, die genutzten Arbeitslasten mit einer größeren *Cassandra*-Clustergröße durchzuführen, selbst wenn der Energieverbrauch dadurch ansteigt.

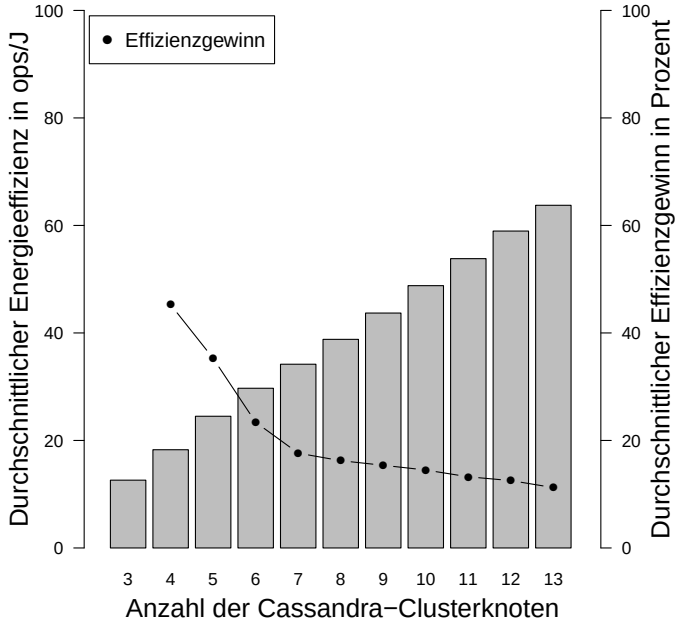


Abbildung 4.19: Durchschnittliche Energieeffizienz für die getesteten *Cassandra*-Clustergrößen aus Experiment F[12]

[12] Aus Darstellungsgründen wurde Abbildung 4.19 nicht numerisch annotiert. Die numerischen Werte sind im Anhang A.1.4 tabellarisch dargestellt.

Zusätzlich zeigt Abbildung 4.19 den “Gewinn” an Energieeffizienz, wenn ein getesteter *Cassandra*-Cluster um einen Clusterknoten erweitert wird. Es ist also der prozentuale Anstieg der Energieeffizienz eines *Cassandra*-Clusters mit n gegenüber einem mit $n - 1$ Clusterknoten. Dieser Anstieg ist in Abbildung 4.19 als verbundene, schwarze Linie dargestellt. Ein Vergleich der prozentualen Anstiege zeigt, dass sie beginnend mit einer *Cassandra*-Clustergröße von 11 Clusterknoten unter die Marke von 10 Prozent fällt und weiter fällt. Als Resultat kann also konstatiert werden, dass mit einem *Cassandra*-Cluster bestehend aus 10 Clusterknoten die gestellten Arbeitslasten am energieeffizientesten ausgeführt werden. Somit konnte das Ziel von Experiment F erfüllt werden. Das heisst, die im Experiment E errechnete optimale Anzahl an *Cassandra*-Clusterknoten konnte experimentell in Experiment F belegt werden.

4.3 Kapitelzusammenfassung

Im vorherigen Kapitel 3 wurden die drei Metriken identifiziert und erläutert, die für einen objektiven Vergleich der Eignung oder Güte eines Datenbanksystems herangezogen werden können. Dabei handelt es sich um die Performance, dem Energieverbrauch und die Energieeffizienz. Für diese drei Metriken wurden in den Textabschnitten 3.1, 3.2 und 3.3 die Faktoren evaluiert, die einen potentiellen Einfluss nehmen können. Dazu gehören nicht nur technische Faktoren wie die vier Kernkomponenten CPU, Arbeits- und Massenspeicher sowie das Mainboard, sondern auch die diversen Konfigurationsmöglichkeiten von Datenbanksystemen und Betriebssystemen.

In Kapitel 4 sind die Ergebnisse von Experimenten dargestellt und erläutert worden, die die Größe des Einflusses bestimmen sollen. Das bedeutet, dass die drei oben genannten Vergleichsmetriken experimentell betrachtet wurden. Dabei wurden sowohl Experimente für ein zentral betriebenes als auch verteiltes Datenbanksystem durchgeführt. Um objektive, vergleichbare Metrikwerte zu gewinnen, wurden in den Experimenten anerkannte Benchmarks genutzt, unter anderem *TPC-H*, *StarSchema* sowie *Yahoo Cloud Serving Benchmark* (*YCSB*).

Die Experimente mit den zentral betriebenen Datenbanksystem *Postgresql*, deren Ergebnisse in Textabschnitt 4.1 beschrieben wurden, zeigen sehr deutlich, dass bereits geringe Änderungen an der Konfiguration des Testaufbaus signifikante Auswirkungen auf die drei oben genannten Metriken haben können. So konnte beobachtet werden, dass *Postgresql* sehr sensibel auf Änderungen der Arbeitsspeicher-Konfiguration reagiert. Diese Konfiguration sagt aus, welchen Anteil am insgesamt verfügbaren Arbeitsspeicher *Postgresql* exklusiv zur Verfügung gestellt wurde, zum Beispiel für Zwischenpuffer und Caches. Es zeigte sich, dass diese Puffer im Rahmen der genutzten Benchmarks kaum bis selten genutzt wurden. Im Gegenteil, dadurch stand die Situation, dass dem zugrunde liegenden Betriebssystem weniger Arbeitsspeicher zur Verfügung stand, um zum Beispiel den Zugriff auf Dateien zu puffern. Da *Postgresql* ein Datenbanksystem ist, dass stark vom Zugriff auf seine Datenbankdateien abhängig ist, entstand die Situation, dass die Ausführung eines Benchmark komplett pausiert wurde, da das Betriebssystem Pufferoperationen durchführte (*swapping*). Das resultierte dann in einer deutlichen Verschlechterung der Performance und damit der Energieeffizienz.

Die Experimente mit dem verteilten Datenbanksystem *Cassandra*, deren Ergebnisse in Textabschnitt 4.2 erläutert sind, zeigten dagegen ein Verhalten, das gegen *Amdahl*'s Gesetz spricht. Die Experimente nutzen einen Versuchsaufbau, der aus sieben beziehungsweise 13 *Cassandra*-Clusterknoten bestand. Die Performance wurde anhand des *YCSB* evaluiert. Hier konnte beobachtet werden, dass sich die Performance mehr als verdoppelte, wenn die *Cassandra*-Clustergröße verdoppelt wurde (superlinearer *speedup*). Hinsichtlich des Energieverbrauch zeigte sich,

dass dieser weniger als das Doppelte betrug, wenn man ebenfalls die *Cassandra*-Clustergröße verdoppelte (sublinearer *speedup*). Das resultierte in einer Situation, dass die Energieeffizienz tatsächlich dadurch gesteigert werden konnte, indem eine möglichst größer *Cassandra*-Cluster genutzt wird.

Es sei an dieser Stelle auch angemerkt, dass zahlreiche Publikationen existieren, die sich intensiv mit der Performance von Datenbanksystemen auseinandersetzen. Dabei sind auch experimentelle Ergebnisse publiziert worden, die auch andere Datenbanksysteme betrachten und andere Benchmarks benutzen, als diejenigen aus den Textabschnitten 4.1 und 4.2. Hier sei insbesondere auf [THS10], [BBN+13], [INI+14] und [SH13] verwiesen.

Ein anderer Aspekt von Kapitel 4 wurde in den beiden Textabschnitten 4.1 und 4.2 nicht erwähnt. Um vergleichbare, objektiv gewonnenen Metrikwerte zu gewinnen, bedeuten Experimente immer einen gewissen Aufwand in puncto Zeit und technischer Ausstattung. Das betrifft nicht nur die Bemühungen in der Konfiguration des Versuchsaufbaus, sondern auch bei der Konfiguration und Durchführung der Benchmarks. Dieser Aufwand wird um so größer, je mehr Einflussfaktoren im Rahmen eines Experimentes untersucht werden sollen. Das zeigt sich zum Beispiel an der Anzahl an Testserien, die bei den Experimenten aus Kapitel 4 durchgeführt wurden.

Es ist offensichtlich, dass nach Möglichkeiten gesucht wird, den erwähnten Aufwand an Zeit und technischer Ausstattung zu minimieren, ohne die Semantik der Experimente zu ändern. Das nachfolgende Kapitel 5 zeigt eine solche Möglichkeit. Sie beschreibt die Herangehensweise, wie die drei Metriken Performance, Energieverbrauch und Energieeffizienz von Datenbanksystemen durch Simulation evaluiert werden können anstatt aufwendige Experimente durchführen zu müssen.

5 Modellierung von Datenbanksystemen

Die vorherigen Kapitel 3 und 4 zeigen, dass die Performance, der Energieverbrauch und damit die Energieeffizienz eines Datenbanksystems durch viele Faktoren beeinflusst wird. Diese Faktoren können in drei Kategorien eingeteilt werden:

1. **Hardware-technische Faktoren** Darunter werden alle Faktoren verstanden, die sich primär auf den Energieverbrauch der genutzten Hardware auswirken. In erster Linie beziehen sich diese Faktoren auf die Auswahl der technischen Komponenten beziehungsweise deren Kerntechnologien. Zusätzlich kommt noch die Auswahl der Architektur hinzu, dass heisst die Zusammenstellung der verwendeten Komponenten. Die Auswahl der Architektur begrenzt die Auswahl der Komponenten, da die eingesetzten Technologien und Übertragungswege die wahlfreie Kombination nicht zulassen.

   Zusätzlich fallen unter diese Kategorie auch alle Faktoren hinsichtlich der Charakterisierung eines Datenbanksystems. Das bedeutet genauer, dass zum Beispiel bei verteilten Datenbanksystemen in Form eines Clusters auch die Auswahl der hardware-technischen Komponenten zur Vernetzung der beteiligten Clusterknoten berücksichtigt werden muss.

2. **Software-technische Faktoren** Diese Faktoren umfassen sämtliche Aspekte, die sich mittels Computersoftware auf die Performance und teilweise auch auf den Energieverbrauch auswirken. Generell können diese Faktoren in zwei Unterkategorien geteilt werden: die erste Unterkategorie beinhaltet alle Konfigurationsmöglichkeiten, die durch das Betriebssystem bereitgestellt werden und Einfluss auf die eingesetzte hardware-technische Architektur haben. Im wesentlichen sind das spezielle Einstellungen der Treiber für die eingesetzte Hardware, die zum Teil große Auswirkungen auf die Performance und den Energieverbrauch haben. Dazu gehören zum Beispiel Einstellungen für Energiesparmodi. Zudem umfasst diese Unterkategorie auch zentrale Konfigurationsmöglichkeiten des eingesetzten Betriebssystems. Darunter fallen zum Beispiel die Wahl des Prozess-Schedulers oder die Zuteilung von verwalteten Betriebsmitteln wie etwa Arbeitsspeicher.

   Die zweite Unterkategorie betrifft das eingesetzte Datenbanksystem an sich. Datenbanksysteme ermöglichen es durch eigene Konfigurationsparameter, sich flexibel an den Einsatzzweck, spezielle Anwendungsszenarien und die Allokation von Betriebsmitteln des umgebenen Betriebssystem anzupassen. Allerdings ist auch die Wahl des Datenbanksystems von Bedeutung, da es Datenbanksysteme gibt, die Daten- und Speichermodelle implementieren, welche sich besonders zur Unterstützung eines Anwendungsfalles eignen.

3. **Organisatorische Faktoren** Diese Faktoren haben im Vergleich mit den beiden vorherigen Kategorien einen eher begrenzten Einfluss. Unter diesen Faktoren werden all jene verstanden, die einen organisatorischen Charakter für die hardware- und softwaretechnischen Faktoren haben. Das sind zum Beispiel die Investitionskosten für den Erwerb und den Betrieb der Hardware beziehungsweise des Datenbanksystems, laufende Kosten für die Kühlung und den Betrieb der eingesetzten Server oder räumliche Faktoren, etwa das Platzangebot für die notwendige Server- und Netzwerktechnik.

Dadurch entstehen zwei Problemfelder, die zum Teil erhebliche Konsequenzen auf Performance, Energieverbrauch und -effizienz haben:

- Alle drei oben genannten Kategorien bieten eine fast unüberschaubare Vielfalt an Zusammenstellungsmöglichkeiten. Eine einfach erscheinende Zusammenstellung ist die Auswahl eines Datenbanksystems, eines dafür geeeigneten Betriebssystems und eine hardware-technische Plattform für ein definierten Anwendungsfall. Dabei muss aber bedacht werden, dass man mittlerweile aus über 150 bekannten Datenbanksystemen, mehreren Betriebssystemen und einer immensen Auswahl an technischen Komponenten wählen kann.
- Die Faktoren, die in den oben genannten Kategorien genannt wurden, beeinflussen sich gegenseitig. Dabei ist es zum Teil sehr schwierig, einen kausalen Zusammenhang herzustellen. Solche Zusammenhängen erscheinen trivial, zum Beispiel dass der Austausch von magnetischen Festplatten durch SSDs eine Erhöhung der gesamten Performance durch höheren Datendurchsatz und kürzere Zugriffszeiten nach sich ziehen kann. Das muss aber experimentell belegt werden, wobei sich auch gezeigt hat, dass der Performancegewinn durchaus nur marginal ausfallen kann.

5.1 Darstellung der Energieeffizienz eines Datenbanksystems als Optimierungsproblem $\mathcal{P}_{EE}$

Die oben genannten drei Kategorien und die daraus resultierenden zwei Problemfelder sollen insgesamt verdeutlichen, dass die Evaluation und Optimierung der Performance, des Energieverbrauches und -effizienz eines Datenbanksystems ein Optimierungsproblem darstellt, das multiple Eingabegrößen hat. Formal ist nach *Korte und Vygen* ein Optimierungsproblem als Quadrupel $\mathcal{P}$ definiert [KV08, S.414]:

$$\mathcal{P} = (I, s, m, g) \tag{5.1}$$

wobei:

- I ist die Menge der Instanzen von $\mathcal{P}$.
- s ist eine Funktion, die für ein gegebenes $x \in I$ die Menge der möglichen Lösungen liefert. Sei ferner
$$S = \bigcup_{x \in I} s(x)$$
die Menge aller möglichen Lösungen.

- m ist eine Maßfunktion, wobei m als $m : I \times S \to \mathbb{N}$ definiert ist. Für ein gegebenes $x \in I$ und die dazugehörige Lösung $y \in s(x)$ gibt der Wert der Maßfunktion $m(x, y)$ die Kosten beziehungsweise das Maß für die Instanz x und der Lösung y an. Ferner sei festgelegt, dass $m(x, y)$ immer definiert ist, sofern $y \in s(x)$.
- g ist das Optimierungsziel, wobei $g \in \{\min, \max\}$.

Ein populäres Beispiel ist das Problem des Handelsreisenden, der eine definierte Anzahl an Kunden besuchen will und die Reihenfolge der Besuche planen will. Eine Instanz I dieses Problem ist in Abbildung 5.1 als Graph dargestellt: der Handelsreisende möchte sieben Kunden besuchen (dargestellt als Knoten A bis F), wobei die Distanzen zwischen den Kunden bekannt sind (Annotationen an den Kanten zwischen den Knoten).

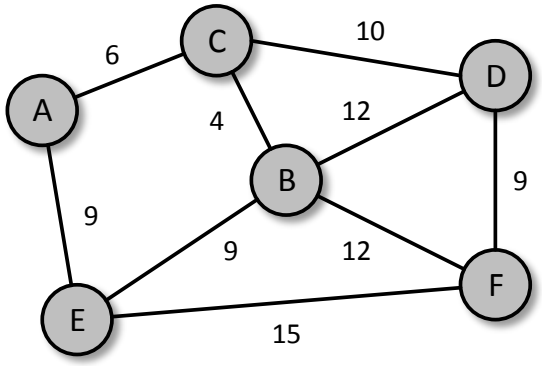


Abbildung 5.1: Problem des Handelsreisenden

In einer Variante des Problems geht man davon aus, dass sich die Kunden in einer euklidischen Ebene befinden und daher die Kostenfunktion m der euklidische Abstand ist. Die Funktion s gibt hierbei alle möglichen Kombinationen an, mit denen der Handelsreisende seine Kunden nacheinander besuchen kann, dass heisst, alle Permutationen der Knotenmenge $\{A, B, C, D, E, F\}$. Dem Handelsreisenden ist natürlich gelegen, die Reisedauer und -kosten für die Kundenbesuche so gering wie möglich zu gestalten, daher ist das Optimierungsziel $g = \{\min\}$[1].

Im Bezug auf die Optimierung der Energieeffizienz von Datenbanksystemen sind die Komponenten des Quadrupels $\mathcal{P}$ aus Gleichung 5.1 semantisch auf eine andere Weise belegt. Zur Abgrenzung sei die Optimierung der Energieeffizienz eines Datenbanksystems als $\mathcal{P}_{EE}$ bezeichnet:

$$\mathcal{P}_{EE} = (I, s, m, g = \{\max\}) \tag{5.2}$$

Aus Gleichung 3.3 von Seite 64 ist bekannt, dass sich die Energieeffizienz eines Datenbanksystems aus der Division der mittleren Performance und der mittleren elektrischen Leistung errechnen lässt. Daher müssen alle Instanzen $I \in \mathcal{P}_{EE}$ mindestens alle essentielle Komponenten enthalten, von denen sichergestellt ist, dass sie Einfluss auf die Performance und den Energieverbrauch nehmen. Die sind im wesentlichen alle Einflussfaktoren, die in den Textabschnitten 3.1 und 3.2 beschrieben sowie durch die experimentellen Ergebnisse von Kapitel 4 belegt sind. Allerdings

[1] Die Lösung des Problems des Handelsreisenden ist nicht Gegenstand dieser Dissertation. Daher muss auf externe Literatur verwiesen werden, zum Beispiel die ausführliche Betrachtung der Lösungen durch *Korte und Vygen* in [KV08, S.567 ff.]

muss auch beachtet werden, dass die Zusammensetzung der Komponenten von I nicht endlich und nicht fest ist [Zib16]. Der Grund dafür sind die am Kapitelanfang beschrieben Problematiken in der Evaluation der Performance und des Energieverbrauches eines Datenbanksystemes.

Am Beispiel der Experimente B bis D von Seite 78 ff. kann dennoch eine Zusammenstellung der Komponenten von I vorgenommen werden, wobei bei diesen speziellen Fall diese Zusammenstellung nur eine Teilmenge $I_{exp} \subset I$ aus Gleichung 5.2 darstellt:

$$I_{exp} \subset I = \{SIZE, FRAC, CPU, MEM, HDD, MAIN, SCHED\}$$

wobei

- die Menge *SIZE* die Größe der Datenbank der Experimente B bis D angibt, also $SIZE = \{1, 5, 10\}$.
- die Menge *FRAC* die Größe des zugeteilten Arbeitsspeicher für das Datenbanksystem angibt, also $FRAC = \{1, 2.5, 5\}$.
- die Mengen *CPU*, *MEM*, *HDD*, *MAIN* die möglichen technischen Komponenten (CPU, Hauptspeicher, Massenspeicher und Mainboard) angeben. Da sich in den Experimenten B bis D diese Auswahl nicht geändert hat, können diese Mengen als konstant mit einem Element angenommen werden. Diese Elemente ergeben sich aus der Beschreibung in Tabelle 4.1 auf Seite 74.
- die Menge *SCHED* die genutzten Prozess-Scheduler und Zugriffsmethoden auf das Datenbanksystem angeben. Das bedeutet im genaueren, dass $SCHED = \{\text{Performance}, \text{EnergySaving}\} \times \{\text{singleConnection}, \text{multipleConnection}\}$ ist.

Ein weitaus schwierige Aufgabe ist es, für das Quadrupel $\mathcal{P}_{EE}$ die Maßfunktion m explizit zu definieren. Eine Literaturrecherche ergab, dass es zum Zeitpunkt dieser Dissertation keine solche Definition existiert. Prinzipiell ist eine solche Definition realisierbar, aber aufgrund der unspezifischen Definition der Instanzmenge I sowie den beschriebenen Problematiken schwer zu generalisieren. Dafür muss eine Maß- beziehungsweise Kostenfunktion m erarbeitet werden, die alle Einfussfaktoren auf die Performance und den Energieverbrauch für alle Datenbanksysteme sowie deren Wechselwirkungen in Betracht zieht und dabei möglichst realitätsnah gestaltet sein muss. Da man auch annehmen kann, dass die Menge dieser Einflussfaktoren nicht fix ist und durch weitere Forschungstätigkeit erweitert werden wird, muss die Maßfunktion m zudem erweiterbar sein. Wie bereits erwähnt ist dieses Vorhaben realisierbar, aber zieht bedeutenden Aufwand nach sich.

Die letzte Komponente in der Definition des Quadrupels $\mathcal{P}_{EE}$ sagt aus, dass eine Optimierung der Energieeffizienz von Datenbanksystemen in dessen Maximierung resultieren soll ($g = \{\max\}$). Hierbei muss aber nach [Zib16] beachtet werden, dass es dabei durchaus verschiedene Herangehensweisen gibt. Wie in Textabschnitt 3.3 auf Seite 67 beschrieben, werden dabei zwei gänzlich unterschiedliche Standpunkte diskutiert. Der erste Standpunkt vertritt die Auffassung, dass die Optimierung der Performance und der Energieeffizienz zwei verschiedene Optimierungsziele sind. Dabei wird versucht, einen Kompromiss zwischen der performantesten und energiesparendsten Konfiguration eines Datenbanksystems zu finden, wobei durchaus Wechselwirkungen zwischen beiden auftreten können. Der zweite Standpunkt sagt aus, dass man nur einen minimalen Cluster eines Datenbanksystems (einen Cluster mit nur einem Knoten) untersuchen muss, wobei die energieeffizienteste Konfiguration ermittelt wird. Man geht dann davon aus, dass sie auch die energieeffizienteste sein wird, wenn der Cluster mit mehr als einem Knoten betrieben

wird. Hierbei ist nicht geklärt, welcher der beiden Standpunkte letztendlich mehr Erfolg im Bezug auf das Optimierungsproblem $\mathcal{P}_{EE}$ bietet. Betrachtet man zusätzlich Gleichung 3.3 auf Seite 64 zur Berechnung der Energieeffizienz eines Datenbanksystems, so ist erkennbar, dass je nach Anwendergruppe unterschiedliche Interessen verfolgt werden [Zib16]:

- Endanwendern von Datenbanksystemen fokussieren eher die Performance im Sinne der Antwortzeiten für ihre Datenbankanwendungen. Das kann dazu führen, dass die Energieeffizienz des eingesetzten Datenbanksystems steigt.
- Betreiber von Datenbanksystemen fokussieren aus wirtschaftlichen Gesichtspunkten eher die Performance im Sinne des Durchsatzes. Auch das kann die Energieeffizienz erhöhen.
- Technische Betreiber von Datenbanksystemen ist sowohl daran gelegen, die Performance allgemein zu steigern als auch den Energieverbrauch dabei zu senken. Damit fokussiert diese Anwendergruppe generell die Optimierung der Energieeffizienz. Dabei ist aber nicht geklärt, welche dieser beiden Aspekte die höhere Priorität geniesst.

Zusammengefasst sollen alle bisher gemachten Aussagen ausdrücken, dass eine Lösung des Optimierungsproblems $\mathcal{P}_{EE}$ formal sehr schwierig ist. Das gilt nicht nur für die Erarbeitung der Komponenten von $\mathcal{P}_{EE}$ (unpräzise Instanzmenge I, komplexe Kostenfunktion m) sondern auch, dass insbesondere für letztere Kostenfunktion m viele Aspekte berücksichtigt werden müssen, etwa die Wechselwirkungen der in I enthaltenen Einflussfaktoren oder Priorisierung einzelner Gesichtspunkte je nach Anwendergruppe.

5.1.1 Lösungsmöglichkeiten für das Optimierungsproblem $\mathcal{P}_{EE}$

Nach *Casale* sowie *Korte und Vygen* existieren zwei allgemeine Lösungsmöglichkeiten für das Optimierungsproblem $\mathcal{P}_{EE}$ [Cas16, KV08]:

1. **Analytische Lösung** Diese Lösungsmöglichkeit besteht darin, analytische Verfahren einzusetzen. Das bedeutet, es werden grundsätzlich mathematische Algorithmen eingesetzt. Diese Lösungsmöglichkeit hat den Vorteil, dass optimale Lösungen für viele Optimierungsprobleme bereits existieren und durch spezielle Computersoftware, zum Beispiel *Mathematica* oder *MathLab*, unterstützt werden. Aufgrund der Vielzahl an mathematischen Verfahren zur Lösung eines Optimierungsproblems sei auf die Auflistung und Beschreibung von *Korte und Vygen* in [KV08] verwiesen.
   Wie bereits erwähnt, gestaltet sich die analytische Lösung des Optimierungsproblems $\mathcal{P}_{EE}$ schwierig. Die Gründe dafür sind, dass einerseits eine durchaus komplexe Gleichung erarbeitet werden muss und andererseits durch Experimente gesichert wird, dass sie realitätsnah ist [DCPU11]. Die Erfahrungen von zum Beispiel *Gregory und Majumdar* sowie *Casale* zeigen in ihren Publikationen die Schwierigkeiten auf, die Energieeffizienz eines *Hadoop*-Clusters beziehungsweise des In-Memory-Datenbanksystems *SAP HANA* zu modellieren [GM16, Cas16].
2. **Experimentelle Lösung** Diese Lösungsmöglichkeit nutzt Verfahren aus der Verhaltensforschung, um experimentell die Einflussfaktoren eines Systems sowie deren Wechselwirkungen, zum Beispiel eines Datenbanksystems, zu bestimmen. Die experimentellen Resultate und Beobachtungen münden schließlich in einem Modell, um das System hinreichend realistisch zu beschreiben. Das Vorgehen im Bezug auf Datenbanksysteme wurde bereits in Textabschnitt 1.2 auf Seite 6 beschrieben.

Der Vorteil der experimentellen Lösung liegt darin, dass durch die experimentell gewonnenen Ergebnisse, Beobachtungen und Erfahrungen existierende Datenbanksysteme sukzessiv hinsichtlich der Energieeffizienz optimiert werden können. Der Nachteil besteht allerdings darin, dass diese Experimente einen nicht zu unterschätzende Aufwand in Zeit bedeuten. Zudem wird das Forschungsthema Energieeffizienz von Datenbanken durch die Durchführung von Experimenten nur explorativ vorangetrieben.

Es sei in diesem Zusammenhang besonders auf die Publikation von *Nicola und Jarke* hingewiesen. Sie haben in [NJ00] sowohl analytische als auch experimentelle Möglichkeiten zur Beschreibung und Evaluation von verteilten Datenbanksystemen zusammengetragen, erläutert und Vor- und Nachteile diskutiert.

Bei beiden Lösungsmöglichkeiten ist auffallend, dass immer ein Modell erarbeitet und genutzt wird, um die Energieeffizienz von Datenbanksystemen zu evaluieren und zu optimieren. Dabei impliziert Gleichung 3.3 zur Berechnung der Energieeffizienz eines Datenbanksystems auf Seite 64, dass dessen Modellierung alle Aspekte der Performance und des Energieverbrauches beinhalten muss. Genauer gesagt, die Modellierung muss qualitativ und quantitativ hinreichend genau sein. Dazu muss eine geeignete Modellierung sowohl alle Faktoren, die die Performance und den Energieverbrauch eines Datenbanksystems beeinflussen, abbildet werden (Quantität). Simulationen beziehungsweise Berechnungen auf Basis der Modellierung müssen die Größe der einzelnen Einflüsse präzise wiedergeben (Qualität).

Insgesamt ergeben sich drei Anforderungen, die eine geeignete Modellierung erfüllen muss:

1. Die Modellierung muss es ermöglichen, die Architektur, das Daten- und Speichermodell sowie alle spezifischen Charakteristiken eines jeden Datenbanksystems widerzugeben, das der Definition 2 auf Seite 38 genügt und Einfluss auf die Performance nehmen kann. Dazu zählt zum Beispiel die Nutzung von Zwischenspeicher, Caches und Datenbankindizes.
2. Die Modellierung muss die Verarbeitung von beliebigen Daten innerhalb dieser Datenbanksysteme gestatten. Damit kann die Ausführung von beliebigen Datenbankanwendungen realisiert werden, insbesondere die Ausführung eines Datenbanksystem-Benchmarks aus Gründen der Vergleichbarkeit von Performancewerten. Das schränkt die Daten auf eine Struktur ein, wie sie zum Beispiel vom Datengenerator des Benchmarks erzeugt wird.
3. Zugleich muss die Modellierung auch alle Komponenten der technischen Plattform beinhalten und ihren Einfluss auf die Performance abbilden, wenn eine Datenbankanwendung im Sinne des vorherigen Punktes simuliert wird.

5.1.2 Modellierung mittels Datenflüsse

Bei genauerer Analyse der drei Anforderungen fällt auf, dass die Modellierung der Energieeffizienz eines Datenbanksystems im wesentlichen auf die Modellierung der Performance von Datenbanksystemen zurückgeführt werden kann. Hierfür sind zwei Methoden bekannt:

1. **Systemsimulation** Bei dieser Methode wird ein komplettes System, zum Beispiel ein Server, technisch simuliert. Der dabei eingesetzte Simulator bietet die Möglichkeit, eine *von-Neumann*-Architektur bis auf Instruktionsebene zu simulieren [Neu93, YL06]. Das bedeutet, dass minimal eine CPU sowie CPU-Cache und Arbeitsspeicher simuliert werden. Allerdings bieten moderne System-Simulatoren mehr Möglichkeiten, unter anderem Massenspeicher und zusätzliche externe Geräte wie etwa eine Grafikkarte [RHWG95, MCE+02].

Der Hauptunterschied der Systemsimulation gegenüber Emulation und Virtualisierung besteht darin, dass die eigentliche Software durch einen zusätzlichen Schritt modelliert werden muss, um daraus Instruktionen zu generieren[23]. Bei der Emulation und Virtualisierung kann die eigentliche Software ohne Änderung ausgeführt werden.
Der Vorteil der Systemsimulation ist, dass äußerst exakte Performancemetriken aus den Simulationen gewonnen werden können, da wie erwähnt bis auf Instruktionsebene simuliert wird. Voraussetzung im Bezug auf Datenbanksysteme ist natürlich, dass diese für einen geeigneten Simulator vorbereitet wurden. Das bedeutet, dass der Binärcode eines Datenbanksystems soweit abstrahiert und generalisiert wird, dass er mittels Systemsimulation simuliert werden kann. Das dieses Vorgehen realistisch ist, zeigen diverse Publikationen, zum Beispiel *Lo et al.* oder *Bakhoda et al.* [LBE+98, BYF+09].
Der Nachteil der Systemsimulation dagegen ist, dass existierende Simulatoren nur in der Lage sind, ein Einzelplatzsystem zu simulieren. Somit ist es bis dato nicht möglich, einen Cluster und damit verteilte Datenbanksysteme zum Zwecke der Performanceanalyse zu simulieren. Somit eignen sie sich nur zur Simulation von zentralen Datenbanksystemen.

2. **Systemanalyse** Bei dieser Methode wird ein gegebenes System analysiert und versucht, zunächst Kernkomponenten zu identifizieren. Danach werden die Wechselwirkungen sowohl zwischen diesen Kernkomponenten als auch mit der äußeren Umgebung getestet und Bestrebungen unternommen, sie formal zu beschreiben. Eine solche Systemanalyse im Bezug auf Datenbanksysteme ist bereits in Textabschnitt 2.3 auf Seite 24 durchgeführt worden, wobei die Kernkomponenten identifiziert worden sind. Die äußere Umgebung kann hierbei als das umgebende Betriebssystem sowie die technischen Komponenten verstanden werden.
   Der Vorteil der Systemanalyse von Datenbanksysteme ist, dass sie durch bewährte formale und grafische Methoden in verschiedenen Aspekten beschrieben werden können. Beispiele sind dabei Datenfluss-, Zustandsübergangs- oder *entity-relationsship*-Diagramme. Somit ist es möglich, die Verarbeitung und Verteilung von Datenströmen sowie das zeitliche Verhalten der Kernkomponenten eines Datenbanksystems zu beschreiben. Das kann dazu genutzt werden, die Performance eines Datenbanksystems zu optimieren, wobei es auch möglich ist, sie zu simulieren.

Insbesondere die Analyse und Modellierung der Datenflüsse innerhalb eines Datenbanksystems birgt ein großes Potential, die oben genannten Anforderungen zu erfüllen. Der Hauptgrund ist, dass die Datenflussmodellierung fundamental für alle Arten von Computersoftware gültig ist, die Daten in gleicher Weise verarbeiten. Das trifft für alle Datenbanksysteme zu, sofern sie der Definition 2 auf Seite 38 genügen. Das bedeutet, sie müssen die gleiche grundlegenden Komponenten aufweisen, wenn man ihre Architektur abstrahiert. Eine solche Abstraktion ist bereits in Textabschnitt 2.3 auf Seite 24 beschrieben, in der vier grundlegende Komponenten identifiziert wurden, die alle Datenbanksysteme gemäß Definition 2 besitzen. Ein Datenflussmodell, das den Datenfluss zwischen diesen vier grundlegenden Komponenten nachbilden kann, ist in der Lage, auch die Verarbeitung dieser Datenflüsse durch diese vier Komponenten abzubilden[4]. Sobald die Verarbeitung der Datenflüsse durch ein Datenflussmodell repräsentiert werden kann, ist

[2] Eine Emulation versucht, eine komplette Umgebung zu schaffen, in der eine geeignete Software ausgeführt werden kann. Beispiele hierfür sind Emulatoren, die kaum mehr genutzte Heimcomputer nachstellen können, etwa einen *Amiga 500* oder einen *Sinclair ZX Spectrum*.

[3] Bei der Virtualisierung wird versucht, technische Komponenten nachzubilden. Im optimalen Fall erkennt ein Betriebssystem nicht, ob es auf realen oder virtuellen Komponenten ausgeführt wird. Dementsprechend kann auch Software, die das Betriebssystem nutzt, ohne Änderungen in der virtualisierten Umgebung ausgeführt werden.

[4] Die Verarbeitungsschritte durch die vier Kernkomponenten eines Datenbanksystems sind grafisch in Abbildung 2.6 auf Seite 26 dargestellt.

auch die Simulation der Performance möglich, womit auch die Möglichkeit eröffnet wird, durch Simulationen Vorhersagen zu treffen.

Dabei muss beachtet werden, dass ein solches Datenflussmodell nicht mit einem Datenflussdiagramm verwechselt werden darf. Letzteres stellt nur eine Übersicht dar, aus welchen Datenquellen Daten bezogen, durch Funktionen verarbeitet und an Datensenken gesendet werden. Ein Datenflussmodell beinhaltet zusätzlich Kontrollstrukturen bei der Verarbeitung der Datenflüsse und erlaubt auch die Betrachtung differenzierter Datenstrukturen, die bei den Datenflüssen zum Einsatz kommen. Oft genutzte Datenflussmodelle sind zum Beispiel Warteschlangen-Netzwerke (sogenannte *queue networks*) und *Petri*-Netze. Es hat sich gezeigt, dass diese beiden Datenflussmodelle insbesondere dann genutzt werden, um die Performance abzubilden und zu simulieren. So haben zum Beispiel *Bontempi und Kruijtzer*, *Kounev et al.* sowie *Nambiar und Poess* gezeigt, dass die Vorhersage der Performancewerte präzise simuliert werden kann [BK02b, Kou08, Nam11, AS11, DV13].

Zudem bieten Datenflussmodelle weitere vorteilhafte Eigenschaften. Wie bereits beschrieben erlauben Datenflussmodelle, differenzierte Datenstrukturen abzubilden und für die Datenflüsse zu nutzen. Das wird zumeist dadurch erreicht, dass die Datenstrukturen in Form von Marken symbolisiert werden. In einem theoretischen Datenflussmodell ist zunächst auch die Anzahl der Marken unerheblich. Das ist für die zweite, oben genannte Anforderung im Bezug auf die Performancemodellierung von Datenbanksysteme wichtig. Dazu stelle man sich vor, dass die Eingabedaten eines Datenflussmodells für ein Datenbanksystem von dem Datengenerator eines Benchmarks stammen. Die dabei genutzten Datenstrukturen können in Form von Marken symboliert werden. Je nachdem, wieviele Daten der Datengenerator erzeugen soll, kann die Anzahl dieser Marken mitskalieren. Somit ergibt sich die Möglichkeit, die simulierten Performancewerte auf Basis des Datenflussmodelles mit realen, experimentell gewonnenen Werten direkt vergleichen zu können. Prinzipiell können auf diese Weise Simulationen der Benchmarks ausgeführt werden, die dieselben Performancemetriken nutzen. Das sind im wesentlichen die Antwortzeit oder der Durchsatz.

Generell erlaubt die Datenflussmodellierung auch die Berücksichtigung anderer Metriken, sofern sie den Datenfluss beeinflussen. So haben zum Beispiel *Dietrich et al.* gezeigt, dass es möglich ist, auch hardware-technische Komponenten in den Datenfluss einzubeziehen [DBCrW92]. Dieser Aspekt ist zur Erfüllung der dritten, oben genannten Anforderung wichtig. Es bedeutet auch, dass man anhand des Datenflusses Rückschlüsse auf die Auslastung dieser technischen Komponenten ziehen kann. Verknüpft man zum Beispiel die simulierte mittlere Auslastung einer technischen Komponente mit einer Metrik zum Energieverbrauch, zum Beispiel der mittleren elektrischen Leistung, so kann der Energieverbrauch errechnet werden. Summiert man diese errechneten Energieverbräuche für alle simulierten technischen Komponenten, so erhält man den Gesamtenergieverbrauch der simulierten technischen Plattform.

Zusammengefasst bedeutet es: ein Datenflussmodell, dessen Komponenten sowohl die Architektur eines Datenbanksystems als auch alle notwendigen technischen Voraussetzungen abstrahieren kann, ist in der Lage, den Datenfluss zu simulieren, der durch einen Benchmark für ein Datenbanksystems definiert wird. Mit einer zusätzlichen Metrik wie dem simulierten mittleren Energieverbrauch pro Modellkomponente ist es möglich, den Gesamtenergieverbrauch zu simulieren. Zusammen mit den Simulationsergebnissen hinsichtlich der Performancemetriken wie Antwortzeit oder Durchsatz ist damit die Simulation der Energieeffizienz laut Gleichung 3.3 auf Seite 64 realistisch.

5.2 Erweiterungen von *Queued Petri Nets*

Wie im vorhergehenden Textabschnitt beschrieben, können Datenflussmodelle dafür eingesetzt werden, um Komponenten, die Datenflüsse zwischen ihnen und die Verarbeitungsschritte innerhalb eines Computerprogrammes zu beschreiben. Ein solches Datenflussmodell kann dann für Simulationen der Performance für beliebige Eingabedaten genutzt werden. Im Bezug auf die Performancemodellierung von Datenbanksystemen wurden diese Eigenschaften genutzt, wobei *Queued Petri Nets* (kurz QPN) verwendet wurden. Dabei sei an dieser Stelle für eine grundlegende Einführung in QPNs und *Petri*-Netzen im allgemeinen auf Textabschnitt 2.5 hingewiesen.

Bevor die eigentlichen QPN-Modelle in den nachfolgenden Textabschnitten erläutert werden, soll in diesem Textabschnitt drei Erweiterungen erläutert werden, die im Zusammenhang mit den QPN-Modellen von Bedeutung sind. Sie haben das Ziel, die Modellierung und Simulation von QPNs komfortabler gestalten, wobei deren fundamentale formale Definition eingehalten wird.

5.2.1 Marken und Submarken

Die erste Erweiterung ist die Einführung des Konzeptes von Submarken im Rahmen dieser Dissertation. Dieses Konzept sieht vor, dass (farbige) Marken auch eine klar abgegrenzte Menge von sogenannten (farbigen) Submarken repräsentieren können. In diesem Fall können die Marken als eine Art Container für die Submarken angesehen werden. Die Hauptgünde für die Einführung des Konzeptes sind:

1. die Repräsentation von strukturierten Daten, wie sie zum Beispiel der Datengenerator eines Benchmarks generieren kann
2. die Verarbeitung dieser Daten mit Hilfe von QPN-Modellen möglichst realitätsgetreu zu simulieren.

Zur Motivation kann sich ein Datengenerator eines Benchmarks für Datenbanksysteme vorgestellt werden. Er generiert Daten gemäß des relationalen Datenmodelles. Das bedeutet, er generiert eine endliche Menge an Datensätzen, wobei diese Datensätze dieselbe Struktur aufweisen, also immer dieselbe Anzahl an Feldern. Es sei angemerkt, dass der eigentliche Inhalt der generierten Datensätze nicht von Bedeutung ist. Sei n die Anzahl der generierte Datensätze und m die Anzahl der Felder pro Datensätz. Dann können die generierten Datensätze als Menge D aufgefasst werden, die ingesamt $n \cdot m$ Elemente umfasst:

$$D = \{\underbrace{d_1, d_2, d_3, \ldots, d_m}_{m \text{ Elemente}}, \underbrace{d_{m+1}, d_{m+2}, d_{m+3}, \ldots, d_{2 \cdot m}}_{m \text{ Elemente}}, \ldots, d_{n \cdot m}\}$$

Um die generierten Datensätze in Form von D nun innerhalb von QPN-Modellen zu repräsentieren, müssen die Elemente von D als farbige Marken aufgefasst werden können. Sei dafür C die Menge an Farben und bezeichne $C_{\mathfrak{M}}$ die Multimenge von C. Die Menge C enthalte genau $m+1$ Elemente:

$$C = \{\underbrace{c_1, c_2, \ldots, c_m}_{\text{Submarken}}, \underbrace{c_{m+1}}_{\text{Datenmarke}}\}$$

Dabei repräsentieren die Elemente c_1 bis c_m die jeweiligen Felder eines generierten und das Element c_{m+1} einen Datensatz. Die obige Menge D lässt sich damit als Multimenge $D_s \subset C_{\mathfrak{M}}$ darstellen:

$$D_S = \{\underbrace{c_1, c_2, c_3, \ldots, c_m}_{m \text{ Elemente}}, \underbrace{c_1, c_2, c_3, \ldots, c_m}_{m \text{ Elemente}}, \ldots, c_m\}$$

Anhand von Multimenge D_s wird also die Struktur der generierten Datensätze abgebildet. Die verbliebene Farbe c_{m+1} soll wie bereits beschrieben einen ganzen Datensatz symbolisieren. Dafür kann eine Multimenge $D_M \subset C_{\mathfrak{M}}$ genutzt werden, die genau n Elemente besitzt:

$$D_M = \{\underbrace{c_{m+1}, c_{m+1}, c_{m+1}, c_{m+1}, c_{m+1}, \ldots, c_{m+1}}_{n \text{ Elemente}}\}$$

Die Multimenge D_M repräsentiert also die generierten Datensätze, wobei der Inhalt und die Struktur nicht von Belang sind.

Für die weitere Betrachtung sei die Menge an Farben c_1 bis c_m als *Submarken* und die Farbe c_{m+1} als *Datenmarke* bezeichnet. Im Bezug auf das oben genannte Konzept bedeutet es, dass eine Datenmarke ein Symbol beziehungsweise ein Repräsentant für die Submarken ist. Semantisch ist also eine Datenmarke eine Art Container für die Submarken. Bezogen auf die Multimengen D_S und D_M sieht das Konzept also eine Lösung vor, wie aus der Multimenge D_S die Multimenge D_M entsteht (Submarken-zu-Marken-Transformation) und umgekehrt (Marken-zu-Submarken-Transformation). Zudem ist damit auch die strukturelle Repräsentation von Daten als farbige Marken und Submarken möglich.

Diese Transformationen können auch genutzt werden, um zum Beispiel statistische Auswertungen vornehmen zu können. Sei jetzt angenommen, dass eine Transformation der Submarken zu Datenmarken bereits vorgenommen wurde. Das bedeutet für obiges Beispiel, dass nun die Multimenge D_M vorliegt, dessen Elemente eine Anzahl n von generierten Datensätzen repräsentiert. Nun möchte man diese generierten Datensätze auf einem Massenspeicher speichern. Der Massenspeicher besitzt eine begrenzte Kapazität und kann nur Datenblöcke mit einer konstanten Größe verarbeiten. Um diesen Massenspeicher modellieren zu können, sei dieser als Multimenge $D_{disk} \subset C_{\mathfrak{M}}$ definiert. Dabei stellen die Elemente dieser Multimenge die Datenblöcke dar. Um diese Elemente von den bereits genutzten Multimengen D_S und D_M abgrenzen zu können, wird die Menge der Farben C um das Element c_{m+2} erweitert und festgelegt, dass Datenblöcke als Marken mit dieser Farbe symbolisiert werden. Diese zusätzliche Farbe c_{m+2} sei im folgenden als *Speichermarke* bezeichnet:

$$C = \{\underbrace{c_1, c_2, \ldots, c_m}_{\text{Submarken}}, \underbrace{c_{m+1}}_{\text{Datenmarke}}, \underbrace{c_{m+2}}_{\text{Speichermarke}}\}$$

In der Realität stellt sich hinsichtlich der generierten Datensätze die Frage, wieviele Datenblöcke des Massespeichers nötig sind, um alle Datensätze speichern zu können. Es ist offensichtlich, dass entsprechende Größenangaben notwendig sind. In den formalen Definitionen von D_S und D_M ist nur formuliert, wie generierte Datensätze strukturiert sind beziehungsweise wie innerhalb von QPN-Modellen als farbige Marken symbolisiert werden können. Um also die Speicherung der Marken aus D_M in D_{disk} nachzubilden zu können, muss zunächst festgelegt werden, welche farbige Marke wieviel Speicherplatz benötigen wird. Dafür wird die Abbildung L genutzt:

$$L : C \to \mathbb{N}^+$$

$\mathbb{N}^+$ bezeichnet die Menge an natürlichen Zahlen größer als Null. Abbildung L weist jeder Farbe aus C eine konstante, ganze und positive Zahl größer als Null zu. Durch diese Zuweisung

wird ausgedrückt, welche Menge an Speicherplatz im Haupt- oder Massenspeicher notwendig wäre, um das Objekt speichern zu können, das eine farbige Marke symbolisiert. Bezogen auf die Multimenge D_S, dessen Elemente Submarken sind, weist Abbildung L also diejenigen Speichergrößen zu, die zur Speicherung der Datenfelder notwendig wären. Dieses Vorgehen deckt sich auch mit der Realität: die Felder der generierten Datensätze genügen einem Datentypen, zum Beispiel einer ganzen Zahl (Datentyp `integer`) oder einer Zeichenkette mit fester Länge (Datentyp `char(n)`, wobei `n` die Länge angibt). Diese Datentypen belegen bei ihrer Speicherung eine konstante Menge an Speicherplatz und kann durch Abbildung L ausgedrückt werden.

Dementsprechend kann nun anhand der Abbildung L errechnet werden, wieviel Speicherplatz ein generierter Datensatz benötigen würde. Sei L_S der Speicherplatz für einen kompletten generierten Datensatz:

$$L_S = \sum L(c) \qquad (c \in C \setminus \{c_{m+1}, c_{m+2}\})$$

Da ein kompletter generierter Datensatz auch durch eine Datenmarke symbolisiert werden kann, ist $L(c_{m+1}) = L_S$. Sei zur besseren Erklärung die oben beschriebene Speichergröße eines Datenblockes des Massenspeichers mit L_{disk} bezeichnet. L_{disk} ergibt sich dabei aus $L(c_{m+2})$, also Speichergrößenzuweisung einer Speichermarke.

Nutzt man die ganzzahlige Division mit Aufrundung, so kann die Anzahl n_{disk} der Speichermarken errechnet werden, die benötigt werden, um die Datenmarken zu transformieren:

$$n_{disk} = \left\lceil \frac{n \cdot L_S}{L_{disk}} \right\rceil$$

Zudem kann auch errechnet werden, wieviele Datenmarken in nur eine Speichermarken transformiert werden können. Semantisch entspricht das der Speicherung von generierten Datensätzen in einen Datenblockes des Massenspeichers. Sei m_{disk} die Anzahl von generierten Datensätzen, die in einen Datenblock gespeichert werden können:

$$m_{disk} = \left\lfloor \frac{L_{disk}}{L_S} \right\rfloor \qquad (L_{disk} > L_S)$$

Nun kann man sich vorstellen, dass die Speicherung von generierten Datensätzen in einen in einen Datenblock des Massenspeichers eine gewisse Zeit in Anspruch nimmt. Sei dafür t_1 die Zeit, die es braucht, um einen Datensatz in einen vorbereiten Datenblock zu speichern. Sei als t_2 die Zeit bezeichnet, um den vorbereiteten Datenblock auf den Massenspeicher zu schreiben. Analog sei t_3 die Zeit, ihn wieder vom Massenspeicher zu lesen und t_4 die Zeit, einen Datensatz vom gelesen Datenblock auszulesen. Damit ist die Zeit t für die gesamte Transformation von Submarken zu Datenmarken zu Speichermarken und deren Rücktransformation:

$$t = t_1 \cdot (m_{disk} \cdot m \cdot n) + t_2 \cdot n_{disk} + t_3 \cdot n_{disk} + t_4 \cdot (m_{disk} \cdot m \cdot n)$$

Anhand der Zeiten t_1 bis t_4 sowie t können weitergehende Berechnungen durchgeführt werden, zum Beispiel die Berechnung des mittleren Durchsatz des Schreibens von Datenblöcken auf den Massenspeicher oder den Durchsatz der Transformation von Datenmarken zu Speichermarken und umgekehrt.

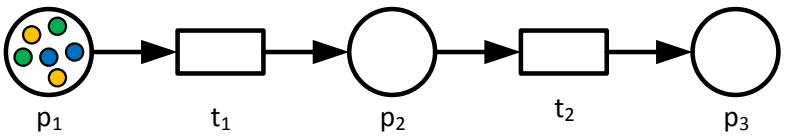
Zur abschließenden Illustration des Konzeptes sei nun ein Datengenerator gegeben, der $n = 2$ Datensätze generiert, wobei jeder Datensatz aus $m = 3$ Datenfeldern besteht. Sei C die Menge der Farben mit $C = \{c_1, c_2, c_3, c_4\}$. Sei ein QPN-Modell gegeben, wie es in den Abbildungen 5.2(a) bis 5.2(c) auf Seite 127 dargestellt ist. Dieses Modell besteht aus den Stellen p_1 bis p_3 sowie den beiden Transitionen t_1 und t_2, die die Stellen verbinden. Sei in der gesamten Abbildung

5.2 c_1 durch die Farbe grün, c_2 durch die Farbe gelb, c_3 durch die Farbe blau und c_4 durch die Farbe rot repräsentiert.

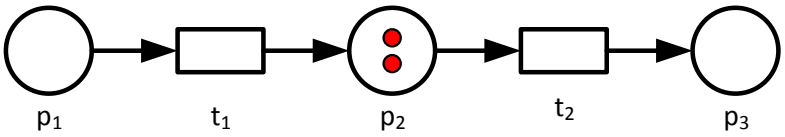
In Abbildung 5.2(a) ist die Markenbelegung des QPN-Modells nach der Generierung der Datensätze dargestellt. Die $m = 3$ Datenfelder der $n = 2$ Datensätze werden durch die Farben c_1 bis c_3 symbolisiert. Die generierten Datensätze werden initial in der Stelle p_1 abgelegt. Dementsprechend sind in Abbildung 5.2(a) $n \cdot m = 6$ farbige Marken zu erkennen. Sei der Einfachheit halber angenommen, dass $\forall c \in \{c_1, c_2, c_3\} : L(c) := 1$ und $L(c_4) := 3$. Durch eine Transitionsregel[5] $TR_1 = (\{c_1, c_2, c_3\}, \{c_4\}, t_1)$ der Transition t_1 wird die Transformation der farbigen Marken c_1 bis c_3, also eines Datensatzes, zu einer farbigen Marke c_4 durchgeführt. Somit sind erstgenannte farbige Marken nun Submarken der letztgenannten farbigen Marke. t_1 bezeichne dabei die Zeit, die es braucht, um diese Transformation durchzuführen.

In Abbildung 5.2(b) ist die Markenbelegung nach der Transformation dargestellt. Durch die Belegungen von L sowie durch die Transformationsregel TR_1 sind zwei farbige Marken c_4 entstanden, die in Stelle p_2 abgelegt werden. Durch eine Transitionsregel $TR_1 = (\{c_4\}, \{c_1, c_2, c_3\}, t_2)$, die in Transition t_2 wirksam ist, kann eine Rücktransformation vorgenommen werden. t_2 bezeichnet hierbei die Zeit, die eine solche Rücktransformation braucht.

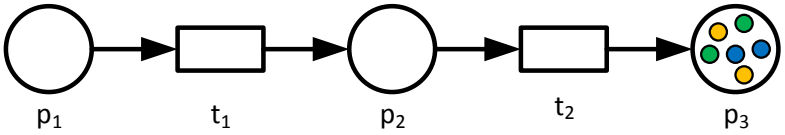
Das Ergebnis der Rücktransformation ist in Abbildung 5.2(c) als Markenbelegung des QPN-Modells dargestellt. Durch die Rücktransformation sind aus den farbigen Marken c_4 wieder die ursprünglichen Marken c_1 bis c_3 entstanden, die in Stelle p_3 abgelegt werden.



(a) Markenbelegung vor Transformation



(b) Markenbelegung nach Transformation



(c) Markenbelegung nach Rücktransformation

Abbildung 5.2: Darstellung der Transformation und Rücktransformation von Marken und Submarken in einem QPN-Modell

Insgesamt kann das Konzept von Marken und Submarken im Zusammenhang mit QPN-Modellen und Transitionsregeln dazu genutzt werden, um viele Zusammenhänge in Datenbanksystemen

[5] Transitionsregeln wird im weiteren Verlauf dieses Textabschnittes 5.2 erläutert, genauer gesagt in Textabschnitt 5.2.3.

exakt zu modellieren. Dazu gehört nicht nur die Repräsentation von Datenstrukturen wie sie etwa von einem Datengenerator erzeugt werden, sondern auch wie zum Beispiel die Datenstrukturen auf Massenspeichern gespeichert werden können. Dabei werden auch die Strukturumwandlung und die dafür nötige Zeit in Betracht gezogen. Diese Vorteile sind im Bezug auf die Simulation von QPN-Modellen, die ein komplettes Datenbanksystem nachstellen sollen, von großer Wichtigkeit.

5.2.2 Relation von Tupeln (Assoziativmarken)

Die zweite Erweiterung im Umgang mit QPN-Modellen, speziell bei deren Simulation, ist das Konzept von Assoziativmarken. Sie ermöglichen es, zwei Multimengen von farbigen Marken miteinander zu assoziieren.

Sei als einleitendes Beispiel n Datensätze gegeben. Die Datensätze sollen in Datenblöcke eines Massenspeichers, zum Beispiel einer magnetischen Festplatte, gespeichert werden, wobei festgelegt sei, dass m Datensätze in einem Datenblock gespeichert werden können. Vom Interesse ist nun, in *welchem* Datenblock ein Datensatz gespeichert wurde. Hierfür sei festgelegt, dass die Datenblöcke aufsteigend mit positiven, ganzen Zahlen eindeutig unterschieden werden können.

Zur Lösung seien die Datensätze als Tupel A mit n Komponenten $A = (a_1, a_2, a_3, \ldots, a_n)$ repräsentiert. Die Festlegung, welcher Datensatz in welchem Datenblock gespeichert werden soll, kann formal durch eine Relation R definiert werden:

$$R : A \to \vec{\mathbb{N}}_0, a \mapsto z$$

wobei $\mathbb{N}_0$ die Menge der natürlichen, positiven, ganzen Zahlen inklusive Null ist. Diese Relation ordnet jeder Komponente $a \in A$ somit eine Zahl $z \in \mathbb{N}_0$ zu. Formal bedeutet obige Definition der Relation R also:

$$R(a_1, a_2, a_3, \ldots, a_n) = (z_1, z_2, z_3, \ldots, z_n) \qquad (z \in \mathbb{N}_0)$$

Nutzt man zusätzlich den Index i der Komponenten aus A, so kann R mittels m und der ganzzahligen Division auch in anderer Form ausgedrückt werden:

$$R : A \to \vec{\mathbb{N}}_0, a_i \mapsto z := \left\lfloor \frac{i}{m} \right\rfloor \qquad (1 \leq i \leq n)$$

beziehungsweise

$$R(a_1, a_2, a_3, \ldots, a_n) = (\left\lfloor \frac{1}{m} \right\rfloor, \left\lfloor \frac{2}{m} \right\rfloor, \left\lfloor \frac{3}{m} \right\rfloor, \ldots, \left\lfloor \frac{n}{m} \right\rfloor)$$

Soll das einleitendes Beispiel in QPNs umgesetzt werden, so ergibt sich die Schwierigkeit, das Tupel A und die Relation R als farbige Marken zu repräsentieren. Zur Lösung kann das Tupel $A = (a_1, a_2, a_3, \ldots, a_n)$ einfach als Tupel mit n Komponenten aufgefasst werden, wobei jede Komponente als farbige Marke angesehen werden kann. Die dabei verwendeten Farben seien als Menge C bezeichnet. In diesem Fall umfasst C minimal ein und maximal n Elemente (= Farben)[6]. Im Bezug auf die Relation R sei eine zweite Menge an Farben C' bezeichnet, wobei C und C' disjunkt sind ($C \cup C' = \emptyset$). Die in Menge C' verzeichneten Elemente (= Farben) stellen dabei die verwendeten Datenblöcke dar. Das bedeutet, dass Elemente aus A nicht mehr

[6] Das resultiert aus der Definition ein Tupels. Die Komponenten eines Tupels müssen nicht notwendigerweise verschieden voneinander sein.

Elementen aus $\mathbb{N}_0$ gemäß Relation R zugeordnet werden, sondern Elementen aus C'. Dabei ist ersichtlich, dass Menge C' minimal ein Element (im Fall von $n = m$) und maximal n Elemente haben kann (im Fall von $m = 1$). Somit kann nun basierend auf Relation R eine Relation R' gebildet werden:

$$R' : A \to C', a_i \mapsto c_j, j := \left\lfloor \frac{i}{m} \right\rfloor \qquad (1 \leq i \leq n, 1 \leq j \leq \left\lceil \frac{n}{m} \right\rceil)$$

Wenn also $C = (c_1, c_2, c_3, \ldots, c_n)$ und $C' = (c'_1, c'_2, c'_3, \ldots, c'_m)$, dann kann Relation R' in folgender formalen Form angegeben werden:

$$R(a_1, a_2, a_3, \ldots, a_n) = (c'_{\lfloor \frac{1}{m} \rfloor}, c'_{\lfloor \frac{2}{m} \rfloor}, c'_{\lfloor \frac{3}{m} \rfloor}, \ldots, c'_{\lfloor \frac{n}{m} \rfloor})$$

Sei zur Illustration des eingangs erwähnten einleitenden Beispiels $n = 10$ und $m = 2$ festgelegt. Somit sind

$$\begin{aligned} A &= (a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}) \qquad a \in C \\ R &= (1, 1, 2, 2, 3, 3, 4, 4, 5, 5) \\ R' &= (r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}) \qquad r \in C' \end{aligned}$$

Betrachtet man A und R' hierbei, so erkennt man eine Bijektivität hinsichtlich der verwendeten Indizes. Die Farbe der dabei repräsentierten farbigen Marken spielt primär dabei keine Rolle. Das ist insbesondere für den QPN-Simulator *QPME* interessant, mit dem alle in Kapitel 5 vorgestellten QPN-Modellen simuliert wurden. Aus Textabschnitt 2.5 ist bekannt, dass farbige Marken im allgemeinen als Multimengen angegeben werden. Innerhalb des Simulators werden sie jedoch als Tupel verwendet, wobei auch die Indizes dafür verwendet werden können, die farbigen Marken zu identifizieren. Zudem bietet der verwendete Simulator auch die Möglichkeit, die Indizes auch für Transitionsregeln zu verwenden.

Insgesamt ermöglicht es das Konzept, das in diesem Textabschnitt vorgestellt wurde, eine Relation zwischen zwei Tupelmengen herzustellen, deren Komponenten farbige Marken repräsentieren, wobei die Indizes der Komponenten genutzt werden.

5.2.3 Transitionsregeln

Eine dritte Erweiterung von QPNs sind sogenannte *Transitionsregeln*, die Anwendung bei Transitionen mit Zeitverzug finden. Die formale Definition wurde von *Kounev und Dutz* im Rahmen der Beschreibung des QPN-Simulators *QPME* publiziert [KD09]. Bei der Implementierung des Simulators haben die beiden Autoren zwei Feststellungen gemacht:

1. Für Endanwender ist es schwierig, das zeitliche Verhalten eines Systems zu modellieren. Der Grund dafür ist, dass sie sich bei der Beschreibung mit Feuerwahrscheinlichkeiten, Wahrscheinlichkeitsverteilungen und mit Warteschlangentheorie auseinandersetzen müssen (siehe Definitionen 5 bis 7). All diese Aspekte sind zwar formal für die quantitative Analyse sowie für eine Simulation notwendig, aber die Mehrheit der Endanwender des Simulators forderte die Möglichkeit, diskrete numerische Verzögerungszeiten anzugeben.
   Dafür wird innerhalb des Simulators eine interne Zeiteinheit genutzt. Es bleibt dem Endanwender überlassen, wie er diese Zeiteinheit interpretiert, etwa 1 Sekunde oder 1 Stunde.

Wichtig ist nur, dass die Verzögerungszeiten durch den Endanwender angegeben werden können. Wird zum Beispiel durch den Endanwender bei einer zeitverzögerten Transition eine numerische Verzögerungszeit von 0.384 angegeben und die interne Zeiteinheit als Sekunde angenommen, so ergibt sich eine Verzögerungszeit von 384 Millisekunden. Analog bedeutet eine numerische Verzögerungszeit von 1.46 eine Verzögerungszeit von 1.46 Sekunden.
Innerhalb des Simulators werden die diskreten, numerisch angegebenen Verzögerungszeiten dann in die entsprechenden Feuerwahrscheinlichkeiten und Wahrscheinlichkeitsverteilungen umgewandelt, sodass alle notwendigen Formalismen und Definitionen bezüglich QPNs eingehalten werden.

2. Für Endanwender ist es schwierig, die Verarbeitung vieler Marken mit einer großen Anzahl an Farben mittels Transitionen zu modellieren. Wie bereits beschrieben, basieren QPNs auf CGSPNs, also farbigen generalisierten stochastischen Petri-Netzen (siehe Defintion 6 auf Seite 47 sowie Definition 5 auf Seite 44 für farbige Petri-Netze). Im Bezug auf die Verarbeitung farbiger Marken geben die beiden Abbildungen I^- und I^+ für eine Transition an, welche Menge an farbigen Marken von den Eingabestellen konsumiert und welche Multimenge an farbigen Marken in die Ausgabestellen abgelegt werden. Problematisch ist hierbei, dass Transitionen nur auf die Verarbeitung dieser beiden Mengen beschränkt sind. Es ist also nicht möglich, für Transitionen mehrere Eingabemengen und korrespondierende Ausgabemengen zu definieren. Daraus resultiert der Gebrauch multipler Transitionen, was das bereits beschriebene Problem des *state space explosion* verstärkt.

Beide Feststellungen finden Verwendung in den sogenannten Transitionsregeln. Dabei beschreibt eine Transitionsregel für eine Transition allgemein, welche Multimenge an farbigen Marken von den Eingabestellen konsumiert und welche Multimenge an farbigen Marken in die Ausgabestellen abgelegt werden sowie eine numerische, diskrete Verzögerungszeit im Sinne einer "Verarbeitungsdauer" der farbigen Marken. Formal ist also eine Transitionregel ein Tripel *TR*:

$$TR = (IMarks, OMarks, Delay)$$

Sei C die Menge aller genutzten Farben in einem QPN-Modell. Dann bezeichnet die Komponente $IMarks \in C_{\mathfrak{M}}$ die Multimenge an farbigen Marken, die aus den Eingabestellen konsumiert werden sollen. Analog bezeichnet $OMarks \in C_{\mathfrak{M}}$ die Multimenge an farbigen Marken, die in die Ausgabestellen abgelegt werden. Die letzte Komponente $Delay \in \mathbb{R}^+$ bezeichnet die positive diskrete Verzögerungszeit. Sie kann auch Null betragen, um auszusagen, dass es keine Verzögerung gibt.

Transitionsregeln lassen sich auch zu einer Liste zusammenfassen. Das bedeutet, dass eine solche Liste ein Tupel darstellt, dass prinzipiell aus einer unbegrenzten Anzahl von Komponenten besteht, wobei diese Komponenten die verschiedenen Transitionsregeln darstellen. Sei somit die Abbildung *TRL*

$$TRL : \mathbb{N} \to TR$$

eine solche Liste von Transitionsregeln. Damit ist es möglich, den Transitionen eines gegebenen QPN-Modells Transitionsregeln zuzuweisen:

$$R : T \to TRL$$

R sei hierbei die Abbildung, die die Zuweisung vornimmt. Dabei sei bemerkt, dass laut Definition die Zuweisung einer prinzipiell beliebig großen Liste von Transitionregeln zu einer Transition möglich ist. In der Praxis kommt dies nicht vor, da der Simulator die Listengröße begrenzt.

Zudem verbieten die Definitionen nicht, dass Duplikate vorkommen. Somit ist es möglich, dass zwei Transitionsregeln einer Liste dieselbe Komponenten besitzen. Hinsichtlich der Abarbeitung der Transitionsregeln beim Feuern einer Transition stellt das eine Redundanz dar.

Das wird umso deutlicher, wenn man die Abarbeitung der Transitionsregeln betrachtet. Pro Transition werden die Transitionsregeln der zugewiesenen Liste nacheinander verarbeitet. Das bedeutet, dass für jede Transitionsregel geprüft wird, ob die Voraussetzungen für das Feuern gegeben sind. Genauer gesagt, es wird geprüft, ob die entsprechenden farbigen Marken in den Eingabestellen vorhanden sind. Sofern das zutrifft, werden sie komsumiert. Sofern die Transitionsregel eine Verzögerungzeit vorsieht, stehen die konsumierten farbigen Marken während dieser nicht zur Verfügung. Erst danach werden gemäß der Transitionsregel neue farbige Marken in die Ausgabestellen erstellt und abgelegt. Dieses Vorgehen zeigt, dass Transitionsregeln mit niedrigem Index eine höhere Priorität genießen als diejenigen mit höherem Index. Das bedeutet, dass die Transitionsregeln an der Spitze die höchste und diejenige am Ende der Liste die niedrigste Verarbeitungspriorität besitzt.

5.3 Modell für das umgebende Betriebssystem

Datenbanksysteme werden im allgemeinen auf einem Betriebssystem ausgeführt. Dabei nutzt ein Datenbanksystem den vom Betriebssystem bereitgestellten Zugriff auf Betriebsmittel wie etwa Arbeits- und Massenspeicher, Netzwerkzugriff und Prozessausführungen. Aus Textabschnitt 4.2 ist bekannt, dass das Betriebssystem als Zwischenschicht zwischen Datenbanksystem und technischer Plattform grundlegenden Einfluss auf die Performance und den Energieverbrauch nimmt.

Nach der DIN-Sammlung 44300 ist die Definition eines Betriebssystems:

> *Ein Betriebssystem umfasst die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften dieser Rechenanlage die Basis der möglichen Betriebsarten des Rechensystems bilden und die insbesondere die Abwicklung von Programmen steuern und überwachen.*

Betrachtet man moderne Betriebssysteme, zum Beispiel *Windows*, *GNU/Linux* oder *MacOS*, so fällt auf, dass sie alle dieselben Aufgaben verfolgen. Im Grunde genommen implementieren sie die Architekturrichtlinien für ein generisches Betriebssystem, wie sie *Tanenbaum* in [TB14] beschrieben hat:

- Abstraktion der technischen Komponenten und Generalisierung des Zugriffes auf diese durch standardisierte Schnittstellen in Form von Treiber und Softwarebibliotheken
- Verwaltung der Betriebsmittel (Prozessoren, Hauptspeicher, Massenspeicher und Rechenzeit)
- Starten, Benden und Koordination der Anwendungsprogramme und Prozesse (sogenanntes *process scheduling*).

Um all diese Aspekte abzubilden, wurde im Rahmen dieser Dissertation ein QPN-Modell geschaffen, dass im folgenden als *Modell für das umgebende Betriebssystem* bezeichnet wird. Es ist vom QPN-Modell inspiriert, das *Kounev und Buchmann* in [KB03] vorgestellt haben (siehe Abbildung 2.15 auf Seite 51). Dieses Modell repräsentiert dabei ein zentrales Serversystem, in dem rudimentär die Betriebsmittel CPU, Arbeits- und Massenspeicher vorhanden sind. Auf dem zentralen

Serversystem wurde dann die Ausführung, die Koordinierung und das Beenden der Anwendungsprogramme modelliert, die von den Benutzer des zentralen Serversystems gestartet wurden. Zur Ausführung wurde dann zusätzlich der "Verbrauch" an Betriebsmitteln bei Ausführung der Anwendungsprogramme festgelegt. Zusätzlich sind im Modell für das umgebende Betriebssystem auch Aspekte aus den analytischen Modellen eingeflossen, die *Lazowska et al.* im Rahmen ihrer Computersystemanalyse erarbeitet haben [LZGS84]. Diese Modelle sind deswegen bedeutsam, da sie die Komponenten eines Betriebssystems (Prozess-Scheduling, Hauptspeicher- und Massenspeicherverwaltung, Prozessoren) anhand von Warteschlangen quantitativ beschreiben. Die analytischen Modelle sind formal sehr komplex und können im Rahmen dieser Dissertation nicht ausführlich erläutert werden. Daher sei auf [LZGS84] verwiesen. Zudem sei auch auch die umfassende Analyse von Betriebssystemen von *Greiner* in [Gre01] hingewiesen.

Es ist offensichtlich, dass das QPN-Modell von *Kounev und Buchmann* nicht ausreicht, um die oben genannten Aufgaben eines generischen Betriebssystems ausreichend abzubilden. Daher wurde es wie bereits erwähnt als Basis genutzt, um das Modell für das umgebende Betriebssystem zu erstellen. Es ist in Abbildung 5.3 dargestellt und besteht aus sieben Stellen und drei Transitionen.

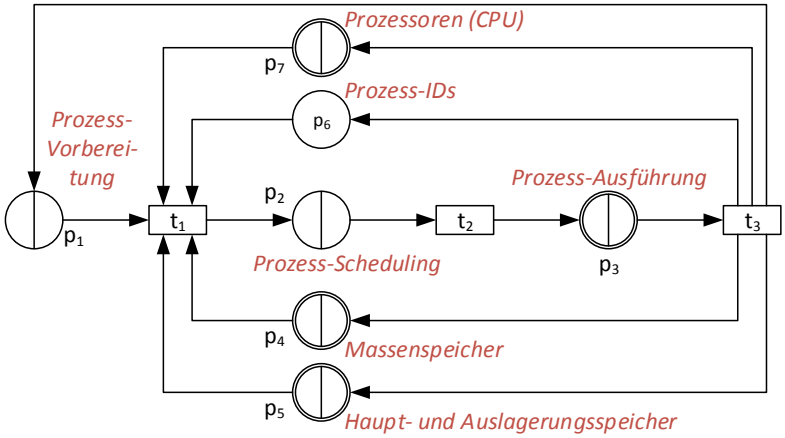


Abbildung 5.3: QPN-Modell für das umgebende Betriebssystem

Die in Abbildung 5.3 dargestellten Stellen können in zwei Arten unterteilt werden:

1. **Ressourcenstellen** Diese Stellen repräsentieren die Betriebsmittel des Systems, das abgebildet werden soll. Dazu gehören die Stellen p_4 (Massenspeicher), p_5 (Haupt- und Auslagerungsspeicher) und p_7 (Prozessoren, CPU). Die Ressourcenstellen haben den wesentlichen Zweck, *Ressourcenmarken* bereitzustellen, die innerhalb der anderen Modelle "konsumiert", dass heisst genutzt werden können. Dementsprechend stehen auch nur eine endliche Anzahl an Ressourcenmarken in den Stellen p_4, p_5 und p_7 bereit. Durch diesen Konsum wird das Verhalten von Anwendungsprogrammen, unter anderem auch Datenbanksysteme, modelliert, während ihrer Ausführung Betriebsmittel zu gebrauchen. Dazu gehört zum Beispiel die dynamische Allokation von Arbeits- und Massenspeicher während der Ausführung.
   In Abbildung 5.3 ist auch erkenntlich, dass die Ressourcenstellen p_4, p_5 und p_7 als Stellen mit innerem Modell ausgeführt sind. Der Hintergrund ist, dass dadurch die Möglichkeit eröffnet wird, verschiedene technische Ausführungen der Betriebsmittel zu modellieren. Dazu gehört zum Beispiel die Modellierung eines Mehrsockel- oder Mehrkernsystems im

Falle der Ressourcenstelle p_7 (CPU). Im weiteren Verlauf dieses Textabschnittes werden die inneren Modelle der genannten Ressourcenstellen genauer erklärt.

2. **Prozessstellen** Diese Stellen bilden die Koordinierung der Prozesse eines Betriebssystems ab. Das sind die Stellen p_1 (Prozessvorbereitung), p_2 (Prozess-Scheduling) und p_3 (Prozessausführungen).
   Stelle p_6 bildet die verfügbaren Prozess-IDs ab und wird als Prozessstelle angesehen, obwohl Prozess-IDs prinzipiell auch als Betriebsmittel betrachtet werden können. Da sie aber keine technische Komponente repräsentieren und näher an der Prozesskoordination stehen, gehört Stelle p_6 zu den Prozessstellen.

Tabelle 5.1 und 5.2 listen zur besseren Übersicht die Stellen und Transitionen auf. Erstgenannte Tabelle listet auch die Marken auf, die von den Stellen zur Verfügung gestellt werden.

Stelle	Marken	Kurzbeschreibung
	Stellen zur Prozesskoordinierung	
p_1	$G = (g_1, g_2, ..., g_{\|G\|})$ wobei $g \subset A \times B \times E \times F \times D$	Prozesse[7]
p_2		Prozess-Scheduling
p_3		Prozess-Ende
p_6	$F = (h_1, h_2, ..., h_{\|F\|})$	Prozess-IDs
	Stellen für Betriebsmittel	
p_4	$A = (s_1, s_2, ..., s_{\|A\|})$	Massenspeicher
p_5	$B = (b_1, b_2, ..., b_{\|B\|})$	Haupt- und Auslagerungsspeicher
p_7	$E = (e_1, e_2, ..., e_{\|E\|})$	Prozessoren, CPU

Tabelle 5.1: Stellen des QPN-Modells für das umgebende Betriebssystem

Transition	Kurzbeschreibung
t_1	Resourcenallokation für einen Prozess
t_2	(existiert nur, um formale QPN-Definition zu erfüllen)
t_3	Resourcendeallokation nach Prozessende

Tabelle 5.2: Transitionen des QPN-Modells für das umgebende Betriebssystem

Das Modell für das umgebende Betriebssystem arbeitet folgendermaßen:

1. Ein neuer Prozess soll gestartet werden, zum Beispiel eine Instanz eines installierten Datenbanksystems. Aus Stelle p_1 wird eine Prozessmarke entnommen, die bereits alle Parameter beziehungsweise Daten für den Prozess enthält. Transition t_1 ist für die Allokation aller Betriebsmittel zuständig, die für den Prozess notwendig sind. Das sind eine oder mehrere Resourcenmarken von den Resourcenstellen p_4, p_5 und p_7 (Massen- und Hauptspeicher, CPU) sowie eine Marke von Stelle p_6 (Prozess-ID). Wenn alle notwendigen Marken bereitstehen, wird die Prozessmarke mit den Resourcenmarken verknüpft. Die Transitionszeit für

[7] Die Menge D als Teil des Tupels G ist in Tabelle 5.1 nicht aufgeführt. Sie wird im Textabschnitt 5.4.1 ab Seite 144 sowie in Textabschnitt 5.7.2 ab Seite 157 erklärt.

t_1 ist einstellbar, um die charakteristischen Zeiten für die Prozesserstellung von existierenden Betriebssystemen zu realisieren.

2. Alle so erzeugten Prozessmarken gelangen zur Stelle p_2, die das Prozess-Scheduling umsetzt. Dabei können verschiedene Strategien genutzt werden, zum Beispiel ein prioritätsbasiertes Schedulung mit dem PRIO-Scheduler. Letztendlich wird in Stelle p_2 entschieden, in welcher Reihenfolge wartende Prozessmarken an die Transition t_2 weitergereicht werden. Auch hier sind die Transitionszeiten kalibrierbar, um die existierenden Schedulingalgorithmen möglichst realitätsgetreu nachbilden zu können.
3. Ausgehend von Transition t_2 wird der Prozess, der durch eine Prozessmarke repräsentiert wird, in Stelle p_3 ausgeführt. Als einzige der drei Prozessstellen ist p_3 als Stelle mit innerem Modell ausgeführt. Somit kann die Ausführung des Prozesses durch ein eigenes inneres Modell modelliert werden. Hierbei ist es wichtig anzumerken, dass die Betriebsmittel in Form von Resourcenmarken nicht pre-allokiert sind, die während des Prozesses gebraucht werden. Das heisst, dass das innere QPN-Modell in p_3 durchaus weitere Resourcenmarken von den Stellen p_4, p_5 und p_7 anfordern und nutzen kann.
4. Wenn der Prozess beendet ist, werden alle Resourcenmarken in Transition t_3 an die entsprechenden Stellen p_4 bis p_7 zurückgegeben. Damit wird die Freigabe der genutzten Betriebsmittel realisiert. Die Prozessmarke gelangt wieder in Stelle p_1 und steht damit einem weiterem, neu zu startenden Prozess zur Verfügung.

Anhand dieser Arbeitsweise ist auch erkennbar, dass das Modell für das umgebende Betriebssystem in Abbildung 5.3 auch in der Lage ist, nebenläufige Prozesse abzubilden. Das ist insbesondere dann wichtig, wenn Mehrbenutzersysteme simuliert werden sollen.

5.3.1 Inneres Modell für den Hauptspeicher

Wie bereits erwähnt, ist Stelle p_5 für das Betriebsmittel Arbeits- und Auslagerungsspeicher in Form von Resourcenmarken der Tupelmenge B zuständig. Von der Perspektive eines Prozesses aus ist es unerheblich, woher angeforderter Hauptspeicher seinen Ursprung hat (Arbeits- oder Auslagerungsspeicher). Moderne Betriebssysteme sind in der Lage, Prozessen auch weitaus mehr Arbeitsspeicher zuzuweisen, als physikalisch vorhanden ist. Dabei wird Auslagerungsspeicher zugewiesen, der zumeist auf langsameren Massenspeicher verwaltet wird. Aktive Segmente des schnellen Arbeitsspeicher werden auf Segmente des Auslagerungsspeichers projeziert und können bei Bedarf ausgetauscht werden. In fast allen modernen Betriebssystemen wird für dieses Vorgehen der *Least-recently-used*-Algorithmus genutzt.

Um dieses Verhalten auch im Modell für das umgebende Betriebssystem abzubilden, enthält Stelle p_5 von Abbildung 5.3 ein inneres Modell, das in Abbildung 5.4 auf Seite 135 abgebildet ist. Dieses innere Modell genügt der formalen Definition beziehungsweise der grafischen Notation, wie sie in Abbildung 2.16 auf Seite 52 dargestellt wird. Zusätzlich basiert es auf den analytischen Betrachtungen zum Arbeitsspeicher-Subsystems beziehungsweise der möglichen Modellierung von *Lazowska et al.* [LZGS84, S.179 ff.].

Das innere Modell für den Haupt- und Auslagerungsspeicher besteht dabei aus drei Stellen p_{5a} bis p_{5c} sowie aus drei Transitionen t_{5a} bis t_{5c}, die die Stellen verbinden. Stelle p_{5a} stellt dabei den Arbeits- und Stelle p_{5b} den Auslagerungsspeicher dar.

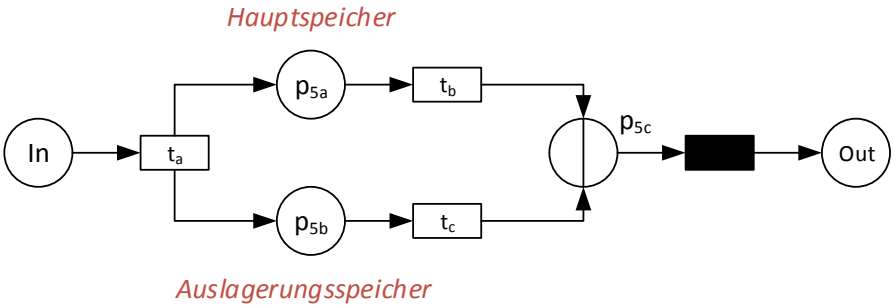


Abbildung 5.4: Inneres Modell für den Haupt- und Auslagerungsspeicher (Stelle p_5 im Modell für das umgebende Betriebssystem in Abbildung 5.3)

Das innere Modell, wie es in Abbildung 5.4 dargestellt ist, unterstützt zwei wesentliche Aufgaben:

1. **Allokation von Hauptspeicher** Gegenüber den anderen QPN-Modellen, die innerhalb dieses Kapitels 5 vorgestellt und erläutern werden, dient dieses innere Modell dazu, Arbeitsspeicher in Form von Ressourcenmarken bereitzustellen. Innerhalb der anderen QPN-Modellen wird in dieser Form die Allokation von Arbeitsspeicher nachgestellt.
   Arbeitsspeicher-Ressourcenmarken können ihren Ursprung entweder in Stelle p_{5a} (Arbeitsspeicher) oder in Stelle p_{5b} (Auslagerungsspeicher) haben. Die Transitionen t_b und t_c enthalten dabei jeweils eine Transitionsregel, die sich nur hinsichtlich der Transitionszeit unterscheidet. Diese Unterscheidung soll verdeutlichen, dass die Entnahme von Marken aus den Stellen p_{5a} und p_{5b} zeitlich abweicht. Der Hintergrund ist, dass so zeitlich modelliert werden kann, dass die Bereitstellung von Hauptspeicher deutlich schneller ist als diejenige von Auslagerungsspeicher.
   Eine entscheidende Rolle spielt Stelle p_{5c}. Sie besitzt eine Warteschlange mit Prioritäts-Scheduler, wobei Marken mit Ursprung aus p_{5a} (Hauptspeicher) höher priorisiert werden als Marken aus p_{5b} (Auslagerungsspeicher). Damit wird modelliert, dass die Allokation von Arbeitsspeicher in erster Linie durch Marken des (im allgemeinen schnelleren) Hauptspeichers durchgeführt werden soll. Erst wenn alle Marken aus p_{5a} erschöpft sind, werden Marken aus p_{5b} entnommen, also vom (im allgemeinen langsameren) Auslagerungsspeicher. Insgesamt stehen die Arbeitsspeicher-Ressourcenmarken über die Transition von Stelle p_{5c} zur Stelle *Out* den anderen QPN-Modellen zur Verfügung. Die Stelle *Out* dient hier nur dazu, dem Formalismus eines inneren Modells zu genügen.
2. **Deallokation von Arbeitsspeicher** Die im ersten Punkt erwähnten QPN-Modelle können konsumierte Arbeitsspeicher-Ressourcenmarken wieder "zurückgeben". Das entspricht aus technischer Sicht der Deallokation von Arbeitsspeicher, dass heisst, allokierter Arbeitsspeicher wird wieder freigegeben und steht einer erneuten Allokation zur Verfügung. Innerhalb des inneren Modells von Abbildung 5.4 wird dieser Sachverhalt durch Arbeitsspeicher-Ressourcenmarken repräsentiert, die das innere Modell durch die Stelle *In* erreichen, also dort abgelegt werden. Die Transition t_a entnimmt beziehungsweise konsumiert diese Marken und erzeugt dann jeweils neue Marken in den Stellen p_{5a} und p_{5b}, wobei die ursprüngliche Herkunft der Marken einbezogen wird. Um auch hier zeitlich zu differenzieren, werden in Transition t_a zwei Transitionsregeln genutzt, die sich in den Transitionszeiten unterscheiden. Damit wird modelliert, dass die Deallokation von Arbeitsspeicher

mit Ursprung Hauptspeicher deutlich schneller vonstatten geht als diejenige mit Ursprung Auslagerungsspeicher.

Allerdings ergibt sich ein Problem. In den erwähnten anderen QPN-Modellen können Arbeitsspeicher-Ressourcenmarken konsumiert werden, die nicht unterscheidbar sind. Formal werden also diese Marken durch nur eine Farbe dargestellt. Sei diese Farbe mit c bezeichnet. Innerhalb des inneren Modells in Abbildung 5.4 werden diese Marken jedoch nach Ursprung unterschieden. Seien also die Marken in Stelle p_{5a} mit der Farbe c_1 und diejenigen in Stelle p_{5b} mit c_2 bezeichnet. Das ist notwendig, damit das prioritätsbasiertes Scheduling in Stelle p_{5c} wirksam wird. Prinzipiell ist es möglich, dass durch Stelle p_{5c} Marken der Farbe c_1 und c_2 konsumiert werden und dann Marken der Farbe c erzeugt werden. Bei der "Rückgabe" der Marken mit der Farbe c (Deallokation von Arbeitsspeicher) entsteht nun das Problem, dass nicht entschieden werden kann, welche ursprüngliche Farbe c_1 oder c_2 die Marke mit der Farbe c hatte.

Das Problem wurde technisch über den Simulator *QPME* gelöst, der im Rahmen dieser Dissertation genutzt wurde [Kou08]. Formal betrachtet stellen die Markenbelegungen allen Stellen des inneren Modells in Abbildung 5.4 Multimengen dar, die auch so im Simulator *QPME* repräsentiert und genutzt werden können. Technisch betrachtet werden sie jedoch als Tupel repräsentiert, dass heisst, die Marken sind über ihre Index unterscheidbar.

Zur Illustration sei mit n die Anzahl an Marken in Stelle p_{5a} mit der Farbe c_1 und mit m die Anzahl der Marken in Stelle p_{5b} mit der Farbe c_2 bezeichnet. Betrachtet man die Warteschlange von Stelle p_{5c}, so ergibt sich folgende Multimenge:

$$\{\underbrace{c_1, c_1, c_1, c_1, \ldots, c_1}_{n \textit{ Elemente}}, \underbrace{c_2, c_2, c_2, \ldots, c_2}_{m \textit{ Elemente}}\}$$

Das erste Element dieser Multimenge sei der Kopf der Warteschlange. Durch Stelle p_{5c} wird dann folgende Multimenge im Ausgabebereich dieser Stelle generiert:

$$\{\underbrace{c, c, c, c, \ldots, c}_{n \textit{ Elemente}}, \underbrace{c, c, c, \ldots, c}_{m \textit{ Elemente}}\}$$

Technisch wird diese Multimenge innerhalb von *QPME* als Tupel mit $n + m$ Komponenten betrachtet:

$$(\underbrace{c_{[1]}, c_{[2]}, c_{[3]}, c_{[4]}, \ldots, c_{[n]}}_{n \textit{ Elemente}}, \underbrace{c_{[n+1]}, c_{[n+2]}, c_{[n+3]}, \ldots, c_{[n+m]}}_{m \textit{ Elemente}})$$

Die eckigen Klammern geben zur besseren Unterscheidung den Index einer Komponenten in diesem Tupel an. Sofern eine Marke mit der Farbe c über die Stelle *In* die Transition t_{5a} erreicht, kann diese Transition anhand des Index entscheiden, in welche Stelle (p_{5a} oder p_{5b}) eine neue Marke mit der Farbe c_1 oder c_2 erstellt und abgelegt wurde:

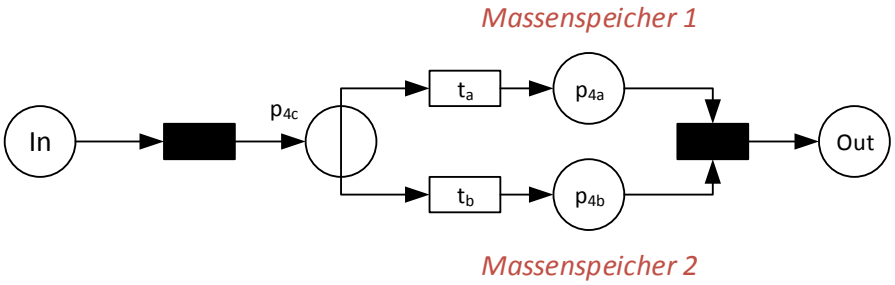
$$\forall x \in \mathbb{N}^+ : 1 \leq x \leq (n+m) : c_{[x]} \begin{cases} x \leq n & \textit{Lege neue Marke mit Farbe } c_1 \textit{ in Stelle } P_{5a} \textit{ ab} \\ \textit{sonst} & \textit{Lege neue Marke mit Farbe } c_2 \textit{ in Stelle } P_{5b} \textit{ ab} \end{cases}$$

Die ursprüngliche Marke mit der Farbe c wird dabei konsumiert.

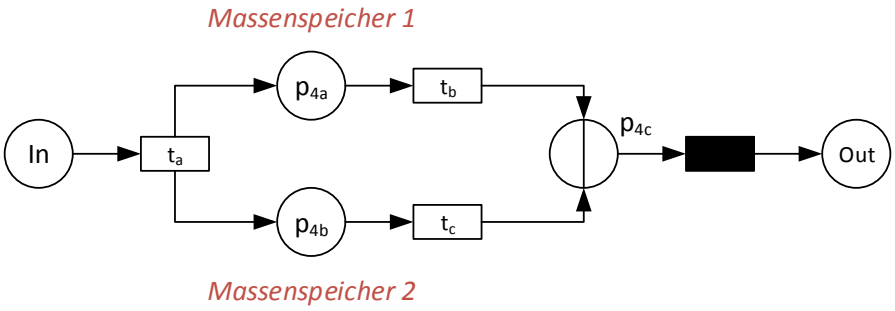
5.3.2 Inneres Modell für den Massenspeicher

Die Stelle p_4, die in Abbildung 5.3 auf Seite 132 erkennbar ist, kann ein inneres Modell enthalten, um den Massenspeicher als Betriebsmittel des umgebenden Betriebssystems abzubilden. Im einfachsten Fall kann dieser Massenspeicher eine Festplatte sein, sodass kein spezifisches inneres Modell erstellt werden muss. In diesem Fall ist eine ordinäre Stelle ausreichend. Sofern aber komplexere Massenspeicher-Szenarien abgebildet werden sollen, ist ein inneres Modell ratsam, zum Beispiel bei der Modellierung von Festplattenverbünden (RAID[8]) oder entfernt liegenden Massenspeichersystemen (SAN[9]).

Abbildung 5.5 zeigt zwei Beispiele eines inneren Modells, das Massenspeicher in Form eines RAID-0-Verbundes aus zwei Festplatten repräsentiert. Es basiert auf den Modellen, die *Noorshams et al.* und *LLadó et al.* bei ihrer Untersuchung und Modellierung von RAID-Festplattenverbünden erstellt haben ([NRKR14, LH11]) sowie auf den Betrachtungen von *Lazowska et al.* zum I/O-Subsystem eines Computers [LZGS84, S.222 ff.]. *Kounev und Buchmann* haben in ihrer Untersuchung zusätzlich gezeigt, dass die modellierten RAID-Festplattenverbünden praktikabel in anderen, übergeordneten Modellen genutzt werden können [KB03]. Das betrifft insbesondere alle QPN-Modelle, die innerhalb dieses Kapitels 5 vorgestellt und erklärt werden.



(a) Modell mit Fokus auf Schreibperformance



(b) Modell mit Fokus auf Leseperformance

Abbildung 5.5: Beispiele für ein inneres Modell für den Massenspeicher (Stelle p_4 im Modell für das umgebende Betriebssystem in Abbildung 5.3)

[8] Akronym für *Redundant Array Of Independant Disks*: ein Verfahren, um mehrere Massenspeicher zu einem logischen zu verbinden, um zum Beispiel den Datendurchsatz oder die Ausfallsicherheit zu steigern.

[9] *Storage Area Network*: ein dediziertes Netzwerk, dass Server und Massenspeichersysteme miteinander verbindet. Damit ist es möglich, dass sich mehrere Systeme den Massenspeicher teilen. Zumeist werden performante, auf Datendurchsatz optimierte Netzwerktechnologien eingesetzt.

Die in Abbildung 5.5 gezeigten inneren Modelle genügen der formalen Definition beziehungsweise der grafischen Notation, wie sie in Abbildung 2.16 auf Seite 52 dargestellt wird. Beide gezeigten QPN-Modelle können dazu dienen, Massenspeicher-Ressourcenmarken des Tupels A aus Tabelle 5.1 auf Seite 133 für die übrigen QPN-Modelle bereitzustellen, die in diesem Kapitel 5 eingeführt werden.

Die Erfahrungen aus den oben genannten Publikationen zeigen, dass man sich bereits bei der Erstellung eines QPN-Modells, das ein komplexeres Massenspeichersystem repräsentieren soll, auf das spätere Simulationsziel fokussieren soll. Genauer gesagt, das QPN-Modell wird so entworfen, dass eine spätere Simulation die Performance beziehungsweise den Durchsatz bei entweder lesenden oder schreibenden Zugriff auf das modellierte Massenspeichersystem evaluieren soll.

Zur Illustration dieses Sachverhaltes können die beiden Teilabbildungen 5.5(a) und 5.5(b) von Abbildung 5.5 betrachtet werden. Beide Teilabbildungen zeigen ein Massenspeichersystem mit zwei Festplatten, die durch die Stellen p_{4a} und p_{4b} repräsentiert werden. Wichtig hierbei ist, an welcher Position Stelle p_{4c} relativ zu den beiden Stellen p_{4a} und p_{4b} modelliert ist:

- Befindet sie sich *vor* den beiden Stellen, ist sie somit eine *Eingabestelle* für beiden anderen Stellen. In Teilabbildung 5.5(a) wird dies dargestellt, wobei Stelle p_{4b} über die Transition t_a und t_b mit den Stellen p_{4a} und p_{4b} verbunden ist.
  Die Funktionsweise des QPN-Modells in Teilabbildung 5.5(a) macht deutlich, dass eine spätere Simulation die Performance beziehungsweise den Durchsatz für Schreibvorgänge auf die beiden modellierten Festplatten zum Ziel hat. Ein wichtiger Fakt hierbei ist, dass die Stellen p_{4a} und p_{4b} initial mit keinerlei Marken belegt sind. Diese Marken werden durch übergeordnete QPN-Modelle erzeugt und gelangen über die Stelle *In* in Teilabbildung 5.5(a) in das innere Modell. Sie werden dann von Stelle p_{4c} konsumiert, wobei eine Warteschlange mit einem Schedulingalgorithmus zum Einsatz kommt, der die gewünschte RAID-Funktionalität implementiert. Soll zum Beispiel RAID-0 (sogenanntes *striping*, Verteilung) modelliert werden, konsumiert Stelle p_{4c} eine Marke von Stelle *In* und legt abwechselnd eine neue erzeugte Marke in Stelle p_{4a} oder p_{4b} ab. Es kann aber RAID-1 (sogenanntes *mirroring*, Spiegelung) modelliert werden. Dabei konsumiert Stelle p_{4c} eine Marke aus Stelle *In* und erzeugt zwei neue Marken, wobei jeweils eine in Stelle p_{4a} und p_{4b} abgelegt werden.
  Die in Teilabbildung 5.5(a) gezeigten Transitionen t_a und t_b kommen darüberhinaus eine zusätzliche Bedeutung zu. Über sie kann die mittlere Zugriffsgeschwindigkeit bei Schreibvorgängen auf die modellierten Festplatten abgebildet werden. So kann zum Beispiel im inneren Modell in Teilabbildung 5.5(a) in diesen Transitionen ein RAID-Verbund modelliert werden, wobei eine SSD mit sehr schneller Zugriffsgeschwindigkeit mit einer magnetische Festplatte mit weniger schnellen Zugriffsgeschwindigkeit verbunden wird.
- Befindet sie sich *nach* den beiden Stellen, ist sie eine *Ausgabestelle* für die beiden anderen. In Teilabbildungen 5.5(b) wird dies dargestellt, wobei die beiden Stellen p_{4a} und p_{4b} über die Transitionen t_b und t_c mit der Stelle p_{4c} verbunden ist.
  Der Hauptunterschied zum vorhergehenden Punkt ist, dass die Stellen p_{4a} und p_{4b} initial bereits mit Marken belegt sind. Wie in Teilabbildung 5.5(b) ersichtlich, sind beide Stellen über die Transitionen t_b und t_c mit Stelle p_{4c} verbunden, die die Marken über die Stelle *Out* anderen QPN-Modellen zur Verfügung stellt. Über die Warteschlange von Stelle p_{4c} und dessen Schedulingalgorithmus kann bestimmt werden, welcher RAID-Modus im inneren Modell auf Teilabbildungen 5.5(b) modelliert werden soll.
  Ein wesentlicher Fakt, der im Zusammenhang mit dem inneren Modell auf Teilabbildung 5.5(b) genannt werden muss, ist die Analogie zur Funktionsweise mit dem inneren Modell

für den Arbeitsspeicher, das in vorhergehenden Textabschnitt beschrieben und auf Abbildung 5.4 dargestellt ist. Vergleicht man Teilabbildung 5.5(b) mit Abbildung 5.4 auf Seite 135, so erkennt man hinsichtlich der Struktur keinerlei Unterschiede. Zudem gleicht sich auch die Funktionsweise: Marken des inneren Modells werden den übrigen QPN-Modellen zum Konsum zur Verfügung gestellt und entspricht somit einer Allokation. Dementsprechend müssen auch die Problematiken im Umgang mit dem inneren Modell in Betracht gezogen werden, wie sie im vorherigen Textabschnitt beschrieben sind[10].
Auch im inneren Modell, wie es in Teilabbildung 5.5(b) dargestellt ist, können die Zugriffszeiten auf die modellierten Massenspeicher über die Transitionen t_b und t_c individuell angepasst werden. Somit wird auch hier die Repräsentation von RAID-Verbünden möglich, in denen sich die Zugriffsgeschwindigkeiten der beteiligten Massenspeicher bei Schreibvorgängen stark unterscheiden. Das ist zum Beispiel der Fall, wenn im inneren Modell in Teilabbildung 5.5(b) die Stelle p_{4a} eine SSD- und Stelle p_{4b} eine magnetische Festplatte darstellen soll.

Im Bezug auf die Simulation der beiden inneren Modelle, wie sie in Abbildung 5.5 dargestellt sind, kann nach erfolgter Simulation der simulierte Fluss der Marken durch die Stellen p_{4a} bis p_{4c} betrachtet werden [NRKR14, LH11, Kou08]. Dabei dienen statistisch Werte, die durch die Simulation gewonnen wurden, zur Errechnung der simulierten Performance beziehungsweise des Datendurchsatzes. Dabei wird zum Beispiel die mittlere Verweildauer der Marken in den drei Stellen sowie die minimale und maximale Population an Marken in diesen Stellen genutzt.

5.3.3 Inneres Modell für den Prozessor

Ressourcenstelle p_7 des Modells für das umgebende Betriebssystem, ersichtlich in Abbildung 5.3 auf Seite 132, repräsentiert das Berechnungs- und Verarbeitungspotential hinsichtlich der Prozessoren für das technische System, das modelliert werden soll. Dazu gehört einerseits die CPU für generelle Berechnungsoperationen und anderseits zumeist dedizierte Spezialprozessoren, die auf die Verarbeitung untypischer Datenstrukturen optimiert sind. Prominente Beispiele sind zum Beispiel Grafikkarten, die äußerst performant Vektor- und Matrizenoperationen durchführen können sowie sogenannte *Floating Point Units* (FPUs), die effizient Gleitkommaberechnungen unterstützen.

Zusätzlich existieren im Bezug auf CPUs auch eine Vielzahl an Architekturen [LZGS84, S.253]. So bieten manche technische Plattformen die Möglichkeit, mehrere CPUs einzubinden (Multi-Sockel-Systeme). Moderne CPUs beinhalten zudem mehrere getrennte Rechenkerne (Mehrkern-CPUs). Um all diese Architekturen und Plattformen zu unterstützen und modellieren zu können, kann oben genannte Resourcenstelle p_7 ein inneres Modell enthalten. Bei trivialen Systemen mit nur einer CPU, keinen Spezialprozessoren und keinen gesonderten Rechenkernen ist dies nicht notwendig.

In Abbildung 5.6 ist ein Beispiel für ein inneres Modell dargestellt, das ein Multi-Sockel-System modelliert. Dieses System besteht aus vier Sockeln, womit ausgedrückt wird, dass vier CPUs modelliert werden sollen. Wie bei den bereits eingeführten anderen inneren Modellen für den Haupt- und Auslagerungsspeicher sowie für den Massenspeicher auch, genügt das innere Modell für den Prozessor in Abbildung 5.6 der formalen Definition beziehungsweise der grafischen Notation, wie sie in Abbildung 2.16 auf Seite 52 dargestellt wird. Insgesamt kann das in Abbildung 5.6

[10] Siehe dazu den Textabschnitt zur Beschreibung des inneren Modells für den Haupt- und Auslagerungsspeicher ab Seite 134.

gezeigte innere Modell dazu dienen, Prozessor-Ressourcenmarken des Tupels E aus Tabelle 5.1 auf Seite 133 für die übrigen QPN-Modelle bereitzustellen, die in diesem Kapitel 5 eingeführt werden. Diese Prozessor-Ressourcenmarken repräsentieren sozusagen das Berechnungs- und Verarbeitungspotential des modellierten technischen Systems. Deren Konsum soll ausdrücken, dass Prozessoren-Rechenzeit verbraucht wurde.

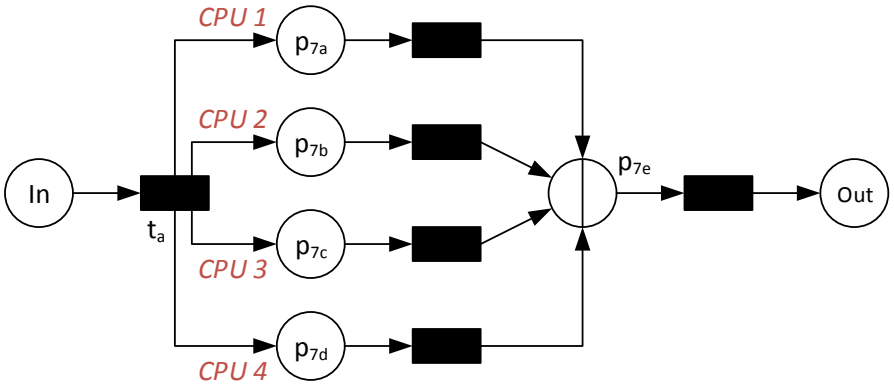


Abbildung 5.6: Beispiel für ein inneres Modell für den Prozessor (Stelle p_7 im Modell für das umgebende Betriebssystem in Abbildung 5.3)

Das in Abbildung 5.6 gezeigte innere Modell besitzt die Stellen p_{7a} bis p_{7d}, wobei jede dieser Stellen einen Prozessor repräsentieren soll. In Abbildung 5.6 wird dies durch die Annotationen "CPU 1" bis "CPU 4" an den entsprechenden Stellen dargestellt.

Betrachtet man das innere Modell in Abbildung 5.6 und vergleicht es mit dem inneren Modell für den Haupt- und Auslagerungsspeicher auf Abbildung 5.4, so erkennt man strukturell kaum Unterschiede. Das bedeutet, dass die Stellen p_{7a} bis p_{7d} mit der Stelle p_{7e} über Transitionen verbunden sind. Sind die erstgenannten Stellen initial mit Marken belegt, so stehen sie über letztgenannte Stelle für eine Allokation durch andere QPN-Modelle zur Verfügung. Die Stelle p_{7e} übernimmt hierbei die Aufgabe, diese Marken über einen geeigneten Schedulingalgorithmus der Warteschlange neu anzuordnen und zu konsumieren, um dann im Ausgabebereich Prozessoren-Ressourcenmarken zu erzeugen und abzulegen. Diese Ressourcenmarken stehen dann in der Stelle *Out* für eine Allokation bereit.

Der Schedulingalgorithmus der Warteschlange von Stelle p_{7e} kann vielfältig ausgeprägt sein. Hierbei sei besonders auf die analytischen Betrachtungen von *Lazowska et al.* verwiesen, die die Schedulingalgorithmen quantitativ beschreiben [LZGS84, S.253 ff.]. So ist zum Beispiel ein Verschränken der Marken aus den Stellen p_{7a} bis p_{7d} möglich (sogenanntes *interleaving*). Um dieses Verfahren zu illustrieren, seien die Markenbelegungen in diesen Stellen jeweils mit einer Multimenge A bis D repräsentiert, wobei in jeder dieser Multimenge eine andere Farbe c_1 bis c_4 für die Marken genutzt wird. Somit entstehen vier Multimengen:

$$A = \{\underbrace{c_1, c_1, c_1, \ldots, c_1}_{|A| \text{ Elemente}}\}, B = \{\underbrace{c_2, c_2, c_2, \ldots, c_2}_{|B| \text{ Elemente}}\}, C = \{\underbrace{c_3, c_3, c_3, \ldots, c_3}_{|C| \text{ Elemente}}\}, D = \{\underbrace{c_4, c_4, c_4, \ldots, c_4}_{|D| \text{ Elemente}}\}$$

Durch die Verschränkung entsteht folgende Reihenfolge der Elemente aus den Multimengen A bis D:

$$\{\underbrace{c_1, c_2, c_3, c_4}_{\substack{1. \text{ Element aus} \\ A \text{ bis } D}}, \underbrace{c_1, c_2, c_3, c_4}_{\substack{2. \text{ Element aus} \\ A \text{ bis } D}}, \underbrace{c_1, c_2, c_3, c_4}_{\substack{3. \text{ Element aus} \\ A \text{ bis } D}}, \ldots\}$$

Es ist auch möglich, die Elemente aus den Multimengen A bis D prioritätsbasiert anzuordnen, wobei die Multimenge A die höchste und die Multimenge D die niedrigste Priorität genießt:

$$\{\underbrace{c_1, c_1, c_1, \ldots, c_1}_{|A| \textit{ Elemente}}, \underbrace{c_2, c_2, c_2, \ldots, c_2}_{|B| \textit{ Elemente}}, \underbrace{c_3, c_3, c_3, \ldots, c_3}_{|C| \textit{ Elemente}}, \underbrace{c_4, c_4, c_4, \ldots, c_4}_{|D| \textit{ Elemente}}\}$$

Allerdings muss auch hier bemerkt werden, dass dieselben Problematiken im Bezug auf die Prozessoren-Ressourcenmarken entstehen, wie sie bereits für die Bereitstellung von Arbeitsspeicher-Ressourcenmarken beschrieben sind (siehe Textabschnitt für das innere Modell für Haupt- und Auslagerungsspeicher ab Seite 134). Das bedeutet, dass aus der Multimenge der Marken mit den Farben c_1 bis c_4 nach der Neuanordnung technisch ein Tupel gebildet wird, dessen Komponenten die Prozessoren-Ressourcenmarken darstellen. Für die zuletzt gezeigte Anordnung bedeutet dies prinzipiell eine Aneinanderreihung der Elemente aus A bis D:

$$\underbrace{\{\underbrace{c_1, c_1, c_1, \ldots, c_1}_{|A| \textit{ Elemente}}, \underbrace{c_2, c_2, c_2, \ldots, c_2}_{|B| \textit{ Elemente}}, \underbrace{c_3, c_3, c_3, \ldots, c_3}_{|C| \textit{ Elemente}}, \underbrace{c_4, c_4, c_4, \ldots, c_4}_{|D| \textit{ Elemente}}\}}_{(c_{[1]}, c_{[2]}, c_{[3]}, \ldots, c_{[|A|]}, c_{[|A|+1]}, c_{[|A|+2]}, c_{[|A|+3]}, \ldots, c_{[|A|+|B|]}, \ldots)}$$

Sei c hierbei die Farbe, die für Prozessoren-Ressourcenmarken genutzt wird. Die eckigen Klammern geben zur besseren Unterscheidung den Index einer Komponente in dem Tupel an.

Durch diese Darstellung als Tupel und die Indizierung ist somit auch eine Deallokation von Prozessor-Ressourcenmarken möglich. Semantisch bedeutet es, dass Prozessorzeit über die vorherige Allokation (Konsum von Prozessor-Ressourcenmarken über die Stelle *Out* in Abbildung 5.6) "verbraucht" wurde und nun neue Prozessorzeit bereitgestellt wird, die wieder konsumiert werden kann. Die Deallokation bezeichnet also die "Rückgabe" von Prozessor-Ressourcenmarken. Für das innere Modell, das in Abbildung 5.6 gezeigt ist, gelangen konsumierte Prozessoren-Ressourcenmarken durch die Stelle *In* in das innere Modell. Da die Prozessoren-Ressourcenmarken durch ihren Index unterscheidbar sind, kann auf die Stelle p_{7a} bis p_{7d} geschlossen werden, aus der diese Marken ihren Ursprung hatte. Somit konsumiert Transition t_a in Abbildung 5.6 die Prozessoren-Ressourcenmarken und erstellt neue Marke mit den Farben c_1 bis c_4, die in die entsprechenden Stellen abgelegt werden.

Im inneren Modell, das in Abbildung 5.6 dargestellt wird, fällt auch auf, dass alle Transitionen als Transitionen ohne Zeitverzug modelliert sind. Damit soll ausgedrückt werden, dass die Allokation und Deallokation von Prozessoren-Ressourcenmarken nicht zeitverzögert ist. In der Realität konnte zum Beispiel das Scheduling von Betriebssystem-Prozessen dazu führen, dass es zu Zeitverzögerungen kam. Der Grund waren Kontextwechsel innerhalb der CPU(s). Das bedeutet, dass zum Beispiel Prozessorregister gesichert werden mussten, um sie mit neuen Werten zu beschreiben sowie die Neubefüllung des Instruktions-Caches der CPU. Mittlerweile sind aber die Kontextwechsel in modernen CPU-Architekturen so effizient gelöst, dass der Zeitverzug zwar noch existent, aber so klein ist, dass er nicht mehr signifikant ist.

5.4 Modell für das Datenbanksystem

Neben dem Modell des umgebenden Betriebssystem, das im vorherigen Textabschnitt 5.3 samt den inneren Modellen beschrieben wurde, stellt das *Modell für das Datenbanksystems* ein zweites grundlegendes QPN-Modell im Rahmen dieser Dissertation dar.

Prinzipiell stellt das Modell für das Datenbankmodell ein Modell dar, dass unabhängig genutzt werden kann. Es ist aber speziell darauf ausgerichtet, innerhalb des Modells für das umgebende Betriebssystem genutzt zu werden. Wie bereits in Textabschnitt 5.3 beschrieben, dient das Modell des umgebenden Betriebssystems dazu, die Allokation und Deallokation von Betriebsmitteln wie Arbeits- und Massenspeicher sowie CPU-Verarbeitungszeit in Form von Ressourcenmarken zu ermöglichen. Zudem wird der Start, die Koordination und das Beenden von Betriebssystem-Prozessen abgebildet. Das Modell für das Datenbanksystem repräsentiert ein generelles Datenbanksystem im Sinne eines Betriebssystem-Prozesses. Das bedeutet, das nachgebildete Datenbanksystem wird als ausführbares Anwendungsprogramm aufgefasst und kann als solches innerhalb des Modells für das umgebende Betriebssystem gestartet und gestoppt werden. Werden mehrere Instanzen gestartet, kann durch das Modell für das umgebende Betriebssystem deren Nebenläufigkeit abgebildet werden. Das wird dadurch bewerkstelligt, dass zu jeder zusätzlichen Instanz eines Datenbanksystem eine neue Instanz des Modells für das Datenbanksystem erzeugt wird, das dann als inneres Modell von Stelle p_3 vom Modell des umgebenden Betriebssystems auf Abbildung 5.3 von Seite 132 ausgeführt wird.

Das Modell des Datenbanksystems wurde so konzipiert, dass die wesentlichsten Kernpunkte und Charakteristiken von Datenbanksystemen abgebildet werden. Dazu gehören:

1. die Daten- und Speichermodelle von Datenbanksystemen, beschrieben in Textabschnitt 2.1 ab Seite 12
2. die vier Kernkomponenten bei der Verarbeitung eines Zugriffes auf die Datenbasis eines Datenbanksystems (Zugangskontrolle, Erstellung des Zugriffsplans, Transaktionskontrolle und Ausführung des Zugriffsplans), die in Textabschnitt 2.3 ab Seite 24 beschrieben sind
3. die Ausführung eines Benchmarks für ein Datenbanksystem, beschrieben in Textabschnitt 3.1 ab Seite 53, als Anwendungsfall für das Modell für das Datenbanksystem

Zusätzlich ist bei der Konzeption des Modells für das Datenbanksystem darauf geachtet worden, dass es der formalen Definition eines inneren Modells beziehungsweise der grafischen Notation genügt, wie sie in Abbildung 2.16 auf Seite 52 dargestellt wird.

Das Modell für das Datenbanksystem ist in Abbildung 5.7 auf Seite 143 dargestellt. Es wirkt recht komplex und unübersichtlich, folgt aber einem gewissen Grundaufbau, auf dem im folgenden näher eingegangen wird. In der Mitte von Abbildung 5.7 erkennt man die vier Kernkomponenten eines Datenbanksystems, die zur Verarbeitung von Datenbankoperationen notwendig sind. Diese vier Kernkomponenten sowie der Ablauf wurden in Textabschnitt 2.3 detailliert erläutert. Die Verarbeitungskette, die von den vier Kernkomponenten gebildet wird, kann im Teilgraphen erkannt werden, der von den Stellen und Transitionen *In* $\rightarrow t_4 \rightarrow p_8 \rightarrow t_5 \rightarrow p_9 \rightarrow t_6 \rightarrow p_{10} \rightarrow t_7$ gebildet wird. Dieser Teilgraph bildet im wesentlichen durch die beteiligten Transitionen t_4 bis t_7 die Kernkomponenten ab, wie sie auch in Abbildung 2.6 auf Seite 26 abgebildet ist.

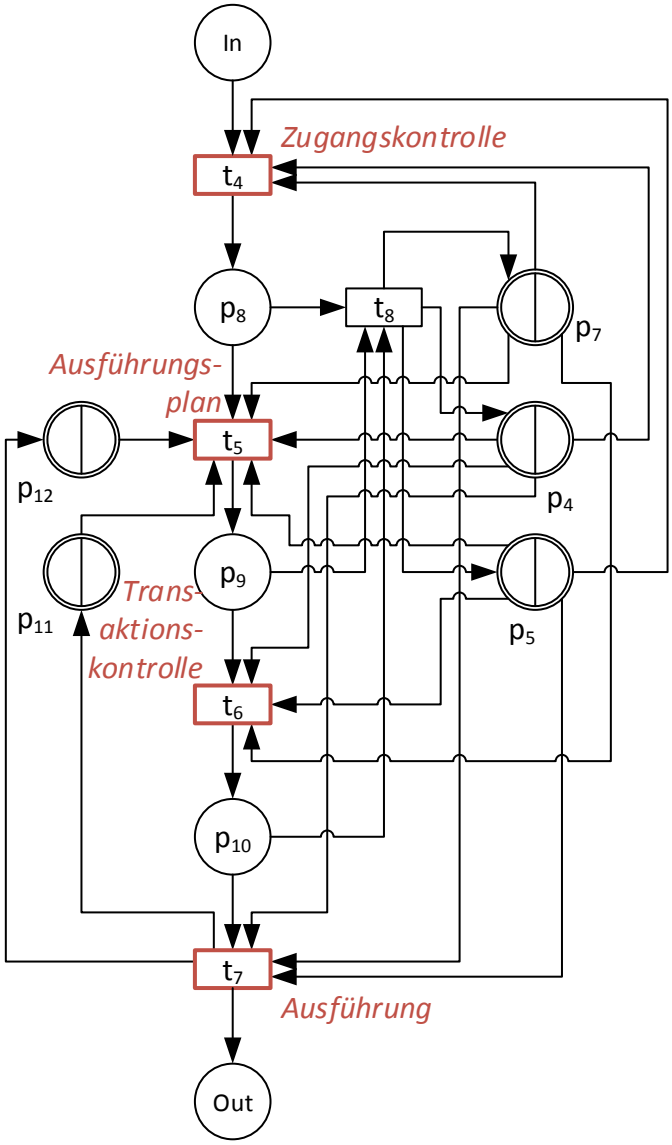


Abbildung 5.7: QPN-Modell für das Datenbanksystem. Es ist darauf ausgerichtet, als inneres Modell in Stelle p_3 im QPN-Modell für das umgebende Betriebssystem in Abbildung 5.3 zu agieren.

Die Kernkomponenten (Zugangskontrolle, Erstellung des Zugriffsplans, Transaktionskontrolle und Ausführung des Zugriffsplans), die in Abbildung 5.7 auch textuell annotiert sind, konsumieren Ressourcenmarken, die von den Ressourcenstellen des Modells für das umgebende Betriebssystem bereitgestellt werden. Diese Ressourcenstelle sind auf der rechten Seite von Abbildung 5.7 erkennbar. Es handelt sich dabei um die Stellen p_4 (Massenspeicher), p_5 (Arbeitsspeicher) und p_7 (Prozessoren). Durch den Konsum von Ressourcenmarken von diesen Stellen soll der Konsum

von Betriebsmitteln modelliert werden, wenn Datenbankoperationen anhand der vier Kernkomponenten verarbeitet werden. Es besteht auch die Möglichkeit, dass Ressourcenmarken durch Transition t_8 in Abbildung 5.7 wieder in die entsprechenden Ressourcenstellen abgelegt werden. Damit wird repräsentiert, dass nach einem entsprechenden Verarbeitungsschritt der Kernkomponenten die allokierten Betriebsmittel wieder freigegeben werden können. Das entspricht somit einer Deallokation der Betriebsmittel. Zu beachten ist, dass bei Transition t_7 bewusst darauf verzichtet wurde, dass Ressourcenmarken wieder deallokiert werden können. Das basiert zum einen auf der formalen Defition eines QPNs, dass zwei Transitionen nicht direkt mit einer Kante verbunden werden dürfen. Zum anderen repräsentiert Transition t_7 die Kernkomponente des Ausführens des Zugriffsplanes. Hier werden hauptsächlich Ressourcenmarken in Form von Massenspeichermarken verarbeitet. Deren Allokation und Deallokation wird von den Transitionen t_5 und t_6 vorgenommen.

Auf der linken Seite von Abbildung 5.7 sind noch die beiden Stellen mit inneren Modell p_{11} und p_{12} zu erkennen. Sie dienen zur Repräsentation der Architektur von Datenbanksystemen. Genauer gesagt, in diesen beiden Stellen werden die eigentlichen Datenbankdateien sowie Datenbankindizes abgebildet. Auf deren Nutzungsweise wird im weiteren Verlauf dieses Textabschnittes eingegangen.

5.4.1 Generelle Arbeitsweise des Modells für das Datenbanksystem

Insbesondere der dritte Punkt der bereits beschriebenen Modellierungsziele auf Seite 142 zeigt bereits, wie das Modell für das Datenbanksystem benutzt werden soll: die Simulation eines Benchmarks für Datenbanksysteme. Prinzipiell kann das Modell für das Datenbanksystem auch für andere modellierte Datenbankanwendungen verwendet werden, aber der vordergründige Zweck soll es sein, die komplette Ausführung eines Benchmarks zunächst zu modellieren und dann zu simulieren.

Betrachtet man die Erläuterung eines Benchmarks für Datenbanksysteme in Textabschnitt 3.1 ab Seite 53, so besteht ein Benchmark aus zwei Teilen: einem Datengenerator und einem Abfragegenerator. Ersterer generiert Daten von zumeist einstellbarer Größe in einem Datenmodell, das von dem Datenbanksystem unterstützt wird. Letzterer generiert Datenbankabfragen für die generierten Daten. Dabei werden Datenbankoperationen generiert, die entweder die Datenbasis modifizieren (Einfüge-, Lösch- und Änderungsoperationen) oder Leseoperationen auf der Datenbasis durchführen.

Die generelle Arbeitsweise des Modells für das Datenbanksystem auf Abbildung 5.7 bedingt auch die Nutzung des Modells für das umgebende Betriebssystem, wie es in Abbildung 5.3 dargestellt ist. Zur schrittweisen Erläuterung der Arbeitsweise beider Modelle sei im folgenden ein spezifisches Datenbanksystem und eine technische Plattform gegeben, auf der das Datenbanksystem ausgeführt werden kann. Zusätzlich ist ein Benchmark gegeben, der für dieses Datenbanksystem geeignet ist. Dann ergibt sich folgende Arbeitsweise:

1. Der Datengenerator des Benchmarks wird zunächst analysiert, nach welchem Datenmodell Daten generiert werden und welche Datenbankstrukturen (zum Beispiel Anzahl und Struktur von Tabellen, Datenbankindizes) dafür festgelegt wurden. Danach wird festgelegt, wie die generierten Daten als Multimenge mit farbigen Marken repräsentiert werden können, wobei durchaus Submarken gemäß Textabschnitt 5.2.1 genutzt werden können. Diese Repräsentation als farbige Marken sei als Multimenge D bezeichnet, wobei die in D enthaltenen Marken *Datenmarken* genannt werden.

Aus der Analyse der Datenbankstrukturen ergeben sich die inneren Modelle für die Stellen p_{11} (Datenbankdateien) und p_{12} (Datenbankindizes) im Modell für das Datenbanksystem in Abbildung 5.7. Sieht zum Beispiel der Datengenerator des Benchmarks die Generierung von Daten laut dem relationalen Datenmodell für sieben Tabellen vor, so kann das innere Modell von Stelle p_{11} aus sieben Stellen bestehen, die die sieben Tabellen repräsentieren sollen. Analog kann als inneres Modell für Stelle p_{12} drei Stellen enthalten, die drei Datenbankindizes repräsentieren, sofern der Datengenerator des Benchmarks die drei Datenbankindizes vorsieht.
Insgesamt kann dieser erste Schritt als allgemeine Vorbereitung beziehungsweise Anpassung des Modells für das Datenbanksystem auf die festgelegte Datenbasis des Benchmarks angesehen werden. Es sei auch bemerkt, dass dieser Schritt nicht verallgemeinert werden kann. Das bedeutet, dass ein allgemeines automatisiertes Verfahren schwierig wäre, um alle realen Datenbanksysteme und Benchmarks einzubeziehen.

2. Im Modell für das umgebende Betriebssystem werden die inneren Modelle für den Haupt- und Auslagerungsspeicher, den Massenspeicher und für die Prozessoren modelliert, sodass sie die verwendete technische Plattform repräsentieren. Die Arbeitsweise der genannten inneren Modelle und des Modells für das umgebende Betriebssystem sind in vorherigen Textabschnitt 5.3 beschrieben. Zudem werden auch die Transitionsregeln und insbesondere die Transitionszeiten dergestalt festgelegt, dass die Allokation und Deallokation von Ressourcenmarken möglichst realistisch ist. Das bedeutet, dass die technischen Komponenten der verwendeten technischen Plattform möglichst realitätsgetreu bezüglich Kapazität, Zugriffszeiten und Datendurchsatz nachgestellt werden. Schlussendlich werden die Ressourcenstellen initial mit Ressourcenmarken belegt.
   Zusammengefasst kann dieser Schritt analog zum ersten Schritt als allgemeine Vorbereitung beziehungsweise Anpassung des Modells für das umgebende Betriebssystem auf die verwendete technische Plattform angesehen werden. Auch hier ergibt sich keine Verallgemeinerung oder ein Automatismus, um diesen Schritt komfortabel zu gestalten.

3. Im Modell für das Datenbanksystem in Abbildung 5.7 werden die Transitionsregeln inklusive der Transitionszeiten der Transition t_4 bis t_8 festgelegt. Dieser Schritt ist sehr aufwendig, denn durch diese Transitionsregeln wird die Architektur des gegebenen Datenbanksystem modelliert. Genauer gesagt, es wird festgelegt, wie die Datenmarken der Multimenge D aus dem ersten Schritt durch das Modell für das Datenbanksystem verarbeitet werden soll. Prinzipiell wird dadurch der Datenfluss der Datenmarken im Modell für das Datenbanksystem beschrieben. Der Datenfluss wird im Besonderen durch die Transition t_4 bis t_8 beeinflusst. Diese Transitionen konsumieren die Datenmarken und konsumieren dabei zusätzlich Ressourcenmarken, um die Verarbeitungsschritte durch realistischen Betriebsmittelverbrauch zu repräsentieren. Das erfordert natürlich eine genaue Kenntnis der internen Abläufe sowie des zeitlichen Verhalten des gegebenen Datenbanksystems und unter Umständen ein Studium der Quelltexte des Datenbanksystems, sofern sie öffentlich einsehbar sind.
   Durch den Datenfluss der Datenmarken aus der Multimenge D soll folgende Ziele erreicht werden:

   - Die generierten Daten des Generators, repräsentiert als Datenmarken, sollen auf dem Massenspeicher persistiert werden. Dazu werden Datenmarken sowie Massenspeichermarken konsumiert und im inneren Modell von Stelle p_{11} verzeichnet und gemäß des Textabschnittes 5.2.2 miteinander assoziiert. Damit wird modelliert, welche Datenmarke auf welcher Massenspeichermarke “gespeichert” wird.

- Gleichzeitig sollen die Datenbankindizes, gegeben durch den ersten Schritt und repräsentiert durch das innere Modell in Stelle p_{12}, gepflegt werden. Das bedeutet hinsichtlich der Datenmarken, dass zur Modellierung, Erstellung und Pflege der Datenbankindizes Massenspeichermarken konsumiert werden. Analog wird auch hier eine Assoziation geschaffen, welche Datenmarke auf welchem der Massenspeichermarken gemäß der Indexstruktur "gespeichert" wurde.

Die Bedeutung der Transitionsregeln und Transitionszeiten der Transitionen innerhalb Modells für das Datenbanksystems muss an dieser Stelle besonders hervorgehoben werden, denn sie bilden sozusagen das Herzstück der gesamten Modellierung in Hinblick auf die Simulation der Performance, des Energieverbrauches und -effizienz eines gegebenen Datenbanksystems, einer technischen Plattform und der modellierten Datenbankanwendungen in Form eines Benchmarks. Dementsprechend wird in den beiden nachfolgenden Textabschnitten der Datenfluss der Datenmarken gesondert erläutert.

4. Sofern alle bisherigen Schritte erfolgreich durchgeführt wurden, kann eine Simulation gestartet werden. Damit wird erreicht, dass das Einfügen der generierten Daten des Datengenerators in die Datenbankstrukturen des Datenbanksystems simuliert wird. Es wird somit die Datenbasis geschaffen. In den Modellen für das umgebende Betriebssystem und für das Datenbanksystem sowie deren inneren Modellen äußert sich das darin, dass die Datenmarken aus der Multimenge D komplett verarbeitet wurden. Dabei wurden Massenspeichermarken verbraucht, um die modellierten Datenbankstrukturen aufzubauen (Datenbanktabellen und Datenbankindizes). Für den nächsten Schritt werden die Markenbelegungen des inneren Modellen für den Massenspeicher (Stelle p_4 in Abbildung 5.3 auf Seite 132 sowie für die Datenbankdateien und Datenbankindizes (Stellen p_{11} und p_{12} in Abbildung 5.7 auf Seite 143) gesichert.
5. Das Modell für das umgebende Betriebssystem und für das Datenbanksystem sowie alle inneren Modelle werden zurückgesetzt. Damit entsteht wieder der Zustand, wie er vor der durchgeführten Simulation des letzten Schrittes vorlag. Es wird aber die vorher gesicherte Markenbelegung wieder eingespielt.

5.4.2 Modifikation von Datenbankinhalten

Ausgehend von der generellen Arbeitsweise des vorhergehenden Textabschnittes soll in diesem Textabschnitt die Modifikation von Datenbankinhalten näher betrachtet werden. Darunter werden alle Datenbankoperationen verstanden, die die Datenbasis verändern. Im wesentlichen sind das also Einfüge-, Lösch- und Änderungsoperationen.

Im folgenden wird eine Multimenge D_1 von Datenmarken angenommen, die die genannten Operationen repräsentieren. D_1 kann am beschriebenen Beispiel der generellen Arbeitsweise mit der Multimenge D gleichgesetzt werden, womit D_1 nur eine Serie von Datenbankoperationen repräsentiert, die Einfügeoperationen enthält. Durch diese Operationen erhält man nach Ausführung der generellen Arbeitsweise die beschriebene Datenbasis. D_1 kann aber auch eine Mischung aus den oben beschriebenen Datenbankoperationen enthalten. Eine solche Mischung erhält man durch Analyse des Abfragegenerators des gegebenen Benchmarks, um zum Beispiel ein *OLTP*-Szenario hinsichtlich der Performance zu evaluieren. Die Datenmarken, die in D_1 enthalten sind, können durch verschiedenen Farben unterscheidbar gehalten werden, um die unterschiedlichen Operationsarten (Einfügen, Löschen und Änderung) zu unterscheiden.

Die Arbeitsweise äußert sich in folgenden Schritten:

1. Prozessstelle p_1 im Modell für das umgebende Betriebssystem auf Abbildung 5.3 enthält initial eine Prozessmarke $g \in G$ (siehe Tabelle 5.1 auf Seite 133). Diese Prozessmarke enthält nur die Multimenge D_1, alle anderen Komponenten werden mit der leeren Menge belegt.
2. Die Prozessmarke g wird von der Transition t_1 im Modell für das umgebende Betriebssystem in Abbildung 5.3 konsumiert. Dabei kommt eine Transitionsregel zum Einsatz, die Ressourcenmarken aus den Ressourcenstellen p_4, p_5 und p_7 allokiert und eine neue Prozessmarke $g' \in G$ in Prozessstelle p_2 ablegt. Die Prozessmarke g' enthält neben den allokierten Ressourcenmarken auch die Multimenge D_1. Durch diesen Schritt wird repräsentiert, dass eine Instanz des Datenbanksystems gestartet wurde, wobei Betriebsmittel konsumiert werden.
3. Die Prozessmarke g' wird durch die Prozessstelle p_2 verarbeitet, wobei ein Prozess-Scheduler für den Fall eingesetzt wird, dass mehrere Prozessmarken in p_2 abgelegt wurden. Transition t_2 konsumiert die Prozessmarke g', erzeugt sie neu und legt sie in Prozessstelle p_3 ab. Eine Kopie erreicht das innere Modell von Prozessstelle p_3, also das Modell für das Datenbanksystem, wie es in Abbildung 5.7 auf Seite 143 dargestellt ist.
4. Im Modell des Datenbanksystems in Abbildung 5.7 wird Prozessmarke g' in der Stelle *In* abgelegt. Transition t_4 konsumiert sie und die enthaltenen Datenmarken werden extrahiert. Diese Datenmarken werden in Stelle p_8 abgelegt.
5. Transition t_5 konsumiert die Datenmarken aus Stelle p_8. Dabei kommen unterschiedliche Transitionsregeln zum Einsatz, die die Färbung der Datenmarken beachten. Das bedeutet genauer:
   - Bei Datenmarken, deren Färbung eine Einfügeoperation repräsentieren, wird geprüft, ob für die jeweilige Datenmarke bereits eine Assoziation mit einer Massenspeichermarke in Stelle p_{11} existiert. Falls ja, bedeutet es, dass der Datensatz, den die Datenmarke repräsentiert, bereits verzeichnet ist. In diesem Falle wird die Datenmarke verworfen. Falls keine solche Assoziation vorliegt, wird die Datenmarke neu erzeugt und in Stelle p_9 abgelegt.
   - Bei Datenmarken, deren Färbung eine Lösch- oder Änderungungsoperationen repräsentieren, wird geprüft, ob für die jeweilige Datenmarke eine Assoziation in den Stellen p_{11} und p_{12} vorliegt. Damit wird modelliert, ob der Datensatz, der durch die Datenmarke repräsentiert wird, bereits verzeichnet ist. Falls ja, werden die entsprechenden Assoziationsmarken bei einer Löschoperation konsumiert. Bei einer Änderungsoperation wird nur die Assoziationsmarke von Stelle p_{12} konsumiert. Die jeweilige Datenmarke wird neu erzeugt und in Stelle p_9 abgelegt.
6. Die Datenmarken, die in Stelle p_9 abgelegt wurden, werden durch Transition t_6 konsumiert, neu erstellt und in Stelle p_{10} abgelegt. Dabei nutzt diese Transition eine Liste von Transitionsregeln, wobei diese so angeordnet sind, dass manche eine höhere Priorität genießen als andere (siehe Textabschnitt 5.2.3). Damit kann geregelt werden, dass zum Beispiel Datenmarken, deren Färbung eine Einfügeoperation repräsentiert, vor Datenmarken konsumiert und in Stelle p_{10} abgelegt werden, deren Färbung eine andere Operation repräsentiert.
7. Transition t_7 konsumiert alle Datenmarken, die in Stelle p_{10} abgelegt wurden und nutzt dabei Transitionsregeln, die folgende Aktionen anhand der Färbung der jeweiligen Datenmarke ausführt:

- Eine Datenmarke, dessen Färbung eine Einfügeoperation repräsentiert, wird neu erzeugt und in die Stellen p_{11} und p_{12} abgelegt. Diese Datenmarken erreichen somit die inneren Modelle dieser Stellen, wobei die Datenmarke mit allokierten Massenspeichermarken assoziiert wird. Damit wird modelliert, dass der Datensatz, der durch die Datenmarke repräsentiert wird, in das Datenbanksystem gespeichert wird (inneres Modell von Stelle p_{11}) und gegebenfalls die Datenbankindizes aktualisiert werden (inneres Modell von Stelle p_{12}).
- Eine Datenmarke, dessen Färbung eine Änderungsoperation repräsentiert, wird neu erzeugt und in die Stelle p_{12} abgelegt. Durch das dortige innere Modell wird die Datenmarke mit einer allokierten Massenspeichermarke assoziiert. Damit wird modelliert, das der Datensatz, der durch die Datenmarke repräsentiert wird, in der Datenbank aktualisiert wurde. Dadurch entstehen unter Umständen Änderungen an den Datenbankindizes.

8. Die Prozessmarke g' wird im Modell für das umgebende Betriebssystem in Abbildung 5.3 auf Seite 132 von Transition t_3 konsumiert. Bis auf die enthaltene Multimenge D_1 werden alle Ressourcenmarke von g' neu erzeugt und in die entsprechenden Ressourcenstellen p_4, p_5 und p_7 abgelegt.

Es sei angemerkt, dass in den beschriebenen Schritten 4 bis 7 auch Ressourcenmarken konsumiert werden. Aus Gründen der Übersichtlichkeit wurden deren Allokation und Deallokation in den Schritten 4 bis 7 nicht beschrieben.

5.4.3 Lesen von Datenbankinhalten

Analog zur Beschreibung der Arbeitsweise für die Modifikation von Datenbankinhalten wird in diesem Textabschnitt das Lesen von Datenbankinhalten genauer erläutert. Unter diesem Ausdruck werden alle Datenbankoperationen verstanden, die Datenbankinhalte anhand von Such- und Filterkriterien abrufen. Sie modifizieren dadurch die Datenbasis nicht. Im allgemeinen werden dadurch Datenbankoperationen verstanden, wie sie in *OLAP*-Szenarios genutzt werden.

Im folgenden wird davon ausgegangen, dass bereits eine Datenbasis erstellt wurde, zum Beispiel durch eine Simulation der Modelle für das umgebende Betriebssystem und für das Datenbanksystem, wobei die Arbeitsweise in den beiden verhergehende Textabschnitten genutzt wird. Das bedeutet, dass die Multimenge D_1 (beziehungsweise D) mit den enthaltenen Datenmarken benutzt wurde, um die Datenbasis aufgrund des Datengenerators des gegegeben Benchmarks zu erstellen.

Es sei nun eine Multimenge D_2 mit Datenmarken angenommen, wobei D_2 eine echte Teilmenge von D_1 darstellt ($D_2 \subseteq D_1$). Dabei sei angemerkt, dass die Datenmarken der Multimenge D_1 die Datensätze repräsentiert haben, die in das Datenbanksystem per Simulation eingefügt wurden. Die Datenmarken in der Teilmenge D_2 repräsentieren die Datensätze, die gelesen werden sollen. Es stellt sich die Frage, wie die Teilmenge D_2 erstellt werden kann. Dazu muss der Abfragegenerator des Benchmarks analysiert werden. Genauer gesagt müssen die einzelnen generierten Abfragen analysiert werden, welche Datensätze gelesen werden müssen.

Es sei an dieser Stelle angemerkt, dass diese Analyse durchaus zeitintensiv ist. Zudem bedingt es eine sehr umfassende und präzise Kenntnis der Datenbanksystem-Interna, um die generierten Abfragen statisch nachzuvollziehen. Allerdings muss auch angemerkt werden, dass in den überwiegenden Fällen die Dokumentation des Benchmarks zu Rate gezogen werden kann, um detaillierte Informationen zu den generierten Abfragen zu erhalten.

Im Bezug auf die Multimenge D_2 von Datenmarken ergibt sich folgende Arbeitsweise:

1. Die ersten vier Schritte aus der Arbeitsweise für die Modifikation von Datenbankinhalten werden ausgeführt, da sie auch für das Lesen von Datenbankinhalten gelten (siehe dazu den vorherigen Textabschnitt). Allerdings wird anstatt der Multimenge D_1 die Multimenge D_2 in den Prozessmarken g und g' genutzt.
   Das bedeutet, dass initial eine Prozessmarke $g \in G$ in der Prozessstelle p_1 im Modell für das umgebende Betriebssystem existiert, wobei D_2 eine Komponente von g ist. Führt man die ersten vier Schritte der Arbeitsweise für die Modifikation von Datenbankinhalten aus, wie sie im vorherigen Textabschnitt beschrieben sind, so gelangt die Prozessmarke in das innere Modell von Prozessstelle p_3, in der das Modell für das Datenmodell ausgeführt wird.
2. Die Datenmarken der Multimenge D_2 befinden sich in Stelle p_8 im Modell für das Datenbanksystem, das in Abbildung 5.7 auf Seite 143 dargestellt ist. Transition t_4 konsumiert die Datenmarken über Transitionsregeln. Dabei wird für die jeweilige Datenmarke die Assoziationsmarken aus den Stellen p_{11} und p_{12} dafür genutzt, die assoziierten Massenspeichermarken zu ermitteln und aus der Ressourcenstelle p_4 zu konsumieren. Damit wird modelliert, dass der Datensatz, der durch die jeweilige Datenmarke repräsentiert wird, vom Massenspeicher gelesen wird. Zugleich wird modelliert, wie die Datenbankdateien und -indizes konsultiert werden, um zu ermitteln, von welcher Stelle des Massenspeichers gelesen werden muss, um den Datensatz zu erhalten.
   Die konsumierte Massenspeichermarke wird erneut erzeugt und in Stelle p_9 abgelegt.
3. Transition t_6 konsumiert die Massenspeichermarke, erzeugt zwei Kopien und legt sie beide in Stelle p_{10} ab. Dabei kann für diese Transition eine Liste mit Transitionsregeln genutzt werden, wobei manche Transitionsregeln eine höhere Priorität bei der Ausführung genießen als andere. Diese Liste ist optional, aber man kann dadurch das Verhalten des Datenbanksystem modellieren, Leseoperationen auf die Datenbankdateien und dadurch auf den Massenspeicher zum Zwecke der Optimierung neu anzuordnen.
4. Transition t_8 konsumiert eine der beiden Massenspeichermarken aus Stelle p_{10}, erzeugt sie neu und legt sie wieder in Ressourcenstelle p_4 ab. Gleichzeitig konsumiert Transition t_7 die andere Massenspeichermarke ohne eine neue zu erzeugen und sie in eine andere Stelle abzulegen.

Auch hier sei es angemerkt, dass in den beschriebenen Schritten 2 und 3 auch zusätzliche Ressourcenmarken konsumiert werden, zum Beispiel Arbeitsspeicher- und Prozessorenmarken. Aus Gründen der Übersichtlichkeit wurden deren Allokation und Deallokation in den Schritten 2 und 3 nicht beschrieben.

5.5 Modell für das Clusternetzwerk

Die in den verhergehenden Textabschnitten 5.3 und 5.4 beschriebenden Modelle für das umgebende Betriebssystem und für das Datenbanksystem reichen aus, um ein zentral betriebenes Datenbanksystem nachzustellen und zu simulieren. Für die Modellierung eines verteilten Datenbanksystems sind diese Modelle aber nicht vollständig geeignet.

Allerdings genügen einige wenige Modifikationen an den bestehenden Modellen sowie eine geringfügige Änderung an der Arbeitsweise, um auch verteilte Datenbanksysteme im Bezug auf Performance, Energieverbrauch und -effizienz modellieren und simulieren zu können.

Aus Textabschnitt 2.4 ist bekannt, dass verteilte Datenbanksysteme einen Computer-Cluster nutzen, um auf den einzelnen Clusterknoten jeweils eine Instanz des Datenbanksystems auszuführen. Die Clusterknoten sind dabei völlig unabhängig voneinander. Das bedeutet, jeder Clusterknoten führt sein eigenes Betriebssystem aus und verwaltet exklusiv seine lokalen Betriebsmittel. Über das Computernetzwerk, dass die Clusterknoten miteinander verbindet, können die Instanzen des verteilten Datenbanksystems kommunizieren und Daten austauschen.

Aus diesen Aussagen lassen sich drei Modellierungsziele extrahieren, um verteilte Datenbanksysteme anhand von QPN-Modellen nachstellen zu können:

1. Modellierung eines Computernetzwerkes
2. Erweiterung des Modelles für das umgebende Betriebssystem im Hinblick auf die Nutzung des Modelles für das Computernetzwerk
3. Erweiterung des Modelles für das Datenbanksystem im Hinblick auf die Nutzung des Modelles für das Computernetzwerk

Die beiden letzten Punkte werden gesondert in den nachfolgenden Textabschnitten 5.6 und 5.7 betrachtet.

Das erste Modellierungsziel besagt, dass ein Computernetzwerk mit Hilfe von QPNs modelliert werden soll. Dabei stellt sich die Frage, ob dies überhaupt möglich ist und wenn ja, auch praktikabel ist. Diese Frage resultiert aus den Aussagen von Textabschnitt 2.4, in der die verschiedenen existierenden Netzwerktopologien und Übertragungstechniken beschrieben wurden, die bei verteilten Datenbanksystemen zum Einsatz kommen. Hier ergibt sich eine Vielzahl von Kombinationsmöglichkeiten der eingesetzten netzwerktechnischen Komponenten, etwa Hubs und Switche. Diese Vielzahl von Kombinationsmöglichkeiten müssen durch ein geeignetes QPN-Modell unterstützt werden.

Eine Literaturrecherche zur Beantwortung der obigen Frage ergab, dass bereits umfangreiche Untersuchungen und Studien zur Modellierung von Computernetzwerken publiziert wurden. So haben *Chang et al.* und *Rygielski et al.* gezeigt, dass es generell möglich ist, Computernetzwerke als QPN-Modell darzustellen [CPH10, RZK13]. Die Kernaussagen beider Publikationen sind, dass Computernetzwerke formal durch QPNs beschrieben und auch praktikabel im Sinne einer Simulation eingesetzt werden können. Insbesondere letztgenannter Punkt bedeutet, dass der Datenaustausch in einem Computernetzwerk als Datenfluss in einem QPN-Modell simuliert werden kann. Das eröffnet die Möglichkeit, anhand von Simulationen Aussagen zur Performance und Datendurchsatz eines Computernetzwerkes zu machen.

Rygielski und Kounev haben die beiden oben genannten Publikationen als Basis genutzt, um weitere Erkenntnisse hinsichtlich der Modellierung von Computernetzwerken zu gewinnen. In ihrer Publikation konnten sie zeigen, dass jedwedes QPN-Modell, das ein Computernetzwerk beschreibt, im wesentlichen einen gerichteten Graph darstellt, der aus inneren beziehungsweise Zwischenknoten sowie Endknoten besteht [RK14]. Die Endknoten bezeichnen hierbei die Kommunikationspartner, sozusagen die Endgeräte des Computernetzwerkes wie etwa Server, während die Zwischenknoten die eingesetzte Netzwerktechnik, zum Beispiel Hubs und Switche, repräsentieren.

In Abbildung 5.8 auf Seite 151 ist ein sehr einfaches QPN-Modell nach [RK14] dargestellt, das ein Computernetzwerk repräsentieren soll. Das Computernetzwerk besteht aus drei Servern, die über einen zentralen Switch Daten austauschen können. Nach *Rygielski und Kounev* sind sie somit Endknoten und werden im QPN-Modell auf Abbildung 5.8 auch so modelliert (Stellen mit den

textuellen Annotationen “Endknoten 1” bis “Endknoten 3”). Der genutzte Switch ist als Komponente der Netzwerktechnik ein Zwischenknoten und verbindet die Endknoten miteinander. Die Kommunikation der drei modellierten Server erfolgt in Abbildung 5.8 durch den Austausch von Marken.

In Abbildung 5.8 fällt auf, dass die Stellen, die Endknoten repräsentieren sollen, jeweils durch zwei Transitionen mit der Stelle, die einen Zwischenknoten repräsentiert, verbunden sind. Zudem fällt auf, dass die gerichteten Kanten dabei entgegengesetzt modelliert sind. Nach [RK14] wird dadurch der Up- und Download der Endknoten modelliert. Beide Aspekte sind aus Sicht des Endknoten definiert. Das bedeutet, dass eine gerichtete Kante, die zu einem Endknoten führt, den Downloadpfad des Endknoten bezeichnet. Analog ist eine gerichtete Kante, die von einem Endknoten wegführt, der Uploadpfad. Die gezeigten Transitionen erfüllen aber auch einen weiteren wichtigen Zweck. Durch entsprechende Transitionsregeln und insbesondere Transitionszeiten kann der Datenfluss in Form von Marken zwischen den Endknoten über die Zwischenknoten beeinflusst werden. Durch die Transitionsregeln kann zum Beispiel festgelegt werden, welche Kommunikationspartner miteinander Daten in Form von Marken austauschen dürfen. Durch die Transitionszeiten kann zum Beispiel für eine spätere Simulation der maximale Datendurchsatz festgelegt werden [RZK13].

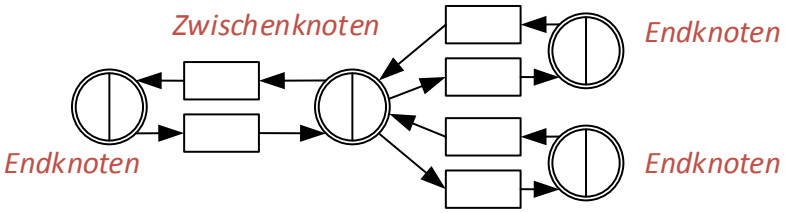


Abbildung 5.8: Generelles QPN-Modell eines Computernetzwerkes nach [RK14]

Zudem fällt in Abbildung 5.8 auf, dass alle gezeigten Stellen als Stelle mit inneren Modell modelliert sind. Damit wird die Möglichkeit geschaffen, in welcher Form der Datenfluss modelliert werden soll. Genauer gesagt, das QPN-Modell in Abbildung 5.8 beschreibt nur das Computernetzwerk, also welche Kommunikationspartner existieren und welche Kommunikationspfade dabei genutzt werden können. Die Kommunikation wird wie bereits beschrieben in Form des Datenflusses von Marken modelliert. Allerdings beschreibt das QPN-Modell auf Abbildung 5.8 nicht, wie die Marken vom Initiator eines Kommunikationspfades generiert und vom Ziel des Kommunikationspfades wieder verarbeitet werden. Diese Aufgabe fällt den inneren Modellen der Stellen in Abbildung 5.8 zu. Es sei angemerkt, dass dies kein Nachteil ist, sondern eine bewusste Entscheidung hinsichtlich der Modellierung anhand von QPN-Modellen [RK14]. Hierbei werden bewusst hierarchisch organisierte QPN-Modelle genutzt, um eine strikte “Aufgabenteilung” wie beschrieben herbeizuführen.

Dieser Ansatz ist sehr interessant und kann im Bezug auf die Modellierung von verteilten Datenbanksystemen genutzt werden. Zur Illustration sei ein verteiltes Datenbanksystem gegeben, dass einen Computercluster nutzt, der aus vier Clusterknoten besteht. Die Clusterknoten werden durch einen Switch miteinander verbunden und ermöglichen den Datenaustausch zwischen den einzelnen Instanzen des verteilten Datenbanksystems.

Auf Basis des QPN-Modells von Abbildung 5.8 ergibt sich aufgrund dieser Beschreibung ein QPN-Modell, wie es in Abbildung 5.9 dargestellt ist. In Abbildung repräsentieren die Stellen mit inneren Modell s_1 bis s_4 die Clusterknoten und die Stelle *Sw* den Switch.

Wie eingangs beschrieben führen die Clusterknoten jeweils ein Betriebssystem aus, das exklusiv die lokalen Betriebsmittel verwaltet. Das legt nahe, dass als inneres Modell der Stellen s_1 bis s_4 in Abbildung 5.9 jeweils eine Instanz des Modelles für das umgebende Betriebssystem genutzt wird. Dabei ist jede Instanz unabhängig voneinander. Das bedeutet, dass die Ressourcenmarken einer Instanz auch nur innerhalb der Instanz verwendet werden können[11].

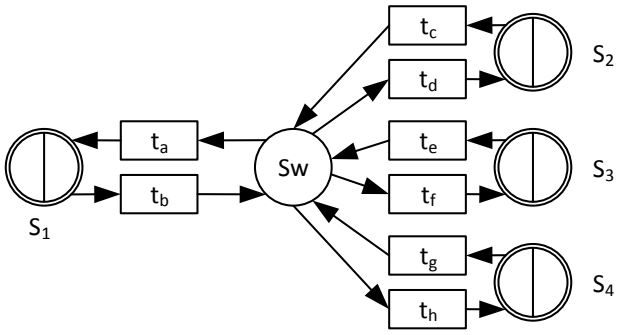


Abbildung 5.9: QPN-Modell eines Computernetzwerkes für ein verteiltes Datenbanksystem. In den Stellen s_1 bis s_4 wird jeweils als inneres Modell die Modelle des umgebenden Betriebssystems und des Datenbanksystems ausgeführt.

In Textabschnitt 5.4 wurde beschrieben, dass das Modell für das umgebende Betriebssystem das Modell für das Datenbanksystem ausführt. Das geschieht in Stelle p_3 auf Abbildung 5.3, wobei für das innere Modell von Stelle p_3 das Modell für das Datenbanksystem vorgesehen ist. Bezogen auf Abbildung 5.9 bedeutet dies, dass alle Instanzen des Modells für das umgebende Betriebssystem in den Stellen s_1 bis s_4 jeweils auch eine Instanz des Modells für das Datenbanksystem enthalten. Insgesamt wird damit modelliert, dass auf den Clusterknoten jeweils eine Instanz des verteilten Datenbanksystems ausgeführt wird, wobei das lokale Betriebssystem der Clusterknoten genutzt wird.

Insgesamt entsteht damit ein hierarchisches QPN-Modell mit drei Ebenen, wie es in Abbildung 5.10 auf Seite 153 als Baumstruktur dargestellt ist. Auf erster, oberster Ebene steht das QPN-Modell für das Computernetzwerk, wie es in Abbildung 5.9 dargestellt wird. Dessen Stellen mit inneren Modell s_1 bis s_4 enthalten als inneres Modell jeweils eine Instanz des Modells für das umgebende Betriebssystem, wie es in Abbildung 5.3 auf Seite 132 dargestellt ist. Diese Instanzen bilden die zweite, mittlere Ebene der Hierarchie. Jede dieser Instanzen führt als inneres Modell in Stelle p_3 eine Instanz des Modells eines Datenbanksystems aus, wie es in Abbildung 5.7 auf Seite 143 zu sehen ist. Diese Instanzen bilden die unterste, letzte Hierarchiestufe. Damit setzt sich das gesamte QPN-Modell aus 17 einzelnen QPN-Modellen zusammen (1×Computernetzwerk + 4×Instanzen des Modelles für das umgebende Betriebssystem + 4×Instanzen für das Datenbanksystem).

[11] Dies ist nur eine Festlegung. Die formale Defintion eines QPN erlaubt die Verwendung aller Marken in allen Modellen sowie inneren Modellen.

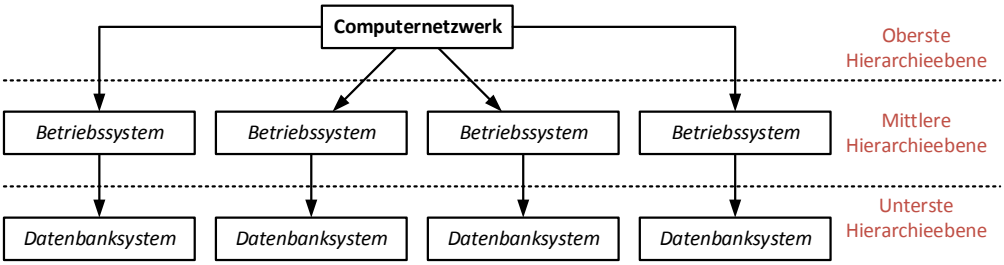


Abbildung 5.10: Hierarchie der QPN-Modelle für ein verteiltes Datenbanksystem

Im Umgang mit diesem hierarchischen QPN-Modell werden aber zusätzliche Festlegungen getroffen. Ressourcenmarken der Instanzen können auch nur in diesen genutzt werden. Anhand von Abbildung 5.10 äußert sich dieses Vorgehen darin, dass Ressourcenmarken in den einzelnen Ästen der gezeigten Baumstruktur und auch dort nur in der mittleren und untersten Ebene genutzt werden dürfen. Zugleich wird auch eine neue Art Ressourcenmarken eingeführt. Sie werden im folgenden als *Netzwerkmarken* bezeichnet und haben den Zweck, den Datenfluss zu modellieren, der beim Datenaustausch in einem Computernetzwerk beziehungsweise Cluster entsteht.

5.6 Erweitertes Modell für das umgebende Betriebssystem

Im vorhergehenden Textabschnitt wurde das hierarchische Modell für ein verteiltes Datenbanksystem, das in Abbildung 5.9 dargestellt ist, zusammen mit einem neuen Ressourcenmarkentyp, den Netzwerkmarken, eingeführt und erläutert.

Betrachtet man das Modell des umgebenden Betriebssystems in Abbildung 5.3 auf Seite 132 sowie die Erläuterungen in Textabschnitt 5.3 dazu, so fällt auf, dass es zu jeder Ressourcenmarke auch eine dazugehörige Ressourcenstelle gibt. Das ist für Netzwerkmarken nicht der Fall. Dementsprechend muss das Modell des umgebenden Betriebssystems um eine solche Ressourcenstelle erweitert werden.

Diese Erweiterung ist in Abbildung 5.11 auf Seite 154 dargestellt, wobei die Änderungen gegenüber dem ursprünglichen Modell für das umgebende Betriebssystem in blau hervorgehoben sind. Die Änderungen bestehen darin, dass eine neue Ressourcenstelle geschaffen wurde, die in Abbildung 5.11 mit p_{13} bezeichnet wird. Da laut der formalen Defintion von QPNs zwei Stellen nicht direkt miteinander verbunden werden dürfen, wird Transition t_{10} dafür genutzt, die Stelle p_{13} mit Stelle p_1 zu verbinden. Das neu entstandene QPN-Modell, das in Abbildung 5.11 gezeigt ist, wird im folgenden als *erweitertes Modell für das umgebende Betriebssystem* bezeichnet, um es vom ursprünglichen Modell abzugrenzen. Insgesamt dienen die Änderungen dazu, eingehende Netzwerkmarken in der Ressourcenstelle p_{13} zu speichern. Transition t_{10} ist dann dafür zuständig, diese Marken zu verarbeiten und eine Prozessmarke zu erstellen. Somit dienen die Erweiterungen in erster Linie dazu, einen Prozess im erweiterten Modell des umgebenden Betriebssystems zu starten. Das ist primär die Ausführung des Modells für das Datenbanksystems als inneres Modell in Stelle p_3.

Zusätzlich muss definiert werden, was *eingehende* und *ausgehende* Netzwerkmarken bedeuten. Dazu kann Abbildung 5.9 auf Seite 152 betrachtet werden. In dieser Abbildung wird das QPN-Modell eines Computernetzwerkes für ein verteiltes Datenbanksystems hierarchisch von oberster

Ebene aus dargestellt. Instanzen des erweiterte Modell für das umgebende Betriebssystem werden als inneres Modell in den Stellen s_1 bis s_4 ausgeführt. Daher ergibt sich die Anforderung, dass das erweiterte Modell für das umgebende Betriebssystem auch der formalen Definition eines inneren Modells gemäß Textabschnitt 2.5 genügen muss. Diese Definition sieht zwei Stellen *In* und *Out* vor, in der ein- und ausgehende Marken gespeichert werden. Stelle p_{13} im erweiterten Modell für das umgebende Betriebssystem vereinigt die Stellen *In* und *Out* in sich. Damit ist es möglich, dass eingehende Netzwerkmarken in Stelle p_{13} abgelegt werden und ausgehende Netzwerkmarken von Stelle p_{13} gleichzeitig konsumiert werden können.

Betrachtet man Abbildung 5.9 auf Seite 152, kann damit auch erklärt werden, wie die Transitionen t_a bis t_h verbunden sind, die in dieser Abbildung gezeigt werden. Die Transitionen sind mit den jeweiligen lokalen Ressourcenstellen für Netzwerkmarken p_{13} verbunden und ermöglichen somit den Austausch dieser Marken. Über Transitionregelsätze für die Transitionen t_a bis t_h wird geregelt, welche Netzwerkmarken konsumiert (ausgehende Netzmarken) und welche erzeugt werden sollen (eingehende Netzwerkmarken).

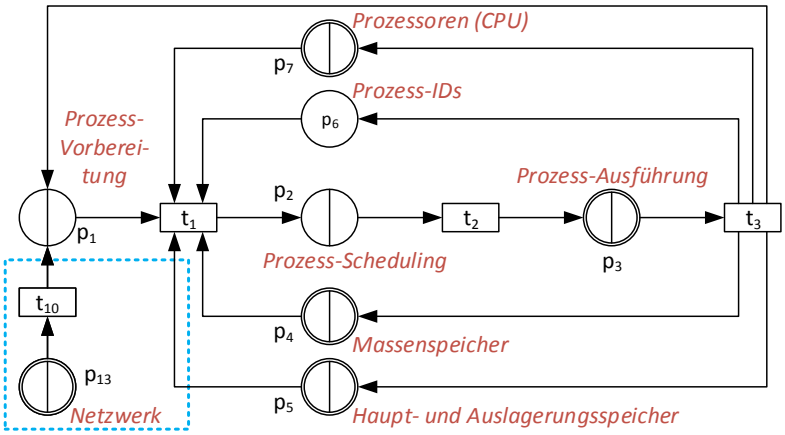


Abbildung 5.11: Erweitertes QPN-Modell für das umgebende Betriebssystem. Die Änderungen gegenüber dem QPN-Modell aus Abbildung 5.3 sind blau markiert.

5.7 Erweitertes Modell für das Datenbanksystem

Durch die Einführung der zusätzlichen Ressourcenstelle p_{13} im erweiterten Modell für das umgebende Betriebssystem in Abbildung 5.11 ist es auch notwendig, diese Ressourcestelle im Modell für das Datenbanksystem einzuführen. Es ist in Abbildung 5.7 auf Seite 143 dargestellt und in Textabschnitt 5.4 erläutert.

Die Änderung am ursprünglichen Modell für das Datenbanksystem ist in Abbildung 5.12 auf Seite 156 dargestellt, wobei sie farblich markiert sind. Die Änderung äußert sich lediglich darin, dass die Ressourcenstelle p_{13} innerhalb des Modelles für das Datenbanksystem verfügbar gemacht wird.

Es sei bemerkt, dass auf Abbildung 5.12 von der ursprünglichen Grundstruktur abgewichen wurde. Das bedeutet, dass alle Ressourcenstellen in Abbildung 5.12 auf der rechte Seite angeordnet werden. Dieses Vorgehen hätte aber die Komplexität und Lesbarkeit des abgebildeten

QPN-Modells weiter erhöht. Demzufolge wurde die Ressourcenstelle p_{13} auf der linken Seite von Abbildung 5.12 modelliert.

Zudem sei auch in Abbildung 5.12 bemerkt, dass Ressourcenstelle p_{13} im Gegensatz zu den anderen dargestellten Ressourcenstellen nicht mit allen Transitionen verbunden ist. Genauer gesagt, Ressourcenstelle p_{13} ist nur mit den Transitionen t_4 und t_7 verbunden. Der Grund dafür ist die Arbeitsweise des erweiterten Modelles für das Datenbanksystem, die in den nachfolgenden drei Textabschnitten genauer erläutert wird.

5.7.1 Generelle Arbeitsweise des erweiterten Modelles für das Datenbanksystem

Die vorangegangenen drei Textabschnitte haben erläutert, wie ein Computernetzwerk beziehungsweise Cluster modelliert werden kann. Dabei wurde auch erklärt, dass dabei ein hierarchisches QPN-Modell entsteht. Ein modellierter Clusterknoten besteht dabei immer aus einer Instanz des erweiterten Modells des umgebenden Betriebssystems, das eine Instanz des erweiterten Modells für das Datenbanksystem ausführt. Sei eine solche Instanzkombination im folgendem als *Knoteninstanz* bezeichnet.

Wie bereits beschrieben, basieren die erweiterten Modelle für das Betriebssystem und des Datenbanksystems auf den beiden "einfachen" Modellen, die in den Textabschnitten 5.3 und 5.4 beschrieben sind. Der Unterschied zwischen den originalen und den erweiterten Modellen besteht darin, dass die erweiterten Modelle einen Markenaustausch über die Modellgrenzen hinaus zulassen, womit eine gewisse Netzwerkfunktionalität nachgestellt werden kann.

Als Konsequenz dieser "Abstammung" kann man feststellen, dass *innerhalb* einer Knoteninstanz die generelle Arbeitsweise angewendet werden kann, wie sie bereits in Textabschnitt 5.4.1 ab Seite 144 beschrieben ist. Man geht also davon aus, dass sich die einzelnen Knoteninstanzen jeweils wie ein zentral betriebenes Datenbanksystem verhalten. Daraus folgt, dass jede Knoteninstanz in der Lage ist, eine Multimenge an Datenmarken zu verarbeiten. Bei einer Simulation der Knoteninstanzen werden diese unabhängig voneinander ausgeführt. Das ermöglicht es, dass jede Knoteninstanz prinzipiell eine andere Multimenge an Datenmarken als Eingabemenge erhält, um sie dann jeweils wie ein zentral betriebenes Datenbanksystem zu verarbeiten.

Dabei ist nicht ausgeschlossen, dass die Multimengen an Datenmarken, die jeweils als Eingabemenge für eine spezifische Knoteninstanz dient, selbst nur Teilmengen einer größeren Multimenge sind. Der Hintergrund ist, dass damit eine Lastverteilung sowie die Replikation modelliert werden können. Dazu stelle man sich drei Knoteninstanzen sowie eine Multimenge D mit 20 Datenmarken vor:

$$D = \{d_1, d_2, d_3, \ldots, d_{20}\}$$

Nun ließe sich die Multimenge D aufgrund der Anzahl von Knoteninstanzen in drei Teilmenge Teilmengen D_a bis D_c segmentieren:

$$D = \{\underbrace{d_1, d_2, d_3, d_4, d_5, d_6}_{D_a}, \underbrace{d_7, d_8, d_9, d_{10}, d_{11}, d_{12}, d_{13}, d_{14}}_{D_b}, \underbrace{d_{15}, d_{16}, d_{17}, d_{18}, d_{19}, d_{20}}_{D_c}\}$$

Analog kann auch eine Replikation der Elemente der Multimenge D gedacht werden. Für obiges Beispiel sei ein Replikationsfaktor von 3 angenommen. Das bedeutet, dass jedes Elemente $d \in D$ dreimal repliziert werden soll. Die daraus entstehende neue Multimenge sei mit D' bezeichnet:

$$D' = D \cup D \cup D = \{d_1, d_2, d_3, \ldots, d_{20}, d_1, d_2, d_3, \ldots, d_{20}, d_1, d_2, d_3, \ldots, d_{20}\}$$

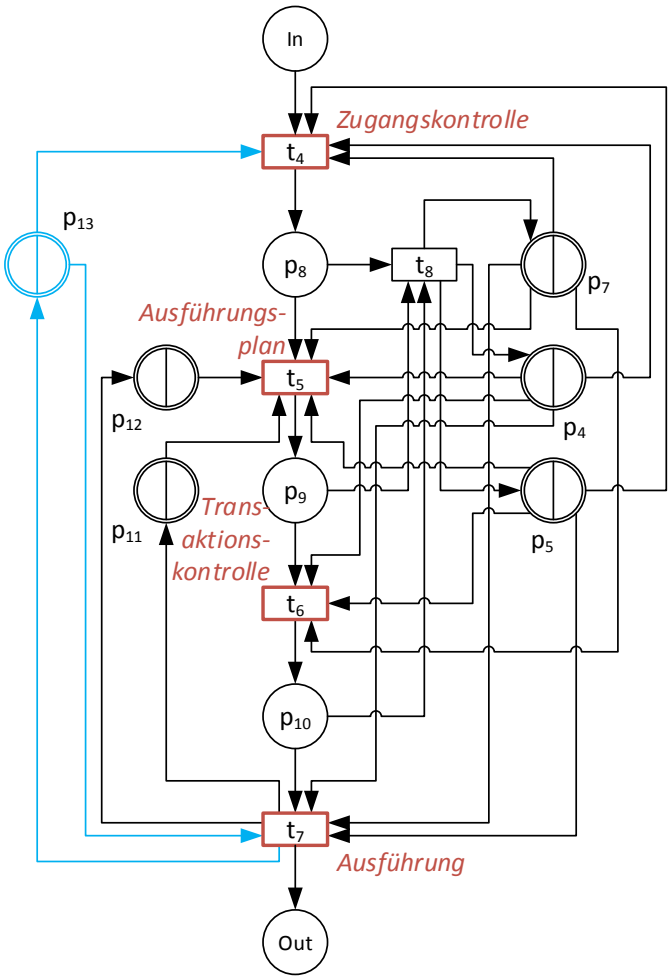


Abbildung 5.12: Erweitertes QPN-Modell für das Datenbanksystem. Die Änderungen gegenüber dem QPN-Modell aus Abbildung 5.7 sind blau markiert.

Auch die Multimenge D' kann gemäß obigen Lastverteilungsverfahren in drei Teilmengen D_d bis D_f segmentiert werden:

$$D' = \{\underbrace{d_1, d_2, d_3, \ldots, d_{20}}_{D_d}, \underbrace{d_1, d_2, d_3, \ldots, d_{20}}_{D_e}, \underbrace{d_1, d_2, d_3, \ldots, d_{20}}_{D_f}\}$$

Nun lässt sich sehr leicht vorstellen, dass jede der drei gedachten Knoteninstanzen eine Teilmenge D_a bis D_c beziehungsweise D_d bis D_f als Eingabemenge erhält, um sie zu verarbeiten. Im Bezug auf die originale Multimenge D kann so eine Lastverteilung und Replikation modelliert werden.

Nun ergibt sich die Frage, *wie* die Knoteninstanzen diese Teilmengen erhalten. Dafür können die Netzwerkfunktionalitäten der erweiterten Modelle für das umgebende Betriebssystem und des

Datenbanksystems genutzt werden, auf denen die Knoteninstanzen basieren. Allerdings muss hier bedacht werden, dass gemäß der Modellierung eines Computernetzwerkes in Textabschnitt 5.5 ab Seite 149 die Nutzung der Netzwerkfunktionalitäten, dass heisst der Austausch von Marken von und zu Knoteninstanzen, nur in Form von Netzwerkmarken geschehen darf. Das bedeutet für obiges Beispiel, dass die Datenmarken der Teilmengen zunächst in Netzwerkmarken transformiert werden, die die Datenmarken als Submarken enthalten. Das QPN-Modell des Computernetzwerks "transportiert" dann die Netzwerkmarken zur jeweiligen Knoteninstanz. Dort können die Netzwerkmarken wieder in Datenmarken zurücktransformiert werden.

Allerdings muss noch ein Aspekt der Netzwerkmodelle bedacht werden. Nach obiger Vorstellung ist es möglich, verteilte Datenbanksysteme zu modellieren. Die Knoteninstanzen bilden dabei jeweils einen Clusterknoten ab, auf dem die Instanz eines verteiltes Datenbanksystems arbeitet. Allerdings muss angemerkt werden, dass bei der Modellierung eines verteilten Datenbanksystems zwischen quasi-verteilten und verteilten Datenbanksystemen unterschieden werden muss. Wie in Abschnitt 2.4 ab Seite 29 beschrieben, sind quasi-verteilte Datenbanksysteme diejenigen, die zum Zwecke der Lastverteilung und Replikation eine externe Komponente brauchen, um Datenbankoperationen auf die Instanzen des Datenbanksystems zu verteilen, die an sich unabhängig und als zentrales Datenbanksystem konzipiert sind. "Echte" verteilte Datenbanksysteme benötigen diese externe Komponente nicht, da die Lastverteilung und Replikation bereits fest implementiert ist[12].

Wie bereits erwähnt, weist die generelle Arbeitsweise der Netzwerkmodelle große Parallelen beziehungsweise Überschneidungen mit der Arbeitsweise des Modells für das Datenbanksystems auf, dass in den Textabschnitten 5.4.1 bis 5.4.3 ab Seite 144 beschrieben ist. Um redundante Textpassagen zu vermeiden, werden in den beiden folgenden Textabschnitten nur die Unterschiede erläutert.

5.7.2 Generelle Arbeitsweise für ein verteiltes Datenbanksystem mit zentraler Komponente

Zur Erläuterung und Illustration der generellen Arbeitsweise für ein verteiltes Datenbanksystem mit zentraler Komponente sei ein Servercluster gegeben, der aus vier Clusterknoten besteht. Diese Clusterknoten werden über einen Switch miteinander verbunden, womit ein Datenaustausch ermöglicht wird. Dieser Servercluster kann als QPN-Modell konzipiert werden, wie es in Abbildung 5.9 auf Seite 152 dargestellt ist. In dieser Abbildung repräsentiert die Stelle s_1 die zentrale Komponente, die Stellen s_2 bis s_4 das quasi-verteilte Datenbanksystem und die Stelle *Sw* den Switch. In den inneren Modellen dieser Stellen wird jeweils eine Instanz des erweiterten Modelles für das Betriebssystem ausgeführt, wie es in Abbildung 5.11 auf Seite 154 dargestellt ist. Jede Instanz führt als inneres Modell für Prozessstelle p_3 eine Instanz des erweiterten Modelles für das Datenbanksystem aus. Insgesamt entsteht dadurch ein hierarchisches QPN-Modell, wie es in Abbildung 5.10 auf Seite 153 dargestellt ist.

Seien zur besseren Abgrenzung

- die Instanzen des erweiterten Modelles, die als inneres Modell in den Stellen s_1 bis s_4 ausgeführt werden, mit ${}_{[1]}OS$ bis ${}_{[4]}OS$ bezeichnet. Zusätzlich seien die Stellen und Transitionen der Instanzen ${}_{[1]}OS$ bis ${}_{[4]}OS$ mit voran- und nachgestellten Index bezeichnet, um sie klar voneinander abzugrenzen. So bezeichnet ${}_{[1]}p_1$ die Stelle p_1 in Instanz ${}_{[1]}OS$ und ${}_{[1]}t_1$ die Transition t_1 in Instanz ${}_{[1]}OS$.

[12] Vgl. dazu auch Abbildung 2.8 auf Seite 36.

- die Instanzen des erweiterten Modelles für das Datenbanksystem, die als inneres Modell in den Stellen ${}_{[1]}p_3$ bis ${}_{[4]}p_3$ ausgeführt werden, mit ${}_{(1)}$*DBS* bis ${}_{(4)}$*DBS* bezeichnet. Analog seien die dort genutzten Stellen und Transitionen mit voran- und nachgestellten Index bezeichnet. So bezeichnet ${}_{(1)}p_1$ die Stelle p_1 in Instanz ${}_{(1)}$*DBS* und ${}_{(1)}t_1$ die Transition t_1 in Instanz ${}_{(1)}$*DBS* (man beachte die runden Klammern).

Sei nun angenommen, dass ein Benchmark existiert, der für das modellierte quasi-verteilte Datenbanksystem geeignet ist. Es können dann die ersten drei Schritte der generellen Arbeitsweise eines Datenbanksystem von Textabschnitt 5.4.1 vollzogen werden. Es ergeben sich aber folgende Unterschiede:

- Die beschriebenden Anpassungen der Instanzen des erweiterten Modelles für das umgebende Betriebssystem ${}_{[1]}$*OS* bis ${}_{[4]}$*OS* müssen jeweils auf den modellierten Clusterknoten angewendet werden. Das bedeutet im besonderen, dass die inneren Modelle für die Ressourcenstellen (Haupt- und Auslagerungsspeicher, Massenspeicher und Prozessoren) auf die technischen Charakteristiken des jeweiligen Clusterknotens angepasst werden müssen. Die beschriebene Vorgehensweise bleibt dabei dieselbe, muss aber für obiges Beispiel viermal vollzogen werden (Anpassung der inneren Modelle der Instanzen ${}_{[1]}$*OS* bis ${}_{[4]}$*OS*).
- Die beschriebenen Anpassungen des erweiterten Modelles für das Datenbanksystem werden nur für die Instanzen ${}_{(2)}$*DBS* bis ${}_{(4)}$*DBS* durchgeführt. Das bedeutet, dass die inneren Modelle für die Stellen ${}_{(2)}p_{11}$ bis ${}_{(4)}p_{11}$ und ${}_{(2)}p_{12}$ bis ${}_{(3)}p_{12}$ (Repräsentation der Datenbankdateien beziehungsweise Datenbankindizes) konzipiert und festgelegt werden. Da davon ausgegangen wird, dass sich die Instanzen ${}_{(2)}$*DBS* bis ${}_{(4)}$*DBS* gleich verhalten sollen, können die dieselben Transitionsregelsätze für die Transition t_4 bis t_8 in den genannten Instanzen verwendet werden. Für die Instanz ${}_{(1)}$*DBS* ist das alles nicht nötig, das sie die zentrale Komponente und keine Instanz des quasi-verteiltes Datenbanksystem repräsentiert.

Allerdings kommen den Transitionen und Stellen in Instanz ${}_{(1)}$*DBS* besondere Bedeutung zu. Durch sie werden der Partitionierungs- und der Replikationsalgorithmus modelliert, die bei der zentralen Komponente zum Einsatz kommen. Dazu kann man sich vorstellen, dass nach Durchführung der ersten drei Schritte der generellen Arbeitsweise eines Datenbanksystem von Textabschnitt 5.4.1 die Multimenge D erzeugt wurde. Sie enthält Daten- und Submarken als Repräsentation der generierten Datensätze, die aus der Analyse des Datengenerators des Benchmarks entstanden sind. Diese Multimenge kann als Eingabemenge für das gesamte hierarchische QPN-Modell angesehen werden, sofern eine Simulation angestrebt wird.

Der Transition t_7 in Instanz ${}_{(1)}$*DBS* kommt nun die Aufgabe zu, durch geeignete Transitionsregeln die Multimenge D in der Weise zu transformieren, dass dadurch die angestrebte Lastverteilung beziehungsweise Datenreplikation modelliert wird. Auf Basis der Multimenge D sowie den oben beschriebenen Teilmengen D_a bis D_f kann die Lastverteilung und Replikation realisiert werden:

1. Die Prozessstelle ${}_{[1]}p_1$ (Instanz des erweiterten Modelles für das umgebende Betriebssystem, siehe Abbildung 5.11 auf Seite 154) enthält initial eine Prozessmarke $g \in G$ (siehe Tabelle 5.1 auf Seite 133). Diese Prozessmarke enthält nur die Multimenge D, alle anderen Komponenten werden mit der leeren Menge belegt.
2. Die Prozessmarke g wird von der Transition ${}_{[1]}t_1$ konsumiert. Es werden Ressourcenmarken aus den Ressourcenstellen ${}_{[1]}p_4$, ${}_{[1]}p_5$ und ${}_{[1]}p_7$ allokiert und eine neue Prozessmarke $g' \in G$ in Prozessstelle ${}_{[1]}p_2$ ablegt. Die Prozessmarke g' enthält neben den allokierten Ressourcenmarken auch die Multimenge D.

3. Die Prozessmarke g' wird durch die Prozessstelle ${}_{[1]}p_2$ verarbeitet, wobei ein Prozess-Scheduler für den Fall eingesetzt wird, dass mehrere Prozessmarken in ${}_{[1]}p_2$ abgelegt wurden. Transition ${}_{[1]}t_2$ konsumiert die Prozessmarke g', erzeugt sie neu und legt sie in Prozessstelle ${}_{[1]}p_3$ ab. Eine Kopie erreicht das innere Modell von Prozessstelle ${}_{[1]}p_3$, also die Instanz ${}_{(1)}$*DBS* des erweiterten Modelles für das Datenbanksystem, wie es in Abbildung 5.12 auf Seite 156 dargestellt ist.
4. In der Instanz ${}_{(1)}$*DBS* wird die Prozessmarke g' in der Stelle *In* abgelegt. Transition ${}_{(1)}t_4$ konsumiert sie und die enthaltenen Datenmarken, also alle $d \in D$, werden extrahiert. Diese Datenmarken werden in Stelle ${}_{(1)}p_8$ abgelegt.
5. Transition ${}_{(1)}t_5$ konsumiert die Datenmarken aus Stelle ${}_{(1)}p_8$, erzeugt sie neu und legt sie in Stelle ${}_{(1)}p_9$ ab. Selbiges gilt auch für Transition ${}_{(1)}t_6$, sodass alle Datenmarken in Stelle ${}_{(1)}p_{10}$ abgelegt werden.
6. Transition ${}_{(1)}t_7$ besitzt einen Transitionsregelsatz, wobei die enthaltenen Transitionsregeln zwei Ziele verfolgen:
   (a) Die Datenmarken, die für einen bestimmten "Empfänger" bestimmt sind (die Instanzen ${}_{[2]}$*OS* bis ${}_{[4]}$*OS*), werden konsumiert und als Submarken in neu erzeugte Netzwerkmarken verlagert. Die Netzwerkmarken werden mit einer Farbe gefärbt, die den "Empfänger" kennzeichnet. Das bedeutet zusammengefasst, dass die Marken der Mengen D_a bis D_f in Netzwerkmarken transformiert werden, wobei die Elemente dieser Menge Submarken entsprechen. Die erzeugten Netzwerkmarken werden in der Stelle ${}_{(1)}p_{13}$ abgelegt.
   (b) Gleichzeitig werden in Stelle ${}_{(1)}p_{11}$ Assoziationsmarken abgelegt. Sie assoziieren jede Datenmarke mit dem "Empfänger" der Netzwerkmarke, in die jeweilige Datenmarke als Submarke abgelegt wurde. Dadurch wird die Partitionierung der zentralen Komponente modelliert. Das bedeutet, es wird verzeichnet, welche Datenmarke welcher Instanz ${}_{[2]}$*OS* bis ${}_{[4]}$*OS* zugewiesen wurde. Ein solches Verzeichnis kann dann genutzt werden, um festzustellen, welche Datenmarke durch welche Instanz verarbeitet wurde.

Die Netzwerkmarken, die in Stelle ${}_{(1)}p_{13}$ abgelegt wurden, werden durch das übergeordnete QPN-Modell des Computernetzwerkes verarbeitet. Genauer gesagt, Transition t_b des QPN-Modell in Abbildung 5.9 auf Seite 152 konsumiert alle Netzwerkmarken aus Stelle ${}_{(1)}p_{13}$, erzeugt sie neu und legt sie in Stelle *Sw* ab. Die dortigen Transitionsregeln konsumieren die Netzwerkmarken und erzeugen sie neu. Beim Konsum wird die Farbcodierung der ursprünglichen Netzwerkmarken insoweit beachtet, dass die neu erzeugten in die zugewiesenen, korrekten Stellen s_2 bis s_4 abgelegt werden.

Dadurch, dass das QPN-Modell aus hierarchisch angeordneten Instanzen besteht, stehen die Netzwerkmarken in den Stellen s_2 bis s_4 gleichzeitig in der Stelle p_{13} der Instanzen des erweiterten Modelles für das umgebende Betriebssystem zur Verfügung. Diese Stelle ist in Abbildung 5.11 auf Seite 154 zu erkennen. Genauer gesagt, die Netzwerkmarken sind in den Stellen ${}_{[2]}p_{13}$ bis ${}_{[4]}p_{13}$ verfügbar. Die Transitionen ${}_{[2]}t_{10}$ bis ${}_{[4]}t_{10}$ besitzen Transitionsregeln, die nur diejenigen Netzwerkmarken konsumieren, deren Farbcodierung aussagt, das sie für die jeweilige Instanz des erweiterten Modells für das umgebende Betriebssystem bestimmt sind. Das bedeutet zum Beispiel, dass die Netzwerkmarken von der Transition ${}_{[2]}t_{10}$ konsumiert werden, die per Farbcodierung für die Instanz ${}_{[2]}$*OS* bestimmt sind. Beim Konsum der Netzwerkmarken werden Datenmarken, die als Submarken enthalten sind, extrahiert und eine neue Prozessmarke wird erzeugt. In dieser Prozessmarke sind die extrahierten Datenmarken enthalten. Dieser Vorgang

wird in allen Instanzen $_{[2]}$*OS* bis $_{[4]}$*OS* gleichzeitig ausgeführt. Fasst man die Instanzen $_{[2]}$*OS* bis $_{[4]}$*OS* als zentral betriebene Datenbanksysteme auf, so kann die Verarbeitung der Prozessmarken dergestalt erfolgen, wie es in Textabschnitt 5.4.1 beschrieben wird.

Das Vorgehen, das bis zu diesem Punkt beschrieben ist, dient nur dazu, die Marken der Multimenge D auf die Instanzen $_{(2)}$*DBS* bis $_{(4)}$*DBS* zu verteilen und verarbeiten zu lassen. Diese Multimenge repräsentiert die Daten, die der Datengenerator des Benchmarks erzeugt hat. Wie bereits erwähnt, entspricht das beschriebene Vorgehen den ersten drei Schritten der generellen Arbeitsweise eines Datenbanksystem, wie es in Textabschnitt 5.4.1 erläutert ist. Es können dann die letzten beiden Schritte dieser Arbeitsweise durchgeführt werden. Das bedeutet, dass eine Simulation durchgeführt und der Zustand des hierarchischen Modells gesichert wird. Das betrifft insbesondere die Stelle $_{(1)}p_{11}$ sowie alle Ressourcenstellen der Instanzen $_{[2]}$*OS* bis $_{[4]}$*OS*.

Es verbleibt noch die Beschreibung der Arbeitsweise, wenn Datenbankinhalte eines verteilten Datenbanksystems mit zentraler Komponente modifiziert oder gelesen werden sollen. Für ein zentral betriebenes Datenbanksystem sind die Arbeitsweisen bereits in den Textabschnitten 5.4.2 und 5.4.3 erläutert worden. Um auch hier textuelle Redundanzen zu vermeiden, werden im folgenden nur die Unterschiede zu den beiden genannten Textabschnitten beschrieben. Sei ferner angenommen, dass der Anfragegenerator des Datenbanksystem-Benchmarks bereits analysiert wurde und damit die Multimengen D_1 und D_2 bereits existieren. Hierbei sei bemerkt, dass erstgenannte Multimenge Datenmarken enthält, die unterschiedlich gefärbt sein können, um unterschiedliche modifizierende Operationen zu unterscheiden (zum Beispiel Einfüge-, Änderungs- und Löschoperationen). Zweitgenannte Multimenge enthält Datenmarken, die repräsentieren, welche Daten vom verteilten Datenbanksystem zum Beispiel durch Datenbankabfragen gelesen werden sollen. Für beide Multimengen gilt, dass sie echte Teilmengen der Multimenge D sind.

Die Arbeitsweise zur Verarbeitung der Multimengen D_1 und D_2 verläuft im Prinzip genauso, wie es bisher beschrieben ist. Das bedeutet, dass sie als Eingabemenge für das hierarchische QPN-Modell genutzt werden. Genauer gesagt, in Prozessstelle $_{[1]}p_1$ wird eine Prozessmarke $g \in G$ initial abgelegt, wobei diese entweder D_1 oder D_2 als Komponente enthält. Danach erfolgt die Verarbeitung dieser Prozessmarke, wie es weiter oben beschrieben ist. Das heisst, die Datenmarken werden in Netzwerkmarken transformiert, transportiert, zurücktransformiert und von den einzelnen Instanzen $_{(2)}$*DBS* bis $_{(4)}$*DBS* verarbeitet. Hier ergibt sich der einzige Unterschied, dass bei der Erzeugung und Färbung der Netzwerkmarken durch die Transitionsregeln in Transition $_{(1)}t_6$ die Assoziationsmarken aus Stelle $_{(1)}p_{11}$ genutzt werden, um den "Empfänger" der Netzwerkmarken zu bestimmen. Diese Transitionsregeln ersetzen somit die bisherigen auf die Weise, dass das bisher modellierte Lastverteilungsverfahren nicht mehr aktiv ist. Stattdessen werden die Partitionierungsinformationen in Form der Assoziativmarken genutzt.

Bei den zuletzt genannten Instanzen $_{(2)}$*DBS* bis $_{(4)}$*DBS* ergibt sich aber ein wesentlicher Unterschied gegenüber der Beschreibung der Arbeitsweise in Textabschnitt 5.4.1: die konsumierten Datenmarken aus dem zweiten Schritt werden zusätzlich neu erzeugt und in Stelle p_{10} abgelegt. Transition t_7 konsumiert sie und transformiert sie in Netzwerkmarken, wobei die Datenmarken als Submarken enthalten sind. Diese Netzwerkmarken werden so gefärbt, dass damit als "Empfänger" die Instanz $_{[1]}$*OS* (also die modellierte zentrale Komponente des quasi-verteilten Datenbanksystems) codiert wird. Die erzeugten Netzwerkmarken werden dann in die Stelle p_{13} abgelegt. Betrachtet man das übergeordnete QPN-Modell für das Computernetzwerk in Abbildung 5.9 auf Seite 152, so konsumieren die Transitionen t_c, t_e und t_g die Netzwerkmarken, erzeugen sie neu und legen sie in Stelle *Sw* ab. Transition t_a konsumiert sie, erzeugt sie erneut und legt sie in Stelle s_1 ab. Dadurch gelangen sie in Stelle $_{[1]}p_{13}$ in der Instanz des erweiterten Modell für das umgebende Betriebssystem, dargestellt in Abbildung 5.11 auf Seite 154. Transition

$_{[1]}t_1$ konsumiert sie, aber erzeugt sie nicht neu, womit sie verworfen werden. Zusammengefasst wird dadurch das Lesen von Datenbankinhalten in einem quasi-verteilten Datenbanksystem modelliert. Das bedeutet, dass eine Datenbankabfrage an die zentrale Komponente gestellt wird. Sie versucht, anhand der Partitionierungsinformationen diese Datenbankabfrage in mehrere aufzuteilen, die dann parallel an die einzelnen Instanzen des quasi-verteilten Datenbanksystems verteilt werden. Sie verarbeiten die aufgeteilten Abfragen parallel und senden das Abfrageergebnis zurück an die zentrale Komponente. Die Ergebnisse werden dann wieder zusammengefasst und ergeben so das eigentiche Ergebnis der ursprünglichen Datenbankabfrage. Dieses Verhalten ist in Abbildung 2.8(a) auf Seite 36 dargestellt.

Bei modifizierenden Datenbankabfragen ist das nur teilweise notwendig. Hier sind zwei Unterschiede erkennbar. Der erste Unterschied ist, dass eine modifizierende Abfrage nur dann aufgeteilt werden kann, wenn anhand der Partitionierungsinformationen sichergestellt werden kann, dass die zu modifizierenden Datenbankinhalte bereits existieren. Das gilt also für Datenbankabfragen, die Datenbankinhalte löschen oder ändern. Bei Datenbankabfragen, die neue Datenbankinhalte einfügen, wird das eingestellte Lastverteilungsverfahren verwendet und die Partitionierungsinformationen entsprechend aktualisiert. Der zweite Unterschied ist, dass bei solchen modifizierenden Datenbankabfragen keinerlei Daten im Sinne der oben genannten Abfrageergebnisse an die zentrale Komponente zurücksenden.

5.7.3 Generelle Arbeitsweise für ein verteiltes Datenbanksystem ohne zentrale Komponente

Die generelle Arbeitsweise für ein verteiltes Datenbanksystem ohne zentrale Komponente unterscheidet sich nur wenig von der, die im vorhergehenden Textabschnitt beschrieben wurde. Die zwei wesentlichen Unterschiede sind dennoch:

1. Betrachtet man die oberste Ebene des hierarchischen QPN-Modells, also das QPN-Modell für das Computernetzwerk in Abbildung 5.9 auf Seite 152, so ergibt sich eine Neuinterpretation der Stelle mit inneren Modell s_1. Vormals repräsentierte diese Stelle die zentrale Komponente. Da diese nur bei einem quasi-verteilten Datenbanksystem gültig ist, entspricht sie nun einer Instanz des verteilten Datenbanksystems wie die übrigen Stelle s_2 bis s_4. Dadurch ist es notwendig, dass natürlich die Instanzen der erweiterten Modelle für das umgebende Betriebssystem und des Datenbanksystem, die in s_1 ausgeführt werden, genauso konfiguriert werden müssen wie diejenigen in s_2 bis s_4. Dabei bleiben alle beschriebenen Arbeitsweisen zur Lastverteilung davon unberührt. Das betrifft insbesondere die Transitionsregeln in Transition $_{(1)}t_7$.
2. Die Datenmarken der Multimengen D, D_1 und D_2 werden weiterhin in Netzwerkmarken als Submarken transformiert und über das QPN-Modell für das Computernetzwerk transportiert. Der Unterschied besteht aber darin, dass die Netzwerkmarken nicht wieder in Datenmarken zurücktransformiert und in neu erstellte Prozessmarken abgelegt werden. Das betrifft die Instanzen $_{[2]}OS$ bis $_{[4]}OS$. Stattdessen werden die Netzwerkmarken direkt in den Stellen $_{(2)}p_{13}$ bis $_{(4)}p_{13}$ genutzt. Mit anderen Worten ausgedrückt: die Instanzen des erweiterten Modelles für das umgebende Betriebssystem werden komplett ignoriert. Die Datenmarken werden in Form von Netzwerkmarken direkt zwischen den Instanzen des erweiterten Modelles für das Datenbanksystem ausgetauscht, wobei wie erwähnt das QPN-Modell für das Computernetzwerkwerk für den Transport der Netzwerkmarken zuständig ist.

Insgesamt wird dadurch das Verhalten eines verteilten Datenbanksystems bei der Verarbeitung von Datenbankabfragen modelliert, wie es in Textabschnitt 2.4 und in Abbildung 2.8(b) auf Seite 36 dargestellt ist. Das bedeutet, dass die einzelnen Instanzen des verteilten Datenbanksystems untereinander zum Zwecke der Lastverteiltung Daten austauschen. Dafür ist keine externe Komponente notwendig.

5.8 Kapitelzusammenfassung

In Kapitel 5 wurden die Faktoren, die in den vorherigenn Kapiteln 3 und 4 herausgestellt und einen bemessbaren Einfluss auf die Performance, den Energieverbrauch und die Energieeffizienz eines Datenbanksystems haben können, zu einem formalen Optimierungsproblem $\mathcal{P}_{EE}$ zusammengefasst.

Für dieses Optimierungsproblem wurde im Rahmen einer Literaturrecherche evaluiert, welche Lösungsmöglichkeiten existieren. Hierbei hat sich herausgestellt, dass es prinzipiell nur zwei Lösungswege gibt. Bei der analytischen Lösung werden mathematische Verfahren eingesetzt, um die Einflussfaktoren auf die oben genannten Metriken in einem mathematischen Modell zu erfassen und berechenbar zu gestalten. Der Vorteil bei analytischen Lösungen besteht darin, dass die Berechnungen sehr effizient durch spezielle Anwendungsprogramme realisiert werden können. Der Nachteil ist allerdings, dass das mathematische Modell durchaus sehr komplex werden kann, um alle Einflussfaktoren möglichst realitätsgetreu abbilden zu können.

Die zweite Lösungsmöglichkeit besteht darin, die Einflussfaktoren experimentell und explorativ zu bestimmen. Von dieser Lösungsmöglichkeit wurde im Rahmen von Kapitel 4 bereits Gebrauch gemacht und die experimentellen Ergebnisse erläutert. Der Vorteil bei dieser Lösungsmöglichkeit besteht darin, dass die experimentellen Ergebnisse direkt dafür genutzt, verschiedene Datenbanksysteme zu vergleichen, sofern anerkannte Vergleichsmetriken genutzt wurden sowie Handlungsempfehlungen zu geben, die praxisrelevant sind. Wie in Textabschnitt 4.3 erwähnt, besteht der Nachteil dieser Lösungsmöglichkeit darin, dass für die Experimente ein gewisser Aufwand an technischer Ausstattung und besonders Zeit aufgewendet werden muss.

Die Gemeinsamkeit beider Lösungen ist allerdings, dass ein grundlegendes Modell erarbeitet wird, das die Basis für mathematische oder experimentelle Lösungen darstellt. Um bei der Wahl der Lösungsmöglichkeit nicht eingeschränkt zu sein, wurden im Hinblick auf das Modell drei Anforderungen aufgestellt, um die Ziele dieser Disseration zu erreichen, die in Textabschnitt 1.1 beschrieben wurden (Modellierung der allgemeinen Charakteristiken von Datenbanksystemen und der technischen Plattform sowie die Unterstützung von generischen Datenbankanwendungen in Form von Benchmarks).

Nach einer Recherche und Analyse, auf welche Weise diese drei Anforderungen am geeignetesten erfüllt werden können, erscheint die Systemanalyse neben der Systemsimulation als aussichtsreichster Kandidat. Der Vorteil der Systemanalyse von Datenbanksystemen ist, dass sie durch bewährte formale und grafische Methoden in verschiedenen Aspekten beschrieben werden können. Darunter fällt auch die Analyse des Datenflusses als fundamentaler Bestandteil. Die Modellierung des Datenflusses innerhalb eines Datenbanksystems birgt großes Potential, die drei genannten Anforderungen als Ganzes zu erfüllen. Die zentrale Aussage im Bezug auf die Lösung des Optimierungsproblems ist:

> *Ein Datenflussmodell, dessen Komponenten sowohl die Architektur eines Datenbanksystems als auch alle notwendigen technischen Voraussetzungen abstrahieren kann, ist*

in der Lage, den Datenfluss zu simulieren, der durch einen Benchmark für ein Datenbanksystems definiert wird. Mit einer zusätzlichen Metrik wie dem simulierten mittleren Energieverbrauch pro Modellkomponente ist es möglich, den Gesamtenergieverbrauch zu simulieren. Zusammen mit den Simulationsergebnissen hinsichtlich der Performancemetriken wie Antwortzeit oder Durchsatz ist damit die Simulation der Energieeffizienz laut Gleichung 3.3 auf Seite 64 realistisch.

Die Aussage gilt es zu belegen. Dazu wurden verschiedene Möglichkeiten evaluiert, wie eine Modellierung des Datenflusses realisiert werden kann. Eine bewährte und anerkannte Methode ist der Einsatz von Petri-Netzen. Dessen am weitesten fortgeschrittenes Derrivat stellen *Queued Petri Nets* (QPNs) dar, dessen formale Grundlagen in Textabschnitt 2.5 beschrieben wurden. Sie wurden genutzt, um verschiedene grundlegende Modelle einzuführen und zu erläutern.

Darunter fällt zum Beispiel das QPN-Modell für das umgebende Betriebssystem, das in Textabschnitt 5.3 erläutert wurde und das abstrakt die Verwaltung und den Einsatz der Betriebsmittel einer technischen Plattform repräsentiert. Das zweite wichtige QPN-Modell ist das Modell für das Datenbanksystem, das in Textabschnitt 5.4 eingeführt und beschrieben wurde. Es ist deutlich komplexer als das vorher genannte QPN-Modell für das umgebende Betriebssystem. Der Grund dafür ist, dass durch dieses Modell die generelle Architektur von Datenbanksystemen abgebildet wird, wie sie in den Textabschnitten 2.1 bis 2.4 detailliert beschrieben wurde.

Es hat sich gezeigt, dass diese beiden QPN-Modelle nicht vollständig ausreichen, um auch verteilte Datenbanksysteme zu modellieren. So wurde in Textabschnitt 5.5 evaluiert, wie der zugrunde liegende Servercluster eines verteilten Datenbanksystems nachgestellt werden kann. In diesem Zusammenhang war es auch notwendig, die Modelle für das umgebende Betriebssystem und des Datenbanksystems zu erweitern. So entstanden zwei zusätzliche QPN-Modelle, die in den Textabschnitten 5.6 und 5.7 beschrieben wurden.

Insgesamt entstand damit ein Grundstock an QPN-Modellen, die es ermöglichen sollen, Simulationen auszuführen. Deren Ziel ist es, eine Datenbankanwendung zu simulieren, die in Verbindung mit einem Datenbanksystem und einer technischen Plattform ausgeführt wird. Das kann unter anderem auch ein Benchmark sein. Genauer gesagt, es soll der Datenfluss simuliert werden, der entsteht, wenn besagte Datenbankanwendung ausgeführt wird. Anhand des Datenflusses können dann Vergleichsmetriken wie die Performance, der Energieverbrauch und die Energieeffizienz abgeleitet werden.

6 Evaluation der Modelle durch Simulation

Die QPN-Modelle, die im Rahmen des vorherigen Kapitels 5 eingeführt und beschrieben wurden, sollen es ermöglichen, Datenbanksysteme hinsichtlich der Performance und des Energieverbrauches der modellierten technischen Plattform zu simulieren. Insgesamt kann dadurch die Energieeffizienz gemäß Gleichung 3.3 auf Seite 64 errechnet werden.

Es muss validiert werden, welche Qualität beziehungsweise welche Güte die besagten QPN-Modelle besitzen, um die oben genannten Metriken in Rahmen einer Simulation für ein gegebenes Datenbanksystem, der genutzten technischen Plattform und eines Anwendungsfalles, etwa einem Benchmark, zu erhalten. Das bedeutet zweierlei: zunächst muss die Frage geklärt werden, ob überhaupt die QPN-Modelle für eine Simulation geeignet sind. Wenn ja, muss ermittelt, mit welcher Qualität das geschieht.

Zur Betrachtung des zweiten Aspektes wird zuerst im nachfolgenden Textabschnitt 6.1 der Begriff der *Genauigkeit* definiert. Die Genauigkeit wird als Vergleichmetrik genutzt, um die Qualität von Simulationen anhand der genannten QPN-Modelle numerisch zu evaluieren. Im Bezug auf den ersten Aspekt kann der Umstand genutzt werden, dass bereits im Rahmen dieser Dissertation experimentelle Ergebnisse in Kapitel 4 vorliegen. Sie hatten das Ziel, die Performance, den Energieverbrauch und die Energieeffizienz von Datenbanksystemen zu untersuchen. Als logische Konsequenz können also die Experimente, die in Kapitel 4 beschrieben wurden, mit Hilfe der QPN-Modelle simuliert werden, um die Genauigkeit zu errechnen.

Dementsprechend wird in den Textabschnitten 6.2 und 6.3 erläutert,

- wie die QPN-Modelle aus Kapitel 5 angepasst wurden, um ein Experiment aus Kapitel 4 nachzustellen
- wie die so angepassten QPN-Modelle konfiguriert wurden
- in welcher Form die Simulationen durchgeführt wurden und
- welche Ergebnisse die Simulationen im Vergleich zu den Experimenten hatten.

6.1 Der Genauigkeitsbegriff

Um die Qualitätsfaktoren im Umgang mit den QPN-Modellen zu erkennen, zu bemessen und sie sicherzustellen, wurde eine Literaturrecherche durchgeführt. Neben der Publikation von *Ören* in [Öre84] ist besonders die Studie von *Robinson* maßgeblich [Rob02]. In seiner Studie hat er viele Simulationsprojekte untersucht und dabei zentrale Qualitätsfaktoren ermittelt. Diese Faktoren sind in Tabelle 6.1 auf Seite 166 samt einer kurzen Beschreibung aufgelistet. Die rechte Spalte dieser Tabelle gibt an, wie relevant die Qualitätsfaktoren hinsichtlich der Simulation der QPN-Modelle sind, die in Kapitel 5 eingeführt und beschrieben wurden.

In seiner Studie beschreibt *Robinson*, dass die Bemessung der Qualität einer Simulation und der zugrunde liegende Modelle insgesamt von drei Aspekten abhängt: der Plausibilität[1], Glaubwürdigkeit und Akzeptanz. Eine genaue Abgrenzung von Plausibilität und Glaubwürdigkeit ist nach *Robinson* schwierig und hängt davon ab, in welchem Kontext eine Simulation angesiedelt ist. Eine Simulation der oben genannten QPN-Modelle fällt aber eindeutig in den Kontext von zeitdiskreten Simulationen, wobei konstante Zeitintervalle die Simulationsgrundlage bilden. Nach *Robinson* können die Qualitätsaspekte Plausibilität und Glaubwürdigkeit danach bemessen werden, wie realistisch die Simulationsergebnisse im Vergleich zum realen System sind. Mit anderen Worten bedeutet es, dass die Differenz der Ergebnisse eines realen Systems möglichst klein gegenüber denjenigen sein sollen, die durch die Simulation des Systems gewonnen wurden. Im optimalen Fall ist kein Unterschied erkennbar, womit die Simulation das reale System wirklichkeitsgetreu widergibt. Nach *Robinson* beeinflusst natürlich eine steigende Plausibilität und Glaubhaftigkeit nachhaltig den letztgenannten Qualitätsaspekt Akzeptanz. Damit ist gemeint, dass eine Simulation sowie die zugrunde liegenden Modelle erst in dem Moment auf ein breites Interesse stoßen, wenn sie plausibel sind. Damit eröffnen sich weitere Möglichkeiten, zum Beispiel die Vorhersagen über das modellierte, reale System durch zusätzliche Simulationen mit geänderten Eingabeparametern.

Bezogen auf die QPN-Modelle, die in Kapitel 5 eingeführt und beschrieben wurden, bedeutet es die Definition der *Genauigkeit*. Die Genauigkeit beschreibt den prozentualen Unterschied von Simulationsergebnissen auf Basis der QPN-Modelle im Verhältnis zu realen Ergebnissen. Im Rahmen dieser Dissertation wird ein Genauigkeitsbegriff benutzt, wie er in Definition 13 beschrieben ist.

Definition 13: Genauigkeit
Die Genauigkeit gibt den Unterschied zwischen einem realen, experimentell gewonnenen Wert a_{exp} und einem simulierten Wert a_{sim} einer Metrik an. Diese Metrik kann die Performance, der Energieverbrauch oder die Energieeffizienz sein.

Die Genauigkeit wird als Prozentsatz angegeben. Eine Genauigkeit von 100 Prozent bedeutet, dass es keinen Unterschied zwischen dem realen und dem simulierten Wert der Metrik gibt. Eine Genauigkeit von 0 Prozent gibt an, dass der reale vom simulierte Wert der Metrik um mehr als das Doppelte abweicht:

$$accuracy(a_{sim}, a_{exp}) = \left[1 - \frac{|a_{sim} - a_{exp}|}{a_{sim}}\right] \cdot 100 \begin{cases} 100\% & wenn\ a_{sim} = a_{exp} \\ 0\% & wenn\ |a_{sim} - a_{exp}| \geq a_{sim} \end{cases}$$

[1] Im Original wird von *validity* gesprochen, was im Deutschen mit *Plausibilität*, *Aussagekraft* oder *Stichhaltigkeit* übersetzt werden kann.

Dabei muss beachtet werden, dass die Genauigkeit als Vergleichmetrik für die Qualität nur bedingt tauglich ist. Gesetzt den Fall, dass konkurrierende QPN-Modelle existieren, die denselben Zweck verfolgen wie die oben genannten, dann muss eine Evaluation dieser Modelle dieselbe Definition der Genauigkeit benutzen. Zudem muss die Evaluation der Genauigkeit exakt dieselbe sein, wie sie in diesem Textabschnitt beschrieben ist. Nur in diesen beiden Fällen ist ein direkter Vergleich der Genauigkeitswerte möglich.

Qualitätsfaktor	**Relevanz**
Modell: Geschwindigkeit, Ästhetik und Benutzbarkeit	sehr hoch
Vertrauen in das Modell: Vertrauenswürdigkeit und Glaubhaftigkeit des Modells und der Simulationsergebnisse	sehr hoch
Simulationsdaten: Verfügbarkeit und Genauigkeit	hoch
Simulationssoftware: Benutzbarkeit, Eignung, Flexibilität, Glaubhaftigkeit	hoch
Glaubhaftigkeit des Modellierenden: Glaubhaftigkeit und Ehrlichkeit des Modellierenden	mittel
Kompetenz des Modellierenden: Fähigkeiten und Kenntnisse des Modellierenden, um Simulation durchführen zu können	hoch
Professionalität des Modellierenden: Engagement des Modellierenden bezüglich der Simulation	nicht relevant
Zuverlässigkeit des Modellierenden	nicht relevant
Kommunikation und Interaktion: Häufigkeit, Inhalt und Angemessenheit der Kommunikation und Interaktion aller Beteiligten eines Simulationsprojektes	nicht relevant
Beteiligung (insbesondere des Kunden) in allen Phasen eines Simulationsprojektes	nicht relevant
Soziale Kompetenz zwischen Kunden und dem Modellierenden	nicht relevant
Weiterbildung des Kunden im Bezug auf das Simulationsmodell und die Simulation	nicht relevant
Anforderungsverständnis: die Bemühungen des Modellierenden, die Anforderungen und Wünsche des Kunden hinsichtlich des Modells und der Simulation zu verstehen	nicht relevant
Ansprechbarkeit des Modellierenden, sobald Kunde Anforderungen und Wünsche ändert	nicht relevant
Wiederherstellung eines Simulationsprojektes nach einem Fehlerfall	nicht relevant
Verfügbarkeit und Kontakt zum Modellierenden, Verfügbarkeit des Modells	nicht relevant
Kosten für das Modell und Simulation	niedrig
Bekenntnis des Kunden seines Unternehmens oder Organisation zum Simulationsprojekt	nicht relevant

Tabelle 6.1: Qualitätsfaktoren für eine Simulation nach [Rob02] sowie Relevanz für die Simulation der QPN-Modelle

6.2 Evaluation der simulierten vertikalen *scale-up*-Szenarien

Wie in Textabschnitt 5.4 beschrieben, kann das QPN-Modell für das umgebende Betriebsystem in Kombination mit dem QPN-Modell für das Datenbanksystem genutzt werden, um ein zentral betriebenes Datenbanksystem zu simulieren, wobei auch dessen technische Plattform und das Betriebssystem auf dieser Plattform einbezogen wird. Dabei wird der Datenfluss simuliert, der entsteht, wenn eine Datenbankanwendung ausgeführt wird, speziell ein Benchmark für die-

ses Datenbanksystem. Zusätzlich wird der Betriebsmittelkonsum bei der Ausführung des Benchmarks simuliert, womit Aussagen zum Energieverbrauch getroffen werden können.

Insgesamt sind Simulationen auf Basis der beiden eingangs genannten QPN-Modelle in der Lage, vertikale *scale-up*-Szenarien zu simulieren. Mit der Beschreibung der experimentell gewonnenen Ergebnisse aus Experiment B ab Seite 78 sowie Experiment C ab Seite 84 besteht die Möglichkeit, die Genauigkeit der beiden oben genannten QPN-Modelle zu evaluieren. Zur Erinnerung sei an dieser Stelle angemerkt, dass die Experimente B und C dieselbe technische Plattform und dasselbe zentrale Datenbanksystem *Postgresql* nutzen, um für verschiedene Datenbankgrößen (1, 5 und 10 GByte) die Performance in Form der Antwortzeit von *OLAP*-Datenbankabfragen und den Energieverbrauch der technischen Plattform zu ermitteln. Dabei wurde in Experiment B der *TPC-H*- und in Experiment C der *StarSchema*-Benchmark verwendet.

Der verbleibende Teil dieses Textabschnittes 6.2 beschreibt die Vorgehensweise, um die Experimente B und C anhand der oben genannten zwei QPN-Modelle zu simulieren. Dabei wurde eine Aufteilung in drei Teile vorgenommen: der erste Teil beschreibt die Anpassung des QPN-Modelles für das umgebende Betriebssystem. Da beide Experimente B und C dieselbe technische Plattform nutzten, gilt diese Beschreibung gleichermaßen für die beiden anderen Teile. Der zweite Teil beschreibt die Anpassung des Modelles für das Datenbanksystem, um eine Simulation von Experiment B durchzuführen. Die Simulationsergebnisse werden dann mit den experimentellen verglichen, um die Genauigkeit zu bestimmen. Analog werden im dritten Teil die Anpassungen für das Datenbanksystem erläutert, um eine Simulation von Experiment C vorzunehmen. Auch hier wird ein Vergleich zwischen den simulierten und experimentell gewonnenen Ergebnissen gezogen, um die Genauigkeit zu errechnen.

6.2.1 Anpassung des Modelles für das umgebende Betriebssystem

Da für Experiment B und C dieselbe technische Plattform genutzt wurde (siehe Tabelle 4.1 auf Seite 74), wird im folgenden die Anpassung des Modelles für das umgebende Betriebssystem beschrieben. Das Modell an sich ist in Abbildung 5.3 auf Seite 132 dargestellt.

Das Modell bietet vier Ressourcenstellen: p_4 für Massenspeicher-, p_5 für Arbeitsspeicher-, p_6 für Prozess-ID- und p_7 für Prozessor-Ressourcenmarken. Ressourcenstelle p_4 besitzt ein inneres Modell, wie es in Abbildung 5.5 auf Seite 137 dargestellt ist. Es repräsentiert die beiden Festplatten, die in der technischen Plattform nach Tabelle 4.1 auf Seite 74 verwendet wurden. Jede dieser Festplatten besitzt eine Speicherkapazität von 1 TByte, wobei die beiden Festplatten als RAID-0-Festplattenverbund genutzt wurden. Auf diesem Verbund wurde ein *ext4*-Dateisystem angelegt, wobei darauf geachtet wurde, dass die Datenblockgrößen identisch sind. Das ermöglichte eine sowohl technische als auch logische Datenblockgröße von 4 kByte. Aufgrund der Beschreibung des inneren Modells in Textabschnitt 5.3.2 wurden die beiden Stellen p_{4a} und p_{4b} mit jeweils $1\ \textit{TByte} \div 4\ \textit{kByte} = 137.438.953.472$ Marken initial belegt. Durch den RAID-0-Verbund stehen damit über Ressourcenstelle p_4 insgesamt $2 \times 137.438.953.472 = 274.877.906.944$ Massenspeicher-Ressourcenmarken zur Verfügung, wobei eine dieser Ressourcenmarke einem Datenblock mit 4 kByte Speicherkapazität entspricht.

Ressourcenstelle p_5 führt ein inneres Modell aus, wie in Abbildung 5.4 auf Seite 135 dargestellt und in Textabschnitt 5.3.1 beschrieben ist. Die Beschreibung der verwendeten technischen Plattform in Tabelle 4.1 auf Seite 74 sagt aus, dass insgesamt 6 GByte Hauptspeicher und 8 GByte Auslagerungsspeicher für die Experimente B und C verfügbar waren. Das bei diesen Experimenten genutzte *GNU/Linux*-Betriebssystem stellt beide Speicherarten als Arbeitsspeicher den Betriebssystemprozessen zur Verfügung. Die Prozesse können Arbeitsspeicher in

Form von Datenblöcken beziehungsweise Seiten anfordern, deren Größe 8 kByte beträgt. Dementsprechend wurde die Stelle p_{5a} initial mit 6 *GByte* $\div$ 8 *kByte* $= 805.306.368$ und Stelle p_{5b} mit 8 *GByte* $\div$ 8 *kByte* $= 1.073.741.824$ Marken belegt. Somit standen über Ressourcenstelle p_5 ingesamt $805.306.368 + 1.073.741.824 = 1.879.048.192$ Arbeitsspeicher-Ressourcenmarken zur Verfügung, wobei eine dieser Marken einen Datenblock beziehungsweise eine Seite mit einer Speicherkapazität von 8 kByte darstellt.

Ressourcenstelle p_6 wurde initial mit 65.495 Prozess-ID-Marken belegt. Diese Anzahl ergibt sich aus der fest eingestellten Anzahl an Prozessen, die unter dem verwendeten Betriebssystem *GNU/Linux* gestartet werden dürfen ($2^{16} = 65.536$), abzüglich der Anzahl der internen Prozesse für das Betriebssystem selber.

Ressourcenstelle p_7 besitzt ein inneres Modell, wie es in Abbildung 5.6 auf Seite 140 dargestellt ist. Die Arbeitsweise dieses inneren Modells wurde in Textabschnitt 5.3.3 erläutert. Damit repräsentiert dieses innere Modell die CPU, die in der technischen Plattform für die Ausführung von Experiment B und C verwendet wurde. Nach dessen Beschreibung in Tabelle 4.1 auf Seite 74 hat diese CPU vier Kerne mit einer nominalen Taktrate von 2.5 GHz pro Kern. Dementsprechend wurden die Stellen p_{7a} bis p_{7d} mit jeweils 2.500 Marken initial belegt, wobei jede Marke einer Taktrate von 1 MHz entspricht[2]. Für die Warteschlange in Stelle p_{7e} in Abbildung 5.6 wurde der *interleaving*-Schedulingalgorithmus festgelegt, wie es in Textabschnitt 5.3.3 beschrieben ist. Insgesamt standen über Ressourcenstelle p_7 genau $4 \times 2.500 = 10.000$ Prozessor-Ressourcenmarken zur Verfügung.

In Tabelle 6.2 ist zur Übersicht die Anzahl der initialen Ressourcenmarken beziehungsweise der Marken in den inneren Modellen aufgelistet.

Ressourcenstelle	Anzahl der Ressourcenmarken	Entsprechung
p_{4a}, p_{4b}	137.438.953.472	1 TByte Massenspeicher
p_4	274.877.906.944	2 TByte Massenspeicher ($p_{4a} + p_{4b}$)
p_{5a}	805.306.368	6 GByte Hauptspeicher
p_{5b}	1.073.741.824	8 GByte Auslagerungsspeicher
p_5	1.879.048.192	Arbeitsspeicher ($p_{5a} + p_{5b}$)
p_6	65.495	Prozess-IDs
p_{7a}, p_{7b}, p_{7c}, p_{7d}	2.500	Prozessor-Kern
p_7	10.000	Prozessor ($p_{7a} + p_{7b} + p_{7c} + p_{7d}$)

Tabelle 6.2: Initiale Markenbelegung in den Ressourcenstellen im Modell für das umgebende Betriebssystem zur Simulation von Experiment B

6.2.2 Simulation von Experiment B (*TPC-H*)

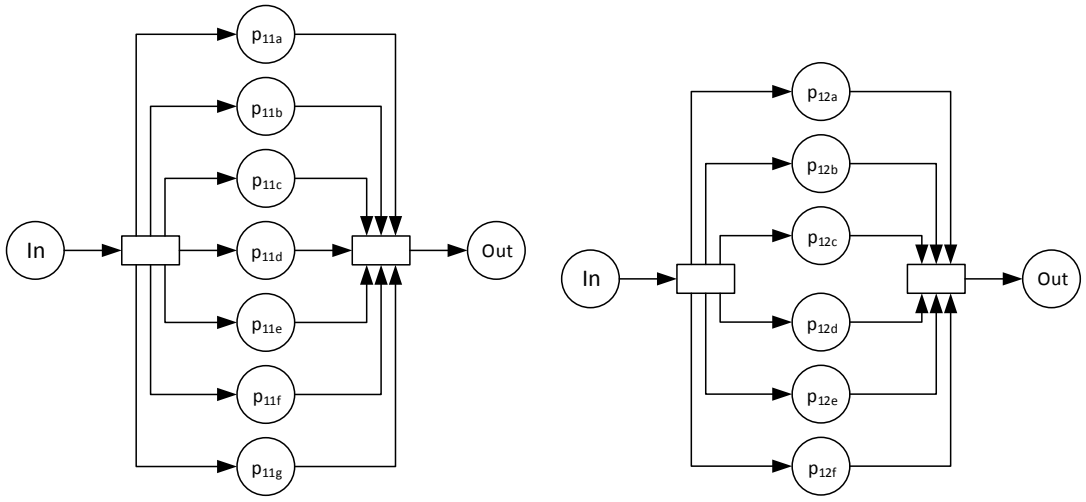
Zur Simulation von Experiment B müssen neben dem Modell für das umgebende Betriebssystem noch das Modell für das Datenbanksystem angepasst werden. Das Modell für das Datenbanksystem ist in Abbildung 5.7 auf Seite 143 dargestellt und in Textabschnitt 5.4 beschrieben. Zur

[2] Die Taktrate gibt lediglich an, wieviele Takte pro Sekunde eine CPU ausführt. Viele Prozessorinstruktionen können innerhalb von einem Takt ausgeführt werden, aber einige brauchen mehrere Taktzyklen für die Ausführung. Daher sagt die Taktrate nichts darüber aus, wieviele Prozessorinstruktionen tatsächlich in einer Sekunde durch eine CPU ausgeführt wurden. Im Rahmen dieser Dissertation wird aber der Einfachheit halber angenommen, dass jede Prozessorinstruktion genau einen Takt benötigt, um ausgeführt zu werden.

Anpassung dieses Modelles wurde die generelle Arbeitsweise des Modelles in Textabschnitt 5.4.1 genutzt, da sie detailliert vorschreibt, welche Schritte durchzuführen sind.

Modellanpassungen Experiment B sah die Verwendung des *TPC-H*-Benchmarks vor, daher wurde zuerst der Datengenerator dieses Benchmarks analysiert. Die Analyse ergab, dass durch den Datengenerator Datensätze nach dem relationalen Datenmodell für insgesamt sieben Tabellen generiert werden. In Textabschnitt 5.4 wurde erläutert, dass das innere Modell von Stelle p_{11} im Modell für das Datenbanksystem die Datenbankdateien repräsentiert. Im Falle des Datenbanksystem, dass in Experiment B verwendet wurde, kann das durch ein inneres Modell repräsentiert werden, wie es in Abbildung 6.1(a) auf Seite 169 dargestellt ist. Dabei repräsentieren die einzelnen Stellen p_{11a} bis p_{11g} jeweils eine Datenbankdatei, in der der Inhalt einer Tabelle gespeichert wird. Somit entstand ein inneres Modell, dass insgesamt sieben Stellen enthält.

Der Datengenerator des *TPC-H*-Benchmarks wurde auch dahingehend analysiert, welche Datenbankindizes für die sieben *TPC-H*-Tabellen erstellt werden müssen. Diese Datenbankindizes werden innerhalb des Modelles für das Datenbanksystem in Abbildung 5.7 auf Seite 143 durch das innere Modell von Stelle p_{12} repräsentiert. Die Analyse des Datengenerator ergab die Erstellung von sechs Datenbankindizes, wodurch ein inneres Modell erstellt wurde, dass sechs Stellen besitzt. Dieses innere Modell ist in Abbildung 6.1(b) dargestellt. Die Stellen p_{12a} bis p_{12f} repräsentieren dabei jeweils ein Tabellenindex.



(a) Stelle p_{11} (Datenbankdateien) (b) Stelle p_{12} (Indexdateien)

Abbildung 6.1: Innere Modelle der Stellen p_{11} und p_{12} im Modell für das Datenbanksystem zur Simulation von Experiment B

In Experiment B wurden insgesamt 18 Testserien durchgeführt, wobei immer eine Kombination aus einer Datenbankgröße (1, 5 und 10 GByte), einem Hauptspeicheranteil (1, 2.5 und 5 GByte) und einem Verbindungstyp (Einfach- und Mehrfachverbindung) verwendet wurde. Insbesondere der erste Testparameter, also die geteste Datenbankgröße, ist für die Anpassung des Modelles für das Datenbanksystem bezüglich der Simulation von Experiment B von Interesse.

Nachdem die notwendigen Modelle zur Simulation von Experiment B angepasst wurden, ist es als nächster Schritt notwendig, die generierte Datensätze des *TPC-H*-Datengenerators als Datenbasis einzufügen.

Das Einfügen dieser Datenbasis wird dadurch erreicht, dass drei Multimengen D_1, D_5 und D_{10} mit Datenmarken und Submarken auf Basis der Analyse des *TPC-H*-Datengenerators erstellt wurden. Diese Multimengen repräsentieren die generierten Datensätze für eine *TPC-H*-Datenbank mit 1, 5 und 10 GByte Daten ohne Datenbankindizes. Jede Datenmarke in den Multimengen repräsentiert einen generierten Datensatz. Die Datenmarken enthalten Submarken zur Repräsentation der Datenfelder. Da der Datengenerator die Erstellung von Datensätzen für insgesamt sieben Tabellen vorsieht, ergeben sich die Multimengen von Datenmarken als Vereinigung von sieben Teilmengen, wobei jede Teilmenge die Datenmarken einer Tabelle enthält. In Tabelle 6.3 sind die Quantitäten an Daten- und Submarken der einzelnen, oben genannten Multimengen aufgelistet. Die Struktur von Tabelle 6.3 kann am Beispiel von Multimenge D_1 erläutert werden. Die Anzahl an Datenmarken ist in der zweiten Spalte von links angegeben, wobei aufgeschlüsselt wird, wieviele Datenmarken für jede Tabelle (= Teilmenge) erzeugt wurden. Zusätzlich wird in der dritten Spalte von links angegeben, wieviele Submarken eine Datenmarke enthält. Die letzte Zeile von Tabelle 6.3 gibt die Summe der Datenmarken beziehungsweise die Summe aller Teilmarken an. Damit enthält Multimenge D_1 7.860.030 Datenmarken mit insgesamt 432.301.150 Submarken. Die Anzahl der Submarken ergibt sich aus dem Produkt der Datenmarken mit der Gesamtanzahl an Submarken.

Die Multimengen D_1, D_5 und D_10 wurden jeweils als Eingabemenge für eine Simulation des angepassten Modelles für das umgebende Betriebssystem in Kombination mit dem angepassten Modelles für das Datenbanksystem genutzt. Dadurch wurden die Stellen p_{11} (Datenbankdateien) und p_{12} (Datenbankindizes) befüllt. Die drei Simulationen wurden dabei gemäß der Beschreibung in Textabschnitt 5.4.1 durchgeführt. Nach jeder erfolgreichen Simulation wurde der Zustand, also die Markenbelegung der angepassten Modelle für das umgebende Betriebssystem und für das Datenbanksystem separat gesichert. Diese Sicherungen sind aus dem Grund durchgeführt worden, um einen definierten Grundzustand für die Simulation der Testreihen herzustellen.

	← *1 GByte Datenbank* →		← *5 GByte Datenbank* →		← *10 GByte Datenbank* →	
***TPC-H*-Tabelle**	**Datenmarken**	**Submarken**	**Datenmarken**	**Submarken**	**Datenmarken**	**Submarken**
`part`	200.000	8	1.000.000	8	2.000.000	8
`supplier`	10.000	7	50.000	7	100.000	7
`customer`	150.000	8	750.000	8	1.500.000	8
`orders`	1.500.000	9	7.500.000	9	15.000.000	9
`lineitem`	6.000.000	16	30.000.000	16	60.000.000	16
`nation`	25	4	25	4	25	4
`region`	5	3	5	3	5	3
$\sum$	7.860.030	432.301.650	39.300.150	2.161.508.250	78.600.300	4.323.016.500
	← *Multimenge* D_1 →		← *Multimenge* D_5 →		← *Multimenge* D_{10} →	

Tabelle 6.3: Anzahl der Datenmarken und Submarken in Multimenge D für verschiedene *TPC-H*-Datenbankgrößen

Die Anzahl der Marken in den inneren Modellen für die Stellen p_{11} und p_{12} des angepassten Modelles für das Datenbanksystem entsprechen der Anzahl der Datenmarken, wie sie in den Multimengen D_1, D_5 und D_{10} vorkommen. Der Grund dafür ist, dass es sich um Assoziativmarken im Sinne der Beschreibung in Textabschnitt 5.2.2 handelt.

Hierbei ist wichtig, wieviele zusätzliche Farben für die Relation zwischen Daten- und Massenspeichermarken notwendig wurden. Die Quantitäten sind in den beiden Tabellen 6.4 und 6.5 aufgelistet.

		← *Datenbankgröße* →		
Stelle	***TPC-H*-Tabelle**	**1 GByte**	**5 GByte**	**10 GByte**
p_{11a}	`part`	5.209	26.042	52.083
p_{11b}	`supplier`	314	1.571	3.142
p_{11c}	`customer`	5.347	26.733	53.467
p_{11d}	`orders`	31.738	158.691	317.383
p_{11e}	`lineitem`	122.070	610.352	1.220.703
p_{11f}	`nation`	1	1	1
p_{11g}	`region`	1	1	1

Tabelle 6.4: Anzahl der unterschiedlichen Farben der Assoziativmarken im inneren Modell von Stelle p_{11}

		← *Datenbankgröße* →		
Stelle	***TPC-H*-Tabelle**	**1 GByte**	**5 GByte**	**10 GByte**
p_{12a}	`part`	247	1229	2454
p_{12b}	`supplier`	13	63	613
p_{12c}	`customer`	185	921	1842
p_{12d}	`orders`	1842	9202	18402
p_{12f}	`nation`	1	1	1
p_{12g}	`region`	1	1	1

Tabelle 6.5: Anzahl der unterschiedlichen Farben der Assoziativmarken im inneren Modell von Stelle p_{12}

Simulationsdurchführung In einem nächsten Schritt wurde der Abfragegenerator des im Experiment B verwendeten *TPC-H*-Benchmarks analysiert. Genauer gesagt, es wurden die 22 einzelnen Datenbankabfragen daraufhin analysiert, welche Datensätze der Datenbasis durch die Abfrage betroffen sind und welche schlussendlich zum Ergebnis der Datenbankabfrage beitragen. Dabei ist offensichtlich, dass die Datenbankabfragen nur diejenigen Datensätze betreffen können, die im vorhergehenden Schritt durch den Datengenerator erzeugt wurden. Da diese Datensätze durch die Multimengen D_1, D_5 und D_{10} repräsentiert werden, wurden insgesamt 66 Teilmengen erzeugt (22 Teilmengen pro Multimenge). Jede Teilmenge repräsentiert dahingehend eine Datenbankabfrage, wie sie der *TPC-H*-Abfragegenerator generiert. Die Teilmengen enthalten somit nur die Datenmarken, die durch Ausführung der entsprechenden Datenbankabfrage gelesen und verwendet werden würden. Der Einfachheit halber werden die 66 Teilmengen in drei Relationen organisiert: $Q_1 : \mathbb{N}^+ \rightarrow D \subseteq D_1$, $Q_5 : \mathbb{N}^+ \rightarrow D \subseteq D_5$ und $Q_{10} : \mathbb{N}^+ \rightarrow D \subseteq D_{10}$. Hier würde zum Beispiel die Relation $Q_5(3)$ die Multimenge an Datenmarken als Teilmenge der Multimenge D_5 für die dritte *TPC-H*-Datenbankabfrage widergeben. Für jede dieser 66 Teilmengen wurden entsprechende Transitionsregelsätze für das angepasste Modell für das Datenbanksystem erarbeitet, die den Datenfluss gemäß des Zieles der repräsentierten Datenbankabfrage steuert. Die

Transitionsregelsätze können ebenfalls als Relationen aufgefasst werden. Analog zu den Relation Q_1, Q_5 und Q_{10} können die Relationen $R_1, R_5, R_{10} : \mathbb{N}^+ \rightarrow T \times \mathit{TRL}$ genutzt werden, um für jede Transition aus dem angepassten Modell für das Datenbanksystem die Liste an Transitionsregel[3] widerzugeben.

Die Relationen Q_1, Q_5 und Q_{10} sowie R_1, R_5 und R_{10} können dazu verwendet werden, um Simulationen der 22 *TPC-H*-Datenbankabfragen durchführen zu können. Genauer gesagt, es wird der Datenfluss simuliert, der entsteht, wenn eine dieser Datenbankabfragen auf eine vorher eingefügte Datenbasis ausgeführt wird. Dabei wurde immer dieselbe Vorgehensweise wiederholt: $\forall q \in Q_1 \cup Q_5 \cup Q_{10}$

1. Der vormals gesicherte Zustand der angepassten Modelle für das umgebende Betriebssystem und des Datenbanksystems nach Einfügen der Datenmarken wurde wieder eingespielt. Das bedeutet
   - wenn $q \in Q_1$: Einspielen des gesicherten Zustandes nach Simulation der Multimenge D_1 als Eingabemenge für die Simulation
   - wenn $q \in Q_5$: Einspielen des gesicherten Zustandes nach Simulation der Multimenge D_5 als Eingabemenge für die Simulation
   - wenn $q \in Q_{10}$: Einspielen des gesicherten Zustandes nach Simulation der Multimenge D_{10} als Eingabemenge für die Simulation

   Bis auf die Stellen p_4, p_{11} und p_{12} wurden alle Stellen auf ihre ursprüngliche, initale Markenbelegung zurückgesetzt (vergleiche dazu Tabelle 6.2 auf Seite 168).
2. Setzen der Transitionsregelsätze für die Transitionen im angepassten Modell des Datenbanksystems. Dabei werden abhängig von q die dazugehörigen Transitionsregelsätze
   - $R_1(q)$ falls $q \in Q_1$
   - $R_5(q)$ falls $q \in Q_5$
   - $R_{10}(q)$ falls $q \in Q_{10}$

   genutzt.
3. Start der Simulation mit q als Eingabemenge für die angepassten Modelle für das umgebende Betriebssystem und des Datenbanksystems. Nach erfolgreicher Simulation werden für eine spätere Analyse alle stochastischen Werte der oben genannten Modelle aufgezeichnet. Das sind zum Beispiel die minimale und maximale Markenbelegung aller Stellen und der mittlere Markendurchsatz der Stellen. Zudem wird auch aufgezeichnet, welche Zeit die Simulation an sich gebraucht hat (sogenannte Simulationsdauer). Als wohl wichtigstes Ergebnis wird die simulierte Zeit aufgezeichnet, bis der Datenfluss aufgrund von q als Eingabemenge terminiert. Diese simulierte Zeit entspricht der Antwortzeit der simulierten *TPC-H*-Datenbankabfrage und kann zum direkten Vergleich mit den experimentell gewonnenen Antwortzeiten von Experiment B genutzt werden.

In Anbetracht der Quantitäten an Marken, die bei den Simulationen genutzt wurden, wurde ein groß dimensionierter Simulationsserver genutzt, dessen technische Charakteristika in Tabelle 6.6 aufgelistet sind. Für die Simulationen wurde der QPN-Simulator *QPME* in Version 2 genutzt [KD09]. Der Hauptspeicher von 1.5 TByte des Simulationsservers beschleunigte die Simulation mittels *QPME* bedeutend beziehungsweise machten sie in Anbetracht der Markenquantitäten erst

[3] Vgl. dazu Textabschnitt 5.2.3 ab Seite 129 für die Definition und Beschreibung einer Liste von Transitionsregeln (TRL).

möglich. Insgesamt wurden 198 einzelne Simulationen gemäß oben beschriebener Vorgehensweise durchgeführt (22 *TPC-H*-Datenbankabfragen $\times$ 3 Datenbankgrößen $\times$ 3 Wiederholungen).

Komponente	Beschreibung
CPU	4$\times$*AMD Opteron 6380* mit je 16 Kernen, 2.5 GHz pro Kern
Hauptspeicher	1.5 TByte (48$\times$32 GByte, DDR-3, ECC)
Betriebssystem	*Ubuntu Server 14.04 x64*
Java Virtual Machine	*Oracle Java SE 7u71*

Tabelle 6.6: Charakteristika des Simulationsservers

Simulationsergebnisse In einem letzten Schritt wurden die aufgezeichneten Werte der Simulationen untersucht, um die Genauigkeit des Modelles für das umgebende Betriebssystem und des Modelles für das Datenbanksystem hinsichtlich der Antwortzeiten und der Energieeffizienz zu errechnen, wobei Experiment B simuliert wurde. Für die Berechnung der Genauigkeit der Antwortzeiten der 22 simulierten *TPC-H*-Datenbankabfragen wurde zunächst der Durchschnitt über alle simulierten Datenbankgrößen von 1, 5 und 10 GByte berechnet. Sie wurden dann mit denjenigen verglichen, die experimentell gewonnen wurden (vergleiche Textabschnitt 4.1.2 ab Seite 78). Gemäß der Definition der Genauigkeit auf Seite 165 wird sie als Prozentsatz angegeben.

Die mittlere Genauigkeit über alle simulierten *TPC-H*-Datenbankgrößen ist für jede der 22 *TPC-H*-Datenbankabfragen in Abbildung 6.2 auf Seite 174 dargestellt. Die mittlere Genauigkeit für den gesamten simulierten *TPC-H*-Benchmark beträgt 63.45 Prozent. Das bedeutet, dass die simulierte und experimentelle Antwortzeit im Schnitt um 26.55 Prozent voneinander abweichen.

In Abbildung 6.2 fällt auf, dass die durchschnittliche Genauigkeit für die *TPC-H*-Datenbankabfrage 1, 9 und 21 im Vergleich zu den Genauigkeiten der übrigen stark abweicht. In den Erläuterungen der experimentellen Ergebnisse von Experiment B in Textabschnitt 4.1.2 wurde das negative Verhalten des verwendeten Datenbanksystems *Postgresql* im Bezug auf diese drei *TPC-H*-Datenbankabfragen beschrieben. Dabei kam es zur Aussetzung des *Postgresql*-Betriebssystemprozesses aufgrund von heftigen Pufferoperationen (*swapping*) und als Folge daraus zu übermäßig langen Antwortzeiten. Eine genauere Analyse der aufgezeichneten Simulationsstatistiken der betreffenden Stellen im angepassten Modell für das umgebende Betriebssystem zeigt, dass dieses Verhalten in den Simulationen nicht beobachtet werden konnte. Daraus resultierten dementsprechende kürzere simulierte Antwortzeiten. Als Konsequenz weichen die simulierten und experimentellen Antwortzeiten für die oben genannten drei *TPC-H*-Datenbankabfragen stark voneinander ab, wodurch sich auch die relativ geringe Genauigkeit erklärt.

Im Bezug auf die Berechnung der Genauigkeit der Energieeffizienz wurden die aufgezeichneten stochastischen Werte der betreffenden Ressourcenstellen p_4, p_5 und p_7 (Massen- und Arbeitsspeicher, Prozessor) als Basis genutzt, um zunächst einen simulierten Energieverbrauch zu errechnen. Mit den experimentellen Ergebnissen von Experiment A, die in Textabschnitt 4.1.1 ab Seite 73 beschrieben sind, liegen Energieverbrauchswerte der technischen Komponenten vor, die im angepassten Modell für das umgebende Betriebssystem simuliert wurden. Man kann also den maximalen Markendurchsatz pro simulierter Sekunde der drei oben genannten Stellen nutzen und ihn mit den experimentellen Energieverbrauchswerten in Relation setzen. Als Resultat kann man einen Energieverbrauchswert pro Ressourcenmarke errechnen, die aus den oben genannten drei Ressourcenstellen während einer Simulation allokiert oder deallokiert wurde. Multipliziert man diesen Energieverbrauchswert pro Ressourcenmarke mit dem durchschnittlichen Ressour-

cenmarkendurchsatz der Ressourcenstelle und der simulierten Antwortzeit, so erhält man den simulierten Energieverbrauch der Ressourcenstelle für eine Simulation. Summiert man diese Werte, erhält man einen simulierten Gesamtenergieverbrauch. Er kann dann gemäß Gleichung 3.3 für die Energieeffizienz auf Seite 64 zusammen mit der simulierten Antwortzeit genutzt werden, um schlussendlich die simulierte Energieeffizienz zu errechnen.

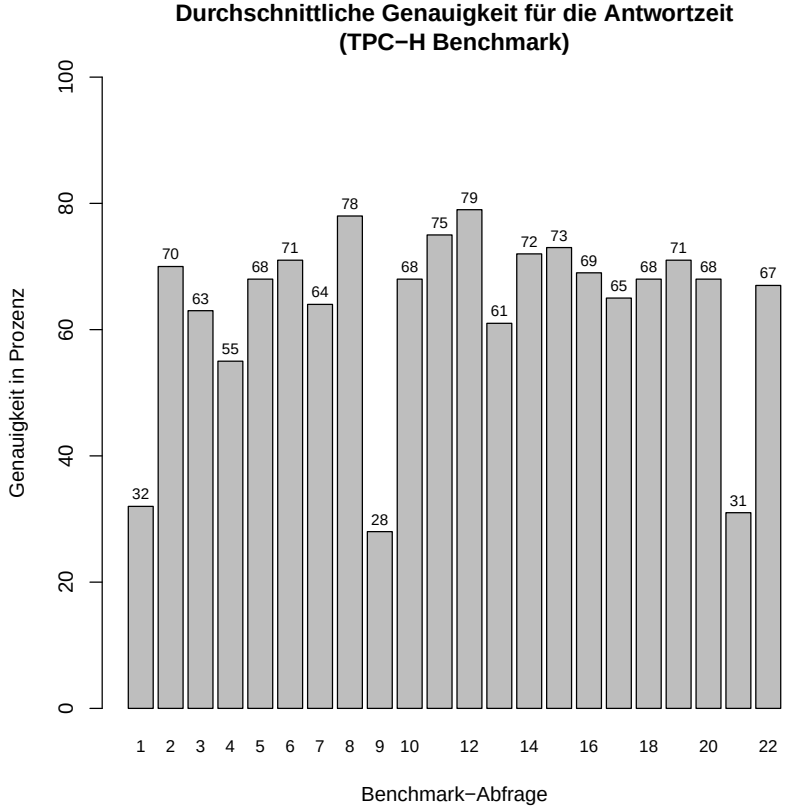


Abbildung 6.2: Mittlere Genauigkeit der Simulation von Experiment B für die Antwortzeiten des *TPC-H*-Benchmarks

Analog zu den simulierten und experimentellen Antwortzeiten wurden auch die simulierten und experimentellen Werte für die Energieeffizienz miteinander verglichen, um eine Genauigkeit laut Definition 13 im Bezug auf die Energieeffizienz zu ermitteln. Die mittlere Genauigkeit über alle simulierten *TPC-H*-Datenbankgrößen von 1, 5 und 10 GByte ist als Prozentangabe in Abbildung 6.3 auf Seite 175 dargestellt. Dabei werden die Genauigkeitsangaben für jede der 22 *TPC-H*-Datenbankabfragen aufgeschlüsselt angegeben. Die mittlere Genauigkeit für den gesamten simulierten *TPC-H*-Benchmark beträgt 78.09 Prozent. Das bedeutet, dass die simulierten und experimentellen Energieeffizienzwerte im Schnitt um 21.91 Prozent voneinander abweichen.

Auch in Abbildung 6.3 zeigt sich, dass die Genauigkeitswerte der Energieeffizienz für die *TPC-H*-Datenbankabfragen 1,9 und 21 signifikant von den übrigen abweichen. Der Grund ist derselbe, wie er bereits für die Abweichungen der Genauigkeitswerte für die Antwortzeit beschrieben ist. Allerdings fallen die Abweichungen der Genauigkeitswerte für die Energieeffizienz nicht so

stark aus, wenn man sie mit denen der Antwortzeiten vergleicht. Der Grund dafür ist, dass die Energieeffizienz gemäß Gleichung 3.3 als Division zwischen der (simulierten) Antwortzeit und dem (simulierten) Energieverbrauch definiert ist. Durch die Division werden die signifikant abweichenden Antwortzeiten mit den weniger signifikant abweichenden Energieverbrauchswerten dividiert. Das führt dazu, dass die Energieeffizienz zwar auch auch abweicht, aber weniger stark ausgeprägt ist.

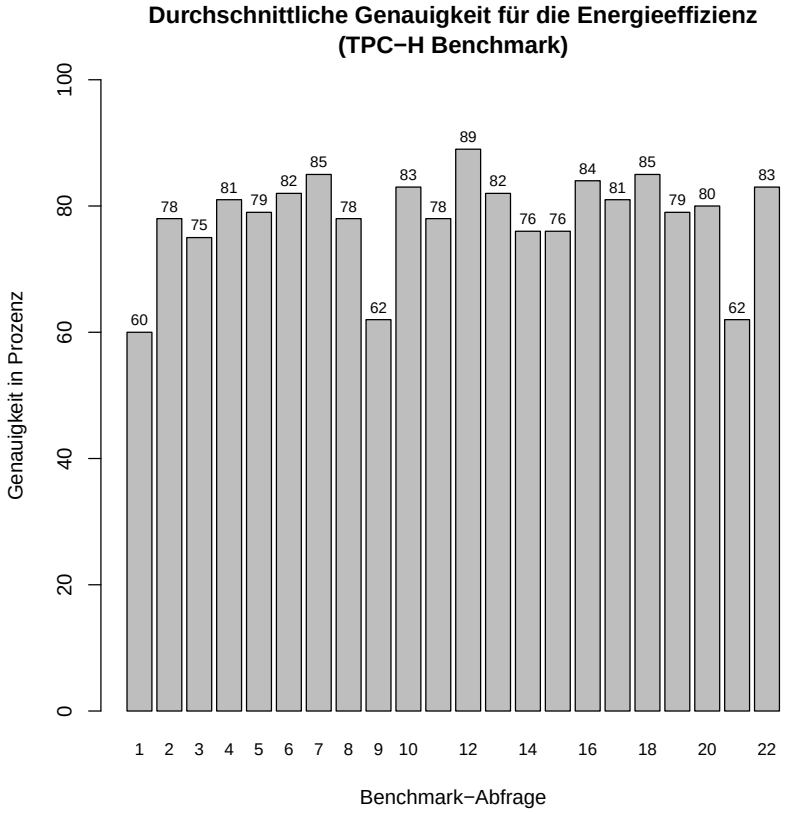


Abbildung 6.3: Mittlere Genauigkeit der Simulation von Experiment B für die Energieeffizienz des *TPC-H*-Benchmarks

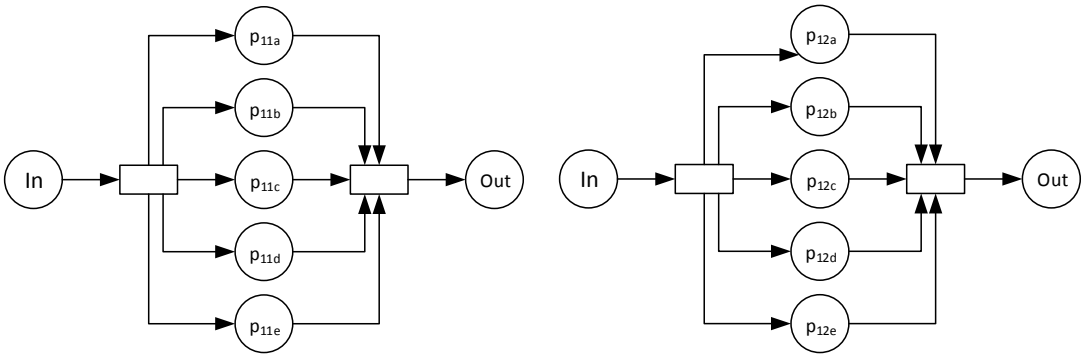
6.2.3 Simulation von Experiment C (StarSchema)

Experiment C kann in nahezu gleicher Weise simuliert werden wie Experiment B. Der Grund dafür ist, dass beide Experimente hinsichtlich Versuchsaufbau und -durchführung fast identisch sind, wenn man die Beschreibungen beider Experimente auf den Seiten 78 und 84 miteinander vergleicht. Als Ergebnis muss nur das Modell für das Datenbanksystem für die Simulation von Experiment C angepasst werden. Die Anpassung des Modelles für das umgebende Betriebssystem ist bereits in Textabschnitt 6.2.1 beschrieben worden.

Modellanpassungen Experiment C unterscheidet sich von Experiment B nur dadurch, dass ein anderer Datenbanksystem-Benchmark verwendet wurde. In Experiment C wurde der *StarSchema*-Benchmark (*SSB*) genutzt, während Experiment B den *TPC-H*-Benchmark eingesetzt wurde. Die technische Plattform und das verwendete Datenbanksystem sind bei beiden Experimenten gleich. Zudem sind auch dieselben Benchmarkkonfigurationen für beide Experimente verwendet wurde, wobei 18 Testserien entstanden. Diese Testserien sind eine Kombination aus einer Datenbankgröße (1, 5 und 10 GByte), einem Hauptspeicheranteil (1, 2.5 und 5 GByte) und einem Verbindungstyp (Einfach- und Mehrfachverbindung).

In einem ersten Schritt wurde der *SSB*-Datengenerator dahingehend analysiert, welche Struktur und welche Quantitäten an Datensätzen generiert werden. Zusätzlich wurde auch analysiert, welche Datenbankstrukturen der *SSB*-Datengenerator erstellt, zum Beispiel Datenbanktabellen und -indizes. Das Ergebnis der Analyse war, dass der *SSB*-Datengenerator Daten nach dem relationalen Datenmodell für insgesamt fünf Tabellen generiert. Zudem werden 5 Datenbankindizes erstellt.

Auf Basis der Analyse des *SSB*-Datengenerators wurden das Modell für das Datenbanksystem, das in Abbildung 5.7 auf Seite 143 abgebildet und in Textabschnitt 5.4 beschrieben ist, im Hinblick auf die Simulation von Experiment C angepasst. Das innere Modell von Stelle p_{11} wurde dergestalt konzipiert, dass es die fünf *SSB*-Tabellen repräsentiert. Auf Abbildung 6.4(a) ist dieses innere Modell dargestellt. Es enthält fünf Stellen p_{11a} bis p_{11e}, wobei jede dieser Stellen die Datenbankdatei repräsentiert, in der der Inhalt einer *SSB*-Tabelle gespeichert wird.



(a) Stelle p_{11} (Datenbankdateien) (b) Stelle p_{12} (Indexdateien)

Abbildung 6.4: Innere Modelle der Stellen p_{11} und p_{12} im Modell für das Datenbanksystem zur Simulation von Experiment C

Das innere Modell von Stelle p_{12} ist in Abbildung 6.4(b) dargestellt. Es wurde auf Basis der Analyse des *SSB*-Datengenerators in einer Weise modelliert, dass es die fünf Datenbankindizes repräsentiert, wie der Datengenerator es vorsieht. Die fünf Stellen p_{12a} bis p_{12e} repräsentieren dabei jeweils einen Datenbankindex. Vergleicht man das innere Modell, dass in Abbildung 6.4(a) dargestellt ist, mit dem auf Abbildung 6.4(b), so erkennt man im Bezug auf die Struktur keinerlei Unterschiede. Ein Ansatz ist es, einfach ein Modell zu erstellen und dann zwei Instanzen dieses Modell als inneres Modell für die Stellen p_{11} und p_{12} zu verwenden. Dabei ergibt sich aber das Problem, dass beide Instanzen zwar Stellen besitzen, die unabhängig voneinander sind, aber dieselben Transitionsregelsätze besitzen würden. Dieses Verhalten ist aber hinsichtlich der Simulation von Experiment C nicht erwünscht, da damit der benötigte Datenfluss der Marken nicht erreicht werden kann.

In Analogie zur Simulation B, die im vorhergehenden Textabschnitt beschrieben wurde, ist der für Experiment C verwendete Datengenerator auch dahingehend analysiert worden, in welcher Struktur und Anzahl Datensätze generiert werden. Im Experiment C sind mittels des Datengenerators Datenbanken mit einer Gesamtgröße von 1, 5 und 10 GByte Daten erstellt worden. Als Konsequenz sind drei Multimengen D_1, D_5 und D_{10} erstellt worden, die Datenmarken und Submarken enthalten. Die Datenmarken repräsentieren dabei jeweils einen generierten Datensatz, wobei die Submarken die Datenfelder des Datensatzes abbilden.

Jede der Multimengen D_1, D_5 und D_{10} setzt sich dabei aus fünf Teilmengen zusammen. Jede Teilmenge enthält die Datenmarken als Reflektion der generierten Datensätze für eine *SSB*-Tabelle. In Tabelle 6.7 kann die Anzahl der Datenmarken pro *SSB*-Tabelle, also einer Teilmenge, direkt abgelesen werden. Betrachtet man zum Beispiel die Multimenge D_5, so können die Anzahlen an Datenmarken der Teilmengen in der mittleren Spalte von Tabelle 6.7 direkt abgelesen werden. Zudem können die Anzahl an Submarken pro Datenmarke in der Spalte abgelesen werden, die sie direkt rechts neben der mittleren Spalte von Tabelle 6.7 befindet. In der unterste Zeile von Tabelle 6.7 kann zusätzlich abgelesen werden, wieviele Datenmarken insgesamt in den drei Multimengen D_1, D_5 und D_{10} verzeichnet sind. Die Gesamtanzahl an Submarken, die ebenfalls in dieser Zeile der Tabelle aufgelistet sind, ergibt sich dabei als Produkt der Gesamtanzahl an Datenmarken mit der Summe der Submarken.

	← *1 GByte Datenbank* →		← *5 GByte Datenbank* →		← *10 GByte Datenbank* →	
***SSB*-Tabelle**	**Datenmarken**	**Submarken**	**Datenmarken**	**Submarken**	**Datenmarken**	**Submarken**
`part`	200.000	9	600.000	9	800.000	9
`supplier`	2.000	7	10.000	7	20.000	7
`customer`	30.000	8	150.000	8	300.000	8
`lineorder`	6.000.000	17	30.000.000	17	60.000.000	17
`date`	2.555	8	2.555	8	2.555	8
$\sum$	6.234.555	305.493.195	30.762.555	1.507.365.195	61.122.555	2.995.005.195
	← *Multimenge* D_1 →		← *Multimenge* D_5 →		← *Multimenge* D_{10} →	

Tabelle 6.7: Anzahl der Datenmarken und Submarken in Multimenge D für verschiedene *SSB*-Datenbankgrößen

Simulationsdurchführung Da die Modelle für das umgebende Betriebssystem und für das Datenbanksystem für eine Simulation angepasst sind, können die Multimengen D_1, D_5 und D_{10} jeweils als Eingabemenge für eine Simulation genutzt werden. Dadurch wurden die Stellen p_{11} (Datenbankdateien) und p_{12} (Datenbankindizes) befüllt. Die drei Simulationen wurden dabei gemäß der Beschreibung in Textabschnitt 5.4.1 ab Seite 144 durchgeführt. Nach jeder erfolgreichen Simulation wurde der Zustand, also die Markenbelegung der angepassten Modelle für das umgebende Betriebssystem und für das Datenbanksystem separat gesichert. Diese Sicherungen bildeten die Basis für die weiteren Simulationen von Experiment C.

Durch die drei durchgeführten Simulationen sind jeweils die Stellen befüllt worden, die in den Abbildungen 6.4(a) und 6.4(b) dargestellt sind. Das sind im wesentlichen die Stellen der inneren Modelle von den beiden Stellen Stellen p_{11} und p_{12} des angepassten Modelles für das Datenbanksystem. Die Anzahl der Marken entsprechen der Anzahl der Datenmarken, wie sie in den Multimengen D_1, D_5 und D_{10} vorkommen. Der Grund dafür ist, dass es sich um Assoziativmarken im Sinne der Beschreibung ab Seite 128 handelt. Hierbei ist wichtig, wieviele zusätzliche Farben für die Relation zwischen Daten- und Massenspeichermarken notwendig wurden. Die Quantitäten sind in den beiden Tabellen 6.8 und 6.9 auf Seite 178 aufgelistet.

		← *Datenbankgröße* →		
Stelle	***SSB*-Tabelle**	**1 GByte**	**5 GByte**	**10 GByte**
p_{11a}	`part`	3.321	9.961	13.282
p_{11b}	`supplier`	36	176	352
p_{11c}	`customer`	450	2.247	4.493
p_{11d}	`lineorder`	81.055	405.274	810.547
p_{11e}	`date`	38	38	38

Tabelle 6.8: Anzahl der unterschiedlichen Farben der Assoziativmarken im inneren Modell von Stelle p_{11}

		← *Datenbankgröße* →		
Stelle	***SSB*-Tabelle**	**1 GByte**	**5 GByte**	**10 GByte**
p_{12a}	`part`	247	816	1063
p_{12b}	`supplier`	4	10	12
p_{12c}	`customer`	38	124	160
p_{12d}	`lineorder`	7.362	24.452	31.813
p_{12e}	`date`	5	5	5

Tabelle 6.9: Anzahl der unterschiedlichen Farben der Assoziativmarken im inneren Modell von Stelle p_{12}

Im gleichen Sinne wie für die Simulation von Experiment B wurde auch der Abfragegenerator analysiert, der im Experiment C genutzt wurde. Bei dieser Analyse wurden die 13 einzelnen Datenbankabfragen daraufhin untersucht, welche Datensätze bei der Ausführung der betreffenden Datenbankabfrage betroffen sind. Zusätzlich wurde auch analysiert, welche dieser betroffenen Datensätze zum eigentlichen Ergebnis der Datenbankabfrage beitragen. Dabei muss beachtet werden, dass die 13 einzelnen Datenbankabfragen des *SSB*-Benchmarks nur auf einer Datenbasis arbeiten, die vom Datengenerator des Benchmarks erzeugt wurde. Da die Datenbasis durch die Multimengen D_1, D_5 und D_{10} repräsentiert werden, wurden für jede dieser Multimenge 13 Teilmengen erstellt, also insgesamt 39 Teilmengen. Jede Teilmenge repräsentiert dahingehend eine Datenbankabfrage, wie sie der *SSB*-Abfragegenerator generiert. Die Teilmengen enthalten somit nur die Datenmarken, die durch Ausführung der entsprechenden Datenbankabfrage gelesen und verwendet werden würden. Auch hier werden der Einfachheit halber die 39 Teilmengen in drei Relationen organisiert: $Q_1 : \mathbb{N}^+ \rightarrow D \subseteq D_1$, $Q_5 : \mathbb{N}^+ \rightarrow D \subseteq D_5$ und $Q_{10} : \mathbb{N}^+ \rightarrow D \subseteq D_{10}$. Die Relation $Q_1(5)$ würde zum Beispiel die Multimenge an Datenmarken als Teilmenge der Multimenge D_5 für die dritte *SSB*-Datenbankabfrage widergeben, wie sie in der Spezifikation des *SSB* definiert ist. Sofern diese Teilmengen als Eingabemenge für eine Simulation genutzt werden, so ist offensichtlich, dass immer andere Datenflüsse modelliert werden müssen, da die enthaltenen Datenmarken eine spezielle Verarbeitung brauchen. Das ergibt sich auch daraus, dass die Teilmengen immer unterschiedliche *SSB*-Datenbankabfragen repräsentieren. Dementsprechend wurden auch 39 Transitionsregelsätze für das angepasste Modell für das Datenbanksystem erarbeitet, die den Datenfluss gemäß des Zieles der repräsentierten Datenbankabfrage steuert.

Auch hier können die Transitionsregelsätze in drei Relationen $R_1, R_5, R_{10} : \mathbb{N}^+ \rightarrow T \times \mathit{TRL}$ organisiert werden. Sie können genutzt werden, um für jede Transition aus dem angepassten Modell für das Datenbanksystem die Liste an Transitionsregel[4] widerzugeben.

Die Relationen Q_1, Q_5 und Q_{10} sowie R_1, R_5 und R_{10} können dazu verwendet werden, um Simulationen der 13 *SSB*-Datenbankabfragen durchführen zu können. Genauer gesagt, es wird der Datenfluss simuliert, der entsteht, wenn eine dieser Datenbankabfragen auf eine vorher eingefügte Datenbasis ausgeführt wird. Da die Vorgehensweise immer identisch ist, sei zur Erläuterung der Vorgehensweise die beiden Menge $X \subset \mathbb{N}^+ = \{1, 5, 10\}$ und $Y \subset \mathbb{N}^+ = \{1, 2, 3, \ldots, 13\}$ festgelegt:
$\forall x \in X : \forall q \in Y$

1. Der vormals gesicherte Zustand der angepassten Modelle für das umgebende Betriebssystem und des Datenbanksystems nach Einfügen der Datenmarken wird wieder eingespielt. Das bedeutet, dass zum Einspielen der gesicherten Zustand der Simulation genutzt wird, in der D_x als Eingabemenge benutzt wurde. Bis auf die Stellen p_4, p_{11} und p_{12} wurden alle Stellen auf ihre ursprüngliche, initale Markenbelegung zurückgesetzt (vergleiche dazu Tabelle 6.7 auf Seite 177).
2. Die Transitionsregelsätze für die Transitionen im angepassten Modell des Datenbanksystems werden festgelegt. Dabei werden die dazugehörigen Transitionsregelsätze $R_x(q)$ genutzt.
3. Start der Simulation mit $Q_x(q)$ als Eingabemenge für die angepassten Modelle für das umgebende Betriebssystem und des Datenbanksystems. Eine Simulation gilt dann als erfolgreich, wenn die Eingabemenge komplett verarbeitet wurde und der Datenfluss terminiert. Nach einer solchen erfolgreichen Simulation werden für eine spätere Analyse alle stochastischen Werte der oben genannten Modelle aufgezeichnet. Das sind zum Beispiel die minimale und maximale Markenbelegung aller Stellen und der mittlere Markendurchsatz der Stellen. Zudem wird auch aufgezeichnet, welche Zeit die Simulation an sich gebraucht hat (Simulationsdauer). Ein substantielles Simulationsergebnis ist die simulierte Zeit, die es braucht, bis die Simulation erfolgreich war. Diese simulierte Zeit entspricht der Antwortzeit der simulierten *SSB*-Datenbankabfrage und kann zum direkten Vergleich mit den experimentell gewonnenen Antwortzeiten von Experiment C genutzt werden.

Für die durchgeführten Simulationen wurde derselbe Simulationsserver genutzt, wie er auch für die Simulation von Experiment B verwendet wurde. Die Charakteristik dieses Simulationsservers sind in Tabelle 6.6 auf Seite 173 aufgelistet. Auf dem Simulationsserver wurde ebenso dieselbe Simulationssoftware eingesetzt (QPN-Simulator *QPME* in Version 2, [KD09]). Insgesamt wurden 117 einzelne Simulationen gemäß oben beschriebener Vorgehensweise durchgeführt (13 *SSB*-Datenbankabfragen $\times$ 3 Datenbankgrößen $\times$ 3 Wiederholungen).

Simulationsergebnisse Wie zuvor in Evaluation der Ergebnisse der Simulationen von Experiment B, können die Simulationsergebnisse des Experimentes C dazu verwendet werden, um die Genauigkeit des Modelles für das umgebende Betriebssystem und des Modelles für das Datenbanksystem hinsichtlich der Antwortzeiten und der Energieeffizienz zu errechnen. So wurde zunächst hinsichtlich der simulierten Antwortzeiten der Durchschnitt über alle simulierten Datenbankgrößen von 1, 5 und 10 GByte berechnet. Diese können dann direkt mit denen verglichen

[4] Vergleiche dazu Textabschnitt 5.2.3 auf Seite 129 für die Definition und Beschreibung einer Liste von Transitionsregeln (TRL).

werden, die experimentell gewonnen wurden und im Textabschnitt 4.1.3 ab Seite 84 beschrieben sind. Der prozentuale Unterschied ergibt dann gemäß der Definition 13 auf Seite 165 die Genauigkeit.

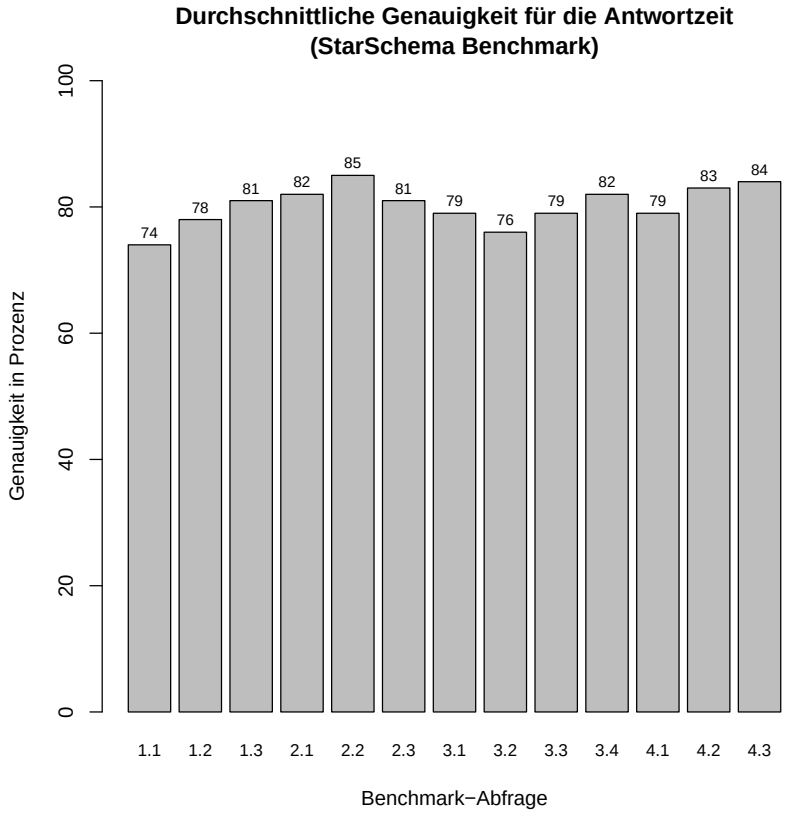


Abbildung 6.5: Mittlere Genauigkeit der Simulation von Experiment C für die Antwortzeiten des *StarSchema*-Benchmarks

Die mittlere Genauigkeit über alle simulierten *SSB*-Datenbankgrößen ist für jede der 13 *SSB*-Datenbankabfragen in Abbildung 6.5 dargestellt. Die mittlere Genauigkeit für den gesamten simulierten *StarSchema*-Benchmark beträgt 80.23 Prozent. Das bedeutet, dass die simulierte und experimentelle Antwortzeit im Schnitt um 19.77 Prozent voneinander abweichen.

Vergleicht man die mittleren Genauigkeiten der Antwortzeiten für den simulierten *TPC-H*-Benchmark in Abbildung 6.2 auf Seite 174 mit denjenigen des simulierten *StarSchema*-Benchmarks in Abbildung 6.5 miteinander, so fällt auf, dass sich ein wesentlich gleichmäßigerer, robusterer Trend abzeichnet. Das bedeutet, es sind keine signifikanten "Ausreißer" erkennbar. Zudem fallen die in Abbildung 6.5 dargestellten Genauigkeiten im Schnitt höher aus als diejenigen von Abbildung 6.2. Damit soll ausgedrückt werden, dass eine Simulation des *StarSchema*-Benchmarks um ungefähr 17 Prozent näher die realen, experimentellen Antwortzeiten widergibt als die Simulation des *TPC-H*-Benchmarks.

Für die Berechnung der simulierten Energieeffizienzwerte des *StarSchema*-Benchmarks wurde auf dasselbe Verfahren zurückgegriffen, dass auch für diejenigen des *TPC-H*-Benchmarks genutzt

wurde. Es ist im vorhergehden Textabschnitt beschrieben. Im Prinzip wurden die aufgezeichneten stochastischen Werte der betreffenden Ressourcenstellen p_4, p_5 und p_7 (Massen- und Arbeitsspeicher, Prozessor) als Basis genutzt, um zunächst einen simulierten Energieverbrauch zu errechnen. Zudem liegen mit den experimentellen Ergebnissen von Experiment A Energieverbrauchswerte der technischen Komponenten vor, die im angepassten Modell für das umgebende Betriebssystem simuliert wurden. Anhand des maximalen Markendurchsatz pro simulierter Sekunde der drei oben genannten Stellen und den experimentellen Energieverbrauchswerten kann man einen Energieverbrauchswert pro Ressourcenmarke errechnen, die aus den oben genannten drei Ressourcenstellen während einer Simulation allokiert oder deallokiert wurden. Durch Multiplikation dieses Energieverbrauchswertes pro Ressourcenmarke mit dem durchschnittlichen Ressourcenmarkendurchsatz der Ressourcenstelle und der simulierten Antwortzeit erhält man den simulierten Energieverbrauch der Ressourcenstelle für eine Simulation. Dieser simulierte Energieverbrauch kann dann gemäß Gleichung 3.3 für die Energieeffizienz auf Seite 64 zusammen mit der simulierten Antwortzeit genutzt werden, um schlussendlich die simulierte Energieeffizienz zu errechnen.

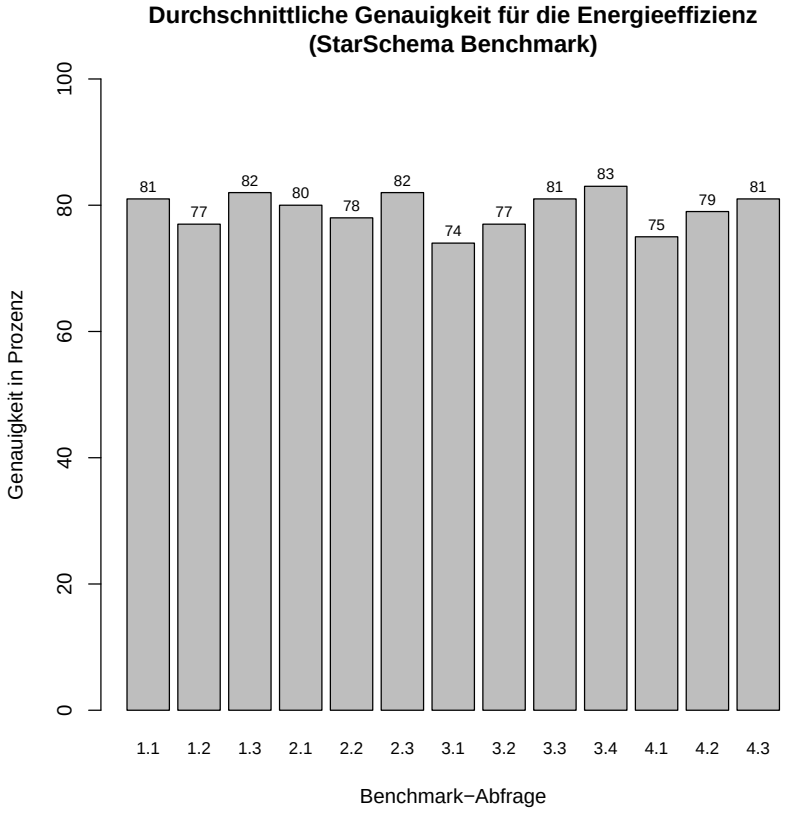


Abbildung 6.6: Mittlere Genauigkeit der Simulation von Experiment C für die Energieeffizienz des *StarSchema*-Benchmarks

Die experimentellen Energieeffizienzwerte des *StarSchema*-Benchmarks aus Experiment C wurden mit den simulierten verglichen, um eine Genauigkeit laut Definition 13 im Bezug auf die Energieeffizienz zu ermitteln. Die mittlere Genauigkeit über alle simulierten *SSB*-Datenbankgrö-

ßen von 1, 5 und 10 GByte ist als Prozentangabe in Abbildung 6.6 auf Seite 181 dargestellt, wobei die Genauigkeitsangaben für jede der 13 *SSB*-Datenbankabfragen aufgeschlüsselt angegeben werden. Die mittlere Genauigkeit für den gesamten simulierten *StarSchema*-Benchmark beträgt 79.23 Prozent. Das bedeutet, dass die simulierten und experimentellen Energieeffizienzwerte im Schnitt um 20.77 Prozent voneinander abweichen.

6.3 Evaluation der simulierten horizontalen *scale-up*-Szenarien

In gleicher Weise, wie die QPN-Modelle für das umgebende Betriebssystem und des Datenbanksystems im vorherigen Textabschnitt 6.2 evaluiert wurden, können die erweiterten Pendants hinsichtlich ihrer Genauigkeit untersucht werden. Das betrifft im wesentlichen die QPN-Modelle, die in den Textabschnitten 5.5, 5.6 und 5.7 beschrieben sind. Genauer gesagt, betrifft es die Modelle für ein Computernetzwerk und die erweiterten Modelle für das umgebende Betriebssystem und des Datenbanksystems. Mit diesen drei Modellen ist es möglich, ein verteiltes Datenbanksystem nachzustellen und für Simulationen des Datenflusses zu nutzen, wie er bei der Simulation einer Datenbankanwendung entsteht. Aus Vergleichsgründen ist diese Datenbankanwendung ein Benchmark für ein verteiltes Datenbanksystem, wie er zum Beispiel bei den beiden Experimenten E und F verwendet wurde.

Die beiden genannten Experimente untersuchten das Skalierungsverhalten des verteilten Datenbanksystems *Cassandra* anhand des *Yahoo Cloud Serving Benchmark* (*YCSB*). Dabei wurde untersucht, wie die Performance in Form der Antwortzeit und der Energieverbrauch von *Cassandra* in Relation zur *Cassandra*-Clustergröße skaliert. Beides wurde auch dahingehend benutzt, um Aussagen zur Energieeffizienz zu machen. Da durch die Durchführung beider Experimente experimentelle Messergebnisse zur Verfügung stehen, können diese dafür genutzt werden, um die Genauigkeit der QPN-Modelle für die erwähnten Metriken zu evaluieren. Gleichermaßen soll auch validiert werden, ob die QPN-Modelle in der Lage sind, ein verteiltes Datenbanksystem, eine Datenbankanwendung in Form eines Benchmarks und die verwendete hardwaretechnische Plattform zu modellieren und zu simulieren.

Der verbleibende Teil dieses Textabschnittes 6.3 gliedert sich in zwei Unterabschnitte. Jeweils ein Unterabschnitt beschreibt die Anpassung der QPN-Modelle zur Simulation von Experiment E beziehungsweise F sowie die Durchführung der Simulationen. Die gewonnenen Simulationsergebnisse, zum Beispiel die simulierten Antwortzeiten oder der simulierte Energieverbrauch, werden dann genutzt, um die Genauigkeit der QPN-Modelle zu errechnen. Dabei wird die Definition 13 der Genauigkeit von Seite 165 genutzt.

6.3.1 Simulation von Experiment E (*Cassandra* mit 7 Clusterknoten und *YCSB*)

Zur Erklärung der Simulation von Experiment E sei zunächst kurz rekapituliert, welches Ziel dieses Experiment hatte[5]. Es untersuchte das Skalierungsverhalten des verteilten Datenbanksystems *Cassandra*, wobei zwei Szenarien getestet wurden, die den verwendeten Datenbank-Benchmark *YCSB* als Konfigurarion dienten. Szenario A sah eine feste und Szenario B eine ansteigende Da-

[5] Für eine ausführliche Beschreibung des Experimentes, des Versuchsaufbaus sowie eine Analyse der experimentellen Ergebnisse sei auf Textabschnitt E ab Seite 98 verwiesen.

tenmenge als Datenbasis vor, die ein *Cassandra*-Cluster speichern musste. Es wurden dabei *Cassandra*-Cluster mit ansteigender Anzahl zwischen drei und sieben Clusterknoten getestet. Auf die Datenbasis eines *Cassandra*-Clusters griffen dann die drei Arbeitslasten `read` (ausschließlich lesende Zugriffe), `write` (ausschließlich modifizierende Zugriffe) und `mixed` (zuerst lesende, dann modifizierende Zugriffe) zu. Die Zugriffe wurden zusätzlich nach zwei Verteilungen (diskrete Gleichverteilung und *Zipf*-Verteilung) auf die beteiligten *Cassandra*-Cluster verteilt.

Modellanpassungen Um Experiment E zu simulieren, ist es in einem ersten Arbeitsschritt notwendig, die QPN-Modelle anzupassen, sodass der Versuchsaufbau von Experiment E nachgestellt werden kann. Genauer gesagt wurden dafür die einzelnen Schritte nacheinander ausgeführt, wie sie in der generellen Arbeitsweise der erweiterten QPN-Modelle ab Seite 155 detailliert beschrieben sind.

Ingesamt entstand ein hierarchisches QPN-Modell mit drei Hierarchieebenen, wie es in Abbildung 5.10 auf Seite 153 dargestellt ist. Auf oberste Ebene wurden in der Summe sieben QPN-Modelle entwickelt, die die getesteten *Cassandra*-Cluster repräsentieren. Der generelle Aufbau dieser Modelle ist in Abbildung 6.7 dargestellt. Jedes dieser Modelle besitzt die Stelle B_1 und *BP* sowie eine Anzahl von Stellen B_2 bis maximal B_8. Die Stelle mit innerem Modell B_1 repräsentiert den *YCSB*-Client. Die Stellen mit innerem Modell B_2 bis B_8 repräsentieren die beteiligten *Cassandra*-Clusterknoten (minimal drei, maximal sieben). Die Stelle *BP* repräsentiert den Switch, der alle Clusterknoten miteinander verbindet und den Datenaustausch zwischen den Clusterknoten ermöglicht. Die in Abbildung 6.7 gezeigten Transitionen modellieren den Up- und Downloadpfad zu den modellierten Clusterknoten. Zusammengefasst kann festgestellt werden, dass durch die sieben QPN-Modelle der Versuchsaufbau von Experiment E nachgestellt wurde, wie er in Abbildung 4.14 auf Seite 101 dargestellt ist.

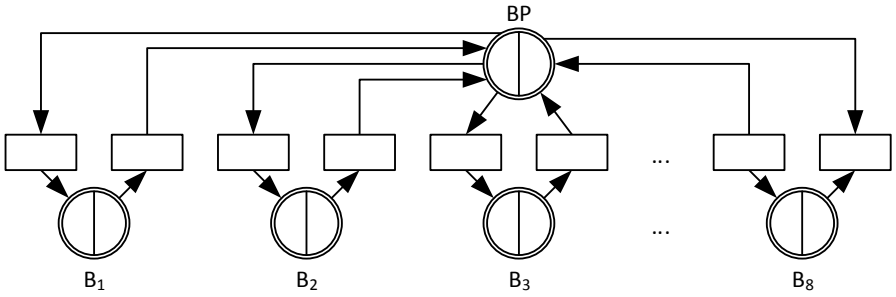


Abbildung 6.7: Oberste Ebene des hierarchischen QPN-Modells zur Simulation von Experiment E. Die Stelle B_1 repräsentiert den *YCSB*-Client, der auf einen *Cassandra*-Cluster zugreift. Die Clusterknoten werden durch die Stellen B_2 bis B_8 repräsentiert. Stelle *BP* repräsentiert den Switch, mit denen die Clusterknoten untereinander Daten austauschen können.

Auf oberster Ebene, also das QPN-Modell wie es in Abbildung 6.7 dargestellt ist, können gemäß der Beschreibung der QPN-Modelle nur Netzwerkmarken zwischen den dargestellten Stellen ausgetauscht werden, womit der Datenfluss des zugrunde liegenden Netzwerks des *Cassandra*-Clusters modelliert werden kann. Als inneres Modell für die Stelle *BP* aus Abbildung 6.7 wird ein QPN-Modell ausgeführt, wie es in Abbildung 6.8 aus Seite 184 dargestellt ist. Dieses QPN-Modell folgt den Modellierungsvorgaben für einen inneren Knoten in einem Computernetzwerk

([RK14], siehe Textabschnitt 5.5 ab Seite 149). Das innere Modell besteht dabei aus drei Transitionen (t_{rx}, t_{tx} und $t_{routing}$) sowie einer Anzahl von Stellen mit Warteschlange BP_a bis BP_g. Die Anzahl der Stellen korrespondiert mit der Anzahl der Stellen B_1 bis B_8. Jede der Stellen BP_a bis BP_g repräsentiert eine Anschlussschnittstelle (sogenannter *port*) des Switches, an der ein Clusterknoten per Datenkabel angeschlossen ist.

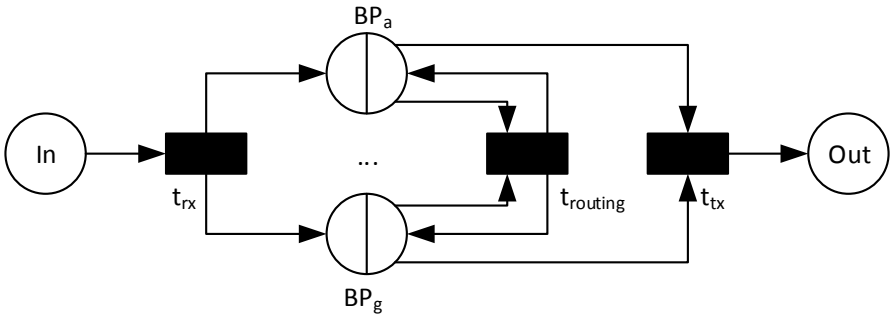


Abbildung 6.8: Inneres Modell der Stelle *BP* aus dem hierarchischen QPN-Modell zur Simulation von Experiment E, das in Abbildung 6.7 dargestellt ist

Die Funktionsweise des inneren Modells von Abbildung 6.8 kann wie folgt beschrieben werden:

1. Netzwerkmarken, die von den Stellen B_1 bis B_8 stammen, werden in der Stelle *In* abgelegt. Damit wird der Uploadpfad eines Clusterknotens modelliert. Die Farben der Netzwerkmarken sind so kodiert, dass sich damit die Quelle und das Ziel der Netzwerkmarke ergibt. Genauer gesagt, die Farbe der Netzwerkmarke gibt an, von welcher Stelle B_1 bis B_8 es stammt und zu welcher Stelle B_1 bis B_8 sie weitergegeben werden soll. Bei acht modellierten Clusterknoten (ein *YCSB*-Clusterknoten plus maximal sieben *Cassandra*-Clusterknoten) ergeben sich $8 \cdot 7 = 56$ mögliche Farben (acht mögliche Quellen und sieben mögliche Ziele pro Quelle).
2. Die Transition t_{rx} konsumiert die Netzwerkmarken, produziert jeweils neue und legt sie jeweils in eine der Stellen BP_a bis BP_g ab. Dabei wird die erwähnte Farbcodierung genutzt: die Netzwerkmarken werden je nach Quelle in die jeweils zugeordnete Stelle abgelegt. Das bedeutet, dass zum Beispiel Netzwerkmarken, die von der Stelle B_1 stammen, in die Stelle BP_a abgelegt werden.
3. Die Transition $t_{routing}$ konsumiert die Netzwerkmarken aus den Stellen BP_a bis BP_g, erzeugt sie neu und legt sie wieder jeweils in eine der Stellen BP_a bis BP_g ab. Die Zuordnung nutzt die Farbcodierung, wobei hier das Ziel der jeweiligen Netzwermarke genutzt wird. Damit wird die interne Routingentscheidung des modellierten Switches nachgestellt.
4. Die Transition t_{tx} konsumiert die Netzwerkmarken der Stellen BP_a bis BP_g, die durch den letzten Schritt erzeugt und abgelegt wurden, erzeugt sie neu und legt sie in der Stelle *Out* ab. Damit wird Downloadpfad eines Clusterknotens modelliert.

Im inneren Modell, das in Abbildung 6.8 dargestellt ist, ist aufallend, dass alle Transitionen als Transitionen ohne Zeitverzug modelliert sind. Der Grund dafür ist, dass der modellierte Switch sehr leistungsfähige technische Komponenten besitzt. Das führte dazu, dass der Empfang, der Versand sowie das Routing von Datenpaketen sehr schnell durchgeführt wird. Der zeitliche Verzug dabei ist kaum messbar und kann vernachlässigt werden.

Als inneren Modell für die in Abbildung 4.14 gezeigten Stellen B_1 bis B_8 kommt jeweils eine Instanz des erweiterten Modelles für das umgebende Betriebssystem zum Einsatz. Die Stellen B_1 bis B_8 stellen die mittlere Ebene des hierarchischen QPN-Modells zur Simulation von Experiment E dar. Nach der Beschreibung aus Textabschnitt 5.5 führen diese ebenfalls innere Modelle aus: jede Instanz des erweiterten Modells für das umgebende Betriebssystem führt auch jeweils eine eigene Instanz des erweiterten Modelles für das Datenbanksystem aus. Sie stellen die unterste Ebene des erwähnten hierarchischen QPN-Modells dar.

Die Anpassung des entstandenen hierarchischen QPN-Modells wurde in drei Teilschritten vollzogen:

1. Der Versuchsaufbau von Experiment E zeigt, dass das zugrunde Computernetzwerk einen maximalen, unidirektionalen[6] Datendurchsatz von 1 GBit pro Sekunde zwischen allen Clusterknoten zuläßt. Der Datenaustausch wird paketorientiert vorgenommen, wobei die Größe eines Datenpaketes auf maximal 1.500 Bytes festgelegt ist. Dementsprechend wurde festgelegt, dass eine Netzwerkmarke genau dieser Größe entspricht. Für den genannten Datendurchsatz von 1 GBit pro Sekunde bedeutet es, dass maximal 89.478 Netzwerkmarken pro simulierter Sekunde zwischen der Stelle *BP* und einer der Stellen B_1 bis B_8 ausgetauscht werden können, also konsumiert beziehungsweise abgelegt werden können (1 GBit $\div$ 1.500 Byte $\approx$ 89.478).
   Als Resultat wurden die Warteschlangen der Stellen BP_a bis BP_g, die in Abbildung 6.8 erkennbar sind, auf eine maximale Kapazität von 98.426 Netzwerkmarken konfiguriert. Das entspricht dem obigen maximalen Netzwerkmarkendurchsatz plus 10 Prozent Überhangspuffer (sogenannter *backlog*) laut der technischen Dokumentation des modellierten Switches.
2. Der Versuchsaufbau von Experiment E nutzte ein Bladecenter, dessen Charakteristiken in Tabelle 4.4 auf Seite 101 eingesehen werden können. Diese Tabelle zeigt, dass der Clusterknoten B1 andere technische Komponenten nutzte als die übrigen Clusterknoten B2 bis B8. Letztere besaßen eine homogene technische Ausstattung. Infolgedessen müssen die inneren Modelle der Stellen $B1$ bis $B2$ im hierarchischen QPN-Modell zur Simulation von Experiment E unterschiedlich angepasst werden. Besser gesagt, die Instanzen des erweiterten Modells für das umgebende Betriebssystem müssen an die technischen Charakteristiken des Versuchsaufbaus angepasst werden. Betrachtet man das erweiterte Modell für das umgebende Betriebssystem in Abbildung 5.11 auf Seite 154, so bedeutet es:
   - Ressourcen-Stelle p_4 besitzt ein inneres Modell, wie es in Abbildung 5.5 auf Seite 137 dargestellt ist. Es repräsentiert die beiden Festplatten, die in jedem der Clusterknoten des Versuchsaufbaus genutzt wurden. Die initiale Belegung an Massenspeicher-Ressourcenmarken unterscheidet sich nur in der Anzahl aufgrund der unterschiedlichen Speicherkapazitäten der Festplatten von Clusterknoten B1 und den übrigen B2 bis B8.
   - Ressourcenstelle p_5 führt ein inneres Modell aus, wie in Abbildung 5.4 auf Seite 135 dargestellt und in Textabschnitt 5.3.1 beschrieben ist. Auch hier unterscheidet sich die initiale Markenbelegung aufgrund der unterschiedlichen Hauptspeichergröße von Clusterknoten B1 im Vergleich zu den übrigen B2 bis B8 des Versuchsaufbaus von Experiment E.
   - Ressourcenstelle p_7 besitzt ein inneres Modell, wie es in Abbildung 5.6 auf Seite 140 dargestellt ist. Die Arbeitsweise dieses inneren Modells wurde in Textabschnitt 5.3.3

[6] Unidirektional bedeutet, dass der erreichte Datendurchsatz für den Up- und Downloadpfad zu einem Clusterknoten in der Summe nicht mehr als 1 GBit/s übersteigen kann.

erläutert. Damit repräsentiert dieses innere Modell die CPUs, die im Versuchsaufbau von Experiment E bei den Clusterknoten genutzt wurden. Alle Clusterknoten besaßen zwei CPUs mit jeweils zwei CPU-Kernen, also insgesamt 4 CPU-Kerne. Die initiale Markenbelegung der entsprechenden Ressourcenstellen unterscheidet sich nur dadurch, dass unterschiedliche nominale Taktfrequenzen vorlagen (Clusterknoten B_1 2.8 GHz pro Kern gegenüber 2 GHz bei den übrigen Clusterknoten B_2 bis B_8).

Die initialen Markenbelegungen der Ressourcenstellen ist in Tabelle 6.10 dargestellt. Bei der Berechnung der Markenanzahlen wurde dasselbe Schema verwendet, dass auch bei der Simulation der Experimente B und C verwendet wurde, um die verwendete technische Plattform in Form von Ressourcenmarken zu modellieren. Dieses Schema ist detailliert in Textabschnitt 6.2.1 ab Seite 167 beschrieben.

	← Stelle B_1 →		← Stelle B_2 bis B_8 →	
Ressourcenstelle	**Markenanzahl**	**Entsprechung**	**Markenanzahl**	**Entsprechung**
Massenspeicher und Massenspeichermarken				
p_{4a}, p_{4b}	78.643.200	300 GByte	38.273.024	146 GByte
$p_4 = p_{4a} + p_{4b}$	157.286.400	600 GByte	76.546.048	292 GByte
Haupt- und Auslagerungsspeicher, Arbeitsspeichermarken				
p_{5a}	4.194.304	32 GByte	2.097.152	16 GByte
p_{5b}	1.073.741.824	8 GByte	1.073.741.824	8 GByte
$p_5 = p_{5a} + p_{5b}$	1.077.936.128	40 GByte	1.075.838.976	24 GByte
Prozess-IDs, Prozessoren und Prozessormarken				
p_6	65.495	Prozess-IDs	65.495	Prozess-IDs
p_{7a}, p_{7b}, p_{7c}, p_{7d}	2.800	Prozessor-Kern	2.000	Prozessor-Kern
$p_7 = p_{7a} + p_{7b} + p_{7c} + p_{7d}$	11.200	Prozessor	8.000	Prozessor

Tabelle 6.10: Initiale Ressourcenmarkenbelegung der Stellen im hierarchischen QPN-Modell zur Simulation von Experiment E

3. Die Prozessstellen p_3 in den Instanzen des erweiterten Modelles für das umgebende Betriebssystem führen als inneres Modell jeweils eine Instanz des erweiterten Modelles für das Datenbanksystem aus.
   Da Experiment E das verteilte Datenbanksystem *Cassandra* evaluierte, müssen die letztgenannten Instanzen an die Architektur von *Cassandra* angepasst werden. Dazu betrachte man das erweiterte Modell des Datenbanksystems, wie es in Abbildung 5.12 auf Seite 156 dargestellt ist. Aus der Architekturbeschreibung von *Cassandra* geht hervor, dass für Lese- und Schreiboperationen auf die gespeicherten Daten massiv auf Puffer zurückgegriffen wird [Hew11, Dat]. Auch die experimentellen Ergebnisse von Experiment E in Textabschnitt 4.2.1 bestätigen dies. Um dieses Verhalten zu modellieren, wurde in Stelle p_{11} ein inneres Modell erstellt, das in Abbildung 6.9 auf Seite 187 dargestellt ist.
   Das in Abbildung 6.9 gezeigte innere Modell besteht im wesentlichen aus nur aus zwei Stellen (p_{11a}, p_{11b}) und drei Transitionen (t_{11a} bis t_{11c}) sowie den formal notwendigen Stellen *In* und *Out*. Der primäre Zweck von Stelle p_{11} gemäß der Beschreibung aus Textabschnitt 5.7 ist es, das Speichern von Daten innerhalb des verteilten Datenbanksystems zu modellieren. Dazu sei angemerkt, dass *YCSB* keine komplexen Datenstrukturen vorschreibt. Im Gegenteil, es wird nur eine Tabelle erstellt und genutzt. Aus diesem Grund ist es nicht notwendig, im inneren Modell von Stelle p_{11} mehreren Stellen zu modellieren, die jeweils

eine Datenbankstruktur in Form von einer Tabelle repräsentiert. Stattdessen werden nur die zwei Stellen p_{11a} und p_{11b} genutzt. Erstere repräsentiert gespeicherte Daten, die im Cache von *Cassandra* gespeichert sind und letztere diejenigen Daten, die persistent auf dem Massenspeicher gespeichert wurden [AS95].

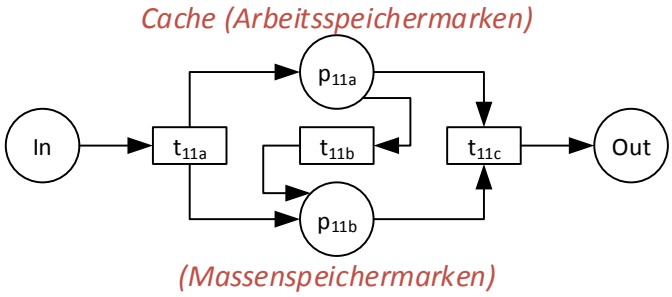


Abbildung 6.9: Inneres Modell für Stelle p_7 im erweiterten Modell für das Datenbanksystem zur Simulation von Experiment E

Cassandra's Cachen von Datenbankinhalten wurde anhand folgender Arbeitsweise des inneren Modells modelliert, das in Abbildung 6.9 dargestellt ist:

- Datenmarken, die gespeichert werden sollen, erreichen das innere Modell, indem sie in Stelle *In* abgelegt werden. Transition t_{11a} konsumiert sie. Diese Transition enthält eine Liste mit zwei Transitionsregeln. Die erste, höher priorisierte Regel allokiert zusätzlich Arbeitsspeichermarken, sofern noch welche verfügbar sind. Die Assoziation zwischen Datenmarke und Arbeitsspeichermarken wird dann in Stelle p_{11a} gespeichert, indem eine Assoziativmarke erzeugt und in dieser Stelle abgelegt wird. Sofern keine Arbeitsspeichermarken verfügbar sind, wird die zweite Transitionsregel ausgeführt. Sie allokiert anstatt Arbeitsspeicher- Massenspeichermarken und erzeugt eine Assoziativmarke für Stelle p_{11b}, in der diese dann auch abgelegt wird[7].
  Insgesamt wird dadurch modelliert, dass zu speichernde Daten zunächst in einem Cache gespeichert werden, der im Hauptspeicher angelegt ist. Datenmarken werden mit konsumierten Arbeitsspeichermarken assoziiert, womit nachgestellt werden soll, dass deren Speicherung im Cache Arbeitsspeicher benötigt.
- Transition t_{11b} enthält nur eine Transitionsregel. Sie besagt, dass wenn eine gewisse Anzahl an Assoziativmarken in Stelle p_{11a} gespeichert sind, diese konsumiert werden sollen. Dabei werden auch die ursprünglichen, assoziierten Arbeitsspeichermarken wieder freigegeben. Für die ursprünglichen, assoziierten Datenmarken werden Massenspeichermarken allokiert, die miteinander assoziiert werden. Die dabei erzeugten Assoziativmarken werden in Stelle p_{11b} abgelegt.
  Insgesamt wird durch diesen Schritt das Verhalten von *Cassandra* modelliert, den Inhalt des Caches zu leeren und die enthaltenen Datenbankinhalte auf den Massenspeicher zu persistieren.

Nach der Beschreibung des erweiterten Modelles für das Datenbanksystem in Textabschnitt 5.7 repräsentiert Stelle mit innerem Modell p_{12} die Indexstrukturen, die innerhalb des modellierten verteilten Datenbanksystems auftreten können. Die Indexstrukturen werden von

[7] Dieser Fall trat während sämtlicher Simulationen nie auf, die im Rahmen dieser Dissertation durchgeführt wurden. Prinzipiell kann diese Transitionsregel sowie die Kante zwischen der Tansition t_{11a} und Stelle p_{11a} in Abbildung 6.9 auch vernachlässigt werden.

der Datenbankanwendung vorgegeben, die simuliert werden soll. Im Falle der Simulation von Experiment E ist es dies der *YCSB*. Wie bereits erwähnt, sieht der *YCSB* keine komplexe Datenbankstrukturen vor, wobei insbesondere keinerlei Indexstrukturen angelegt werden. In Hinblick auf die Stelle p_{12} bedeutet es, dass kein inneres Modell notwendig ist, da keine Indexstrukturen abgebildet werden müssen. Diese Stelle ist zwar formal vorhanden, wird aber für die Simulation von Experiment E nicht genutzt.

Simulationsdurchführung Nach der Anpassung der QPN-Modelle wurden Simulationen von Experiment E vorgenommen. Die einzelnen Simulationen orientierten sich dabei an den Testserien, wie sie auch zur Gewinnung der experimentellen Ergebnissen genutzt wurden. Eine Testreihe umfasste dabei die vier Arbeitslasten `load`, `read`, `write` und `mixed`, wobei zwei Verteilungen unterschieden wurden (diskrete Gleichverteilung und *Zipf*-Verteilung). Für die beiden Szenarien A und B von Experiment E wurde jeweils eine Testreihe für einen *Cassandra*-Cluster mit ansteigender Clusterknotenanzahl durchgeführt. In der Summe mussten 20 Testserien simuliert werden (2 Verteilungen $\times$ 2 Szenarien $\times$ 5 Clustergrößen). Da jede Testserie aus vier Arbeitslasten besteht, die einzeln simuliert werden müssen, wurden insgesamt 240 einzelne Simulationen durchgeführt (20 Testserien $\times$ 4 Arbeitslasten $\times$ 3 Wiederholungen).

Die Simulation einer Testserie für eines der beiden Szenarien, eine der beiden Verteilungen und einer bestimmten Anzahl von *Cassandra*-Clusterknoten (beginnend mit 3) geschah immer nach dem gleichem Schema:

1. Für Szenario A wurde die Prozesstelle p_1 im inneren Modell von Stelle B_1 des hierarchischen QPN-Modells mit 350 Millionen Prozessmarken befüllt. Dieses innere Modell führt eine Instanz des erweiterten Modelles für das umgebende Betriebssystems aus. Jede dieser Prozessmarken enthält genau eine Datenmarke, womit 350 Millionen Datenmarken entstanden, die es zu verarbeiten galt. Das entspricht den 350 Millionen Einfügeoperationen der Arbeitslast `load` für Szenario A.
   Für Szenario B wurde die oben genannte Prozessstelle p_1 mit einer Anzahl von Prozessmarken gefüllt, die dem Vielfachen von 50 Millionen und der gegebenen Anzahl an simulierten *Cassandra*-Clusterknoten entspricht. Die genaue Anzahl an Prozessmarken kann in Tabelle 4.3(b) auf Seite 100 in der Zeile abgelesen werden, die mit `load` bezeichnet ist.
   Die Warteschlange der Prozessstelle p_2 ist darauf konfiguriert, dass maximal 1.500 Prozessmarken von Prozessstelle p_1 konsumiert werden können. Damit wird das konfigurierte Verhalten von Experiment E nachgestellt, dass der *YCSB*-Client die gestellten Arbeitslasten mit 1.500 Betriebssystem-Threads gleichzeitig ausführt.
   Für beide Szenarien wurden die Transitionsregelsätze für die Transitionen t_5 und t_7 in der Instanz des erweiterten Modell für das Datenbanksystem angepasst. Diese Instanz wird als inneres Modell in Prozessstelle p_3 ausgeführt. Die Transitionsregelsätze dienen dazu, die Datenmarken in Netzwerkmarken zu transformieren. Dabei wird die gegebene Verteilung berücksichtigt. Die Netzwerkmarken erhalten eine Farbe, die der weiter oben genannten Farbcodierung entspricht. Das bedeutet, dass die Netzwerken eine Farbe bekamen, die aussagt, dass die Quelle der Netzwerkmarke die Stelle B_1 und das Ziel eine der Stellen B_2 bis B_8 ist. Das Ziel wird anhand der gegegeben Verteilung festgelegt.
2. Die Simulation der Arbeitslast `load` wurde gestartet. Damit werden alle Datenmarken, die in den Prozessmarken enthalten sind, in Netzwerkmarken transformiert. Die Netzwerkmarken werden dann zu den jeweiligen Zielstellen $B2$ bis B_8 “transportiert”. Dort werden sie wieder in Datenmarken zurücktransformiert und lokal verarbeitet. Genauer gesagt, die einzelnen Instanzen des erweiterten Modelles für das Datenbanksystems, dass für die *Cassan-*

dra-Architektur angepasst wurde, verarbeiten die Datenmarken anhand der Arbeitsweise, die in Textabschnitt 5.7.1 ab Seite 155 beschrieben ist.
Insgesamt wird durch die Simulation der Arbeitslast `load` erreicht, dass die Daten des *YCSB*-Datengenerators in Form von Datenmarken im hierarchischen QPN-Modell verteilt und gespeichert werden. Dadurch entsteht eine Datenbasis, die für die Simulation der nachfolgenden Arbeitslasten vorausgesetzt wird.

3. Es werden nacheinander die Arbeitslasten `read`, `write` und `mixed` simuliert. Dafür wird für jede dieser Simulationen die im ersten Schritt genannte Stelle p_1 mit 10 Millionen Prozessmarken befüllt. Jede Prozessmarke enthält eine Datenmarke. Die Datenmarken stellen nur eine Auswahl derjenigen dar, die bereits im ersten Schritt verwendet wurden. Die Anzahl der Prozessmarken kann direkt in den beiden Tabellen 4.3(a) für Szenario A und 4.3(b) für Szenario B auf Seite 100 abgelesen werden.

Die Prozessmarken, in denen die Datenmarken enthalten sind, können nach diesem Simulationsschema als Eingabemenge für das hierarchische QPN-Modell angesehen werden.

Zur Simulation der Testserien wurde dieselbe Simulationssoftware und derselbe Simulationsserver verwendet, die auch zur Simulation von Experiment B und C zum Einsatz kamen, deren Simulationsergebnisse im vorherigen Textabschnitt 4.1 beschrieben sind. Das bedeutet, dass die QPN-Simulationssoftware *QPME* in Version 2 zur Erstellung des hierarchischen QPN-Modells und dessen Verwendung zur Simulation von Experiment E genutzt wurde. Die technischen Charakteristiken des Simulationsservers sind in Tabelle 6.6 auf Seite 173 aufgelistet.

Bei allen durchgeführten Simulationen der Testserien von Experiment E wurden dieselben stochastischen Werte des hierarchischen QPN-Modells inklusive alle ausgeführten Instanzen der erweiterten Modelle des umgebenden Betriebssystems und des Datenbanksystems aufgezeichnet. Analog zu den Simulationen von Experiment B und C aus dem vorherigen Textabschnitt 4.1 sind das zum Beispiel die minimale und maximale Markenbelegung aller Stellen und der mittlere Markendurchsatz der Stellen. Bei allen durchgeführten Simulationen wurde als simulierte Zeiteinheit 1 Sekunde verwendet. Damit konnte für alle Simulationen eine simulierte Zeit aufgezeichnet werden, bis der Datenfluss aufgrund der Eingabemenge terminiert. Diese simulierte Zeit entspricht der Antwortzeit einer Arbeitslast, wie sie experimentell im Rahmen von Experiment E gemessen wurde. Sie kann somit für einen direkten Vergleich genutzt werden. Darüber hinaus wurde auch die Simulationsdauer aufgezeichnet, also die Zeitdauer bis eine Simulation an sich gebraucht hat, um zu terminieren.

Simulationsergebnisse In gleicher Weise wie bei den Simulationen der Experimente B und C im vorhergehenden Textabschnitt 4.1 erlauben die Simulationsergebnisse die Berechnung der Genauigkeit in Hinblick auf die Performance in Form der Antwortzeiten und dem Energieverbrauch. Gemäß Definition 13 für die Genauigkeit auf Seite 165 gibt sie den prozentualen Unterschied zwischen der experimentell ermittelten und simulierten Antwortzeit einer Arbeitslast beziehungsweise eines Energieverbrauchswertes an. Die errechneten Genauigkeitswerte können dann als eine Art “Qualitätsindikator” für das hierarchische QPN-Modell zur Simulation von Experiment E herangezogen werden. Im weiteren Sinne können sie auch dafür genutzt werden, die “Güte” der angepassten QPN-Modelle zu bemessen. Das betrifft also alle QPN-Modelle, die in den Textabschnitten 5.5, 5.6 und 5.7 eingeführt und beschrieben worden.

Bei der Berechnung der Genauigkeit im Bezug auf die Antwortzeiten wurden zunächst die Durchschnittswerte aller drei Wiederholungen einer Testreihe errechnet. Das bedeutet, dass pro simulierter Testreihe Durchschnittswerte der simulierten Antwortzeit der vier Arbeitslasten `load`,

`read`, `write` und `mixed` berechnet wurden. Diese Werte können dann direkt mit denjenigen verglichen werden, die experimentell ermittelt und in Textabschnitt 4.2.1 beschrieben wurden.

In Tabelle 6.11 sind die durchschnittlichen Genauigkeiten der vier genannten Arbeitslasten für die simulierten *Cassandra*-Clustergrößen dargestellt. Bei den dargestellten Genauigkeitsangaben wird bei den Arbeitslasten `read`, `write` und `mixed` zusätzlich nach der Verteilung unterschieden. Zudem ist Tabelle 6.11 in zwei Teiltabellen unterteilt, die die Genauigkeitswerte jeweils für ein Szenario zeigen (Teiltabelle 6.11(a) für Szenario A beziehungsweise 6.11(b) für Szenario B).

	← *Cassandra*-Clustergröße →				
Arbeitslast	**3**	**4**	**5**	**6**	**7**
`load`	79.49	81.35	82.01	83.71	84.13
Diskrete Gleichverteilung					
`read`	77.33	78.87	80.47	82.85	83.28
`write`	75.17	76.55	79.02	81.61	83.83
`mixed`	76.23	77.48	77.97	79.11	79.84
Zipf-Verteilung					
`read`	78.67	79.53	81.41	83.24	84.75
`write`	77.11	78.36	79.79	81.85	82.64
`mixed`	77.28	77.96	78.56	79.44	80.42

(a) Szenario A (feste Datenmenge)

	← *Cassandra*-Clustergröße →				
Arbeitslast	**3**	**4**	**5**	**6**	**7**
`load`	77.33	78.83	79.56	81.87	83.61
Diskrete Gleichverteilung					
`read`	76.57	78.05	79.92	81.46	82.79
`write`	76.19	77.88	79.69	81.49	82.85
`mixed`	78.36	79.03	80.17	81.18	82.61
Zipf-Verteilung					
`read`	78.13	79.26	79.96	80.44	81.29
`write`	78.26	79.44	80.05	80.62	81.51
`mixed`	79.21	79.47	79.82	80.17	80.63

(b) Szenario B (ansteigende Datenmenge)

Tabelle 6.11: Genauigkeit der Antwortzeiten für die Simulation von Experiment E in Prozent

Vergleicht man die Genauigkeitswerte in beiden Teiltabellen 6.11(a) und 6.11(b), so fällt auf, dass sie mit steigender *Cassandra*-Clustergröße ansteigen. Zusätzlich ist zu erkennen, dass sich die Genauigkeitswerte hinsichtlich der Verteilung unterscheiden. Dazu betrachtet man nur die Genauigkeitswerte in den beiden Teiltabellen, die mit `read`, `write` oder `mixed` bezeichnet sind. Diejenigen, bei der eine *Zipf*-Verteilung simuliert wurde, fallen höher aus als diejenigen mit einer simulierten diskreten Gleichverteilung.

Vergleicht man zudem die Genauigkeitswerte für diese drei Arbeistlasten in Teiltabelle 6.11(a) mit denjenigen aus Teiltabelle 6.11(b), so ist feststellbar, dass letztere eine kleinere Spannbreite haben. Das bedeutet, dass die Differenz zwischen dem minimalen und maximalen Genauigkeitswert geringer ausfällt.

Dasselbe Verhalten kann auch bei den experimentellen Messergebnissen von Experiment E in Textabschnitt 4.2.1 beobachtet werden. Sie zeigen, dass es bei allen Arbeitslasten mit *Zipf*-Verteilung zu einer geringeren Belastung des Clusternetzwerkes kam. Das heisst, es wurden weniger Datenpakete zwischen den Clusterknoten ausgetauscht. Dieser Fakt konnte auch bei den Simulationen von Experiment E nach Analyse der stochastischen Werte von Stelle *BP* im verwendeten hierarchischen QPN-Modell bestätigt werden (siehe Abbildung 6.7 auf Seite 183). Die Analyse zeigt, dass die mittlere Population an Netzwerkmarken bei simulierten Arbeitslasten mit *Zipf*-Verteilung um 6.7 Prozent geringer ist als bei denjenigen mit simulierter diskreter Gleichverteilung.

In den beiden Teiltabellen 6.11(a) und 6.11(b) ist auch erkennbar, dass es bei den Genauigkeitswerten keine statistischen Ausreißer gibt. Das ist ein Indikator dafür, dass das hierarchische QPN-Modell zur Simulation von Experiment E prinzipiell dazu geeignet ist, die experimentellen Ergebnisse nachzustellen. Das äußert sich zum Beispiel darin, dass die Analyse der Simulationsergebnisse dieselben Ergebnisse liefert, wie sie bereits in Textabschnitt 4.2.1 beschrieben sind.

Errechnet man die mittlere Genauigkeit der Antwortzeiten aller simulierten Arbeitslasten, so ergibt sich ein Wert von 79.93 Prozent. Das bedeutet, dass im Durchschnitt die simulierten Antwortzeit um 20.07 Prozent von der experimentellen abweicht.

Bezüglich der Genauigkeit des Energieverbrauches wurde dieselbe Berechnunsgrundlage verwendet, wie sie schon im vorherigen Textabschnitt 4.1 bei der Simulation von Experiment B und C verwendet und beschrieben wurde. Das bedeutet, dass der simulierte Energieverbrauch ein errechneter Wert ist, der sich als Produkt aus einem Energieverbrauchswert pro allokierter und deallokierter Ressourcenmarke, dem mittleren Markendurchsatz der jeweiligen Ressourcenstelle und der simulierten Antwortzeit ergibt. Hierbei muss aber beachtet werden, dass bei der Simulation von Experiment E verschiedene *Cassandra*-Clustergrößen und damit verschiedene Anzahlen von Instanzen des erweiterten Modells für das umgebende Betriebssystem verwendet wurden. In diesen Instanzen sind die Ressourcenstellen p_4, p_5 und p_7 (Massen- und Arbeitsspeicher, Prozessor), die von Interesse für die Berechnung sind. Für alle drei wird das obige Produkt gebildet und die Produkte summiert. So ergibt sich ein simulierter Gesamtenergieverbrauch pro Instanz. Das entspricht dem simulierten Gesamtenergieverbrauch für einen simulierten Clusterknoten in einer Simulation. Summiert man die simulierten Gesamtenergieverbräuche aller simulierten Clusterknoten, so bezeichnet dies den simulierten Gesamtenergieverbrauch des gesamten simulierten Clusters.

Dieser simulierte Gesamtenergieverbrauch des gesamten Clusters ermöglicht einen direkten Vergleich mit den realen, experimentell erhobenen Energieverbrauchswerten von Experiment E. Laut Definition 13 der Genauigkeit wird der Unterschied prozentual angegeben. Analog zu Tabelle 6.11 mit den Genauigkeitsangaben zu den Antwortzeiten sind in Tabelle 6.12 die Genauigkeitswerte für den Gesamtenergieverbrauch dargestellt. Dabei hat Tabelle 6.12 denselben Aufbau wie Tabelle 6.11. Das bedeutet, die Genauigkeitswerte werden in zwei Teiltabellen 6.12(a) und 6.12(b) als Prozentangabe dargestellt, wobei jede Teiltabelle eines der simulierten Szenarios A beziehungsweise B zeigt. Die in den beiden Teiltabellen gezeigten Genauigkeitswerte entsprechen den jeweiligen Durchschnittswerten der drei Wiederholungen einer simulierten Testreihe von Experiment E.

	← *Cassandra*-Clustergröße →				
Arbeitslast	**3**	**4**	**5**	**6**	**7**
`load`	61.53	59.76	55.27	51.97	48.56
Diskrete Gleichverteilung					
`read`	54.82	53.22	51.09	48.34	45.13
`write`	51.64	49.93	46.25	43.61	41.83
`mixed`	53.12	51.59	48.89	47.38	45.15
Zipf-Verteilung					
`read`	55.82	54.31	52.12	50.42	47.23
`write`	58.41	56.86	53.63	51.10	49.18
`mixed`	59.34	57.44	55.02	52.97	50.65

(a) Szenario A (feste Datenmenge)

	← *Cassandra*-Clustergröße →				
Arbeitslast	**3**	**4**	**5**	**6**	**7**
`load`	63.78	61.24	58.61	55.72	53.69
Diskrete Gleichverteilung					
`read`	57.81	56.26	55.11	53.98	52.35
`write`	55.48	54.36	53.06	51.25	49.84
`mixed`	58.66	57.49	56.27	55.04	53.13
Zipf-Verteilung					
`read`	59.89	56.31	54.91	52.54	50.65
`write`	60.13	58.41	56.82	54.37	53.21
`mixed`	59.71	57.87	55.21	53.85	52.14

(b) Szenario B (ansteigende Datenmenge)

Tabelle 6.12: Genauigkeit des Energieverbrauches für die Simulation von Experiment E in Prozent

Vergleicht man die Genauigkeitswerte für die Antwortzeit aus Tabelle 6.11 mit denjenigen für den Gesamtenergieverbrauch in Tabelle 6.12, so ist auffallend, dass letztere signifikant geringer ausfallen. Vergleicht man die mittlere Genauigkeit der Antwortzeit über alle durchgeführten Simulationen von 79.93 Prozent mit der des Gesamtenergieverbrauches von 53.75 Prozent, so

ist das ein Unterschied von nahezu 26 Prozent. Das bedeutet gleichzeitig, dass die realen, experimentell gewonnenen Gesamtenergieverbrauchswerte von Experiment E um fast die Hälfte (46.25 Prozent) von denjenigen abweichen, die aus den Simulationsergebnissen errechnet wurden.

Der Vergleich der Genauigkeitswerte aus Tabelle 6.11 mit denen aus Tabelle 6.12 zeigt zudem, dass letztere im Gegensatz zu ersteren mit zunehmender *Cassandra*-Clustergröße sinken. Eine Analyse der aufgezeichneten stochastischen Werte der durchgeführten Simulationen zeigt, dass die Ursache für dieses Verhalten und den insgesamt niedrigen Genauigkeitswerten für den Gesamtenergieverbrauch nicht im Datenfluss der Marken zu finden ist. Mit anderen Worten heisst das, dass die Simulation anhand des hierarchischen QPN-Modells und der darin ausgeführten Instanzen der QPN-Modelle nicht das ursächliche Problem ist. Das Problem ist die Berechnungsbasis, anhand dessen der simulierte Gesamtenergieverbrauch errechnet wird. Im wesentlichen betrifft das den festgelegten Energieverbrauchswert pro allokierter und deallokierter Ressourcenmarke (Hauptspeicher-, Massenspeicher- und Prozessorenmarke).

Im vorherigen Textabschnitt 4.1 wurde beschrieben, dass sich diese Basiswerte grundsätzlich von den maximalen Markendurchsätzen der Ressourcenstellen in Verbindung mit experimentell gewonnenen Energieverbrauchswerten errechnen lassen. Im Falle der Simulationen von Experiment B und C lagen dafür auch sehr präzise experimentell erhobene Energieverbrauchswerte vor. Der Grund dafür war, dass der Versuchsbau es zuließ, den Energieverbrauch der einzelnen technischen Komponenten zu exakt messen. Dieses Vorgehen war für die Simulation von Experiment E nicht durchführbar. Das verwendete Bladecenter, dessen Charakteristiken in Tabelle 4.4 auf Seite 101 dargestellt sind, besitzen propritäre, nicht dokumentierte Stromschienen innerhalb des Bladecenters und auch in den einzelnen Blades. Daher konnte der Versuchsaufbau nicht dahingehend modifiziert werden, wie es bei den Experimenten B und C der Fall war. Stattdessen musste auf Referenzwerte für den Energieverbrauch zurückgegriffen werden, wie sie in den technischen Datenblättern für das Bladecenter sowie für die verbauten technischen Komponenten zu finden sind. Als Folge dessen enstanden Basiswerte, die weniger präzise waren. Das führte zu den oben beschriebenen Genauigkeitswerten für den Gesamtenergieverbrauch der Simulation von Experiment E.

Zusätzliche Simulationsergebnisse Neben der Evaluation der Genauigkeiten hinsichtlich der Antwortzeiten und des Energieverbrauches wurden auch andere Aspekte im Bezug auf Experiment E betrachtet. Vergleicht man die Simulationen der Experimente B und C mit Experiment E, dann ist der Unterschied in den Komplexitätsgraden auffallend. Darunter ist zu verstehen, dass der betriebenen Aufwand für die Simulationen stark unterschiedlich ist. Für die Simulation der Experimente B und C waren zwar nur zwei QPN-Modelle und mehrere innere Modelle notwendig, aber die Anpassungen zur Simulation des *TPC-H*-Benchmarks (Experiment B) und des *StarSchema*-Benchmarks (Experiment C) erforderten die Nutzung einer sehr großen Anzahl von Markentypen und -farben und einer bedeutenden Menge an Transitionsregeln zur Verarbeitung dieser Marken. Dieser Umstand resultiert in einem großen Komplexitätsgrad.

Im Vergleich dazu ist der Komplexitätsgrad der Simulation von Experiment E vergleichsweise gering. Auch wenn ein hierarchisches QPN-Modell mit multiplen Instanzen genutzt wurde, fällt die Anpassung moderat aus. Das liegt im wesentlichen an der recht simplen Struktur des simulierten *Yahoo Cloud Serving Benchmarks*, da weder komplexe Datenbankstrukturen noch Datenbankabfragen modelliert werden mussten. Dadurch fielen die Multimengen an Marken und Markenfarben recht klein aus, was wiederum ein positiven Effekt auf die Anzahl der Transitionsregeln hatte.

Dieser vergleichsweise geringe Komplexitätsgrad hat auch Auswirkungen auf die Simulation selber. So war es möglich, dass eine Simulation nach dem weiter oben gegebenen Simulationsschema im Durchschnitt bereits nach 3 Minuten und 46 Sekunden beendet war. Das bedeutet, dass sie erfolgreich terminierte. Vergleicht man zum Beispiel die experimentelle Evaluation der Antwortzeit mit der simulierten, ist das um annähernd Faktor 8.000 schneller. Nicht inbegriffen sind dabei die Vorarbeiten, zum Beispiel das Einrichten des experimentellen Versuchsaufbaus oder die Anpassung der QPN-Modelle.

Insgesamt ermöglichte die recht kurze Simulationsdauer auch Simulationen durchzuführen, die nicht mehr im Rahmen von Experiment E lagen. Das bedeutet, dass durch recht einfache Anpassungen *Cassandra*-Cluster simuliert wurden, die mehr als sieben Clusterknoten umfassten. Das wurde dadurch erreicht, dass das hierarchische QPN-Modell, wie es in Abbildung 6.7 auf Seite 183 dargestellt ist, um mehrere Stellen und darin enthaltenene Instanzen erweitert wurde. In der Analyse der experimentellen Ergebnisse von Experiment E in Textabschnitt 4.2.1 wurde durch lineare Regression berechnet, dass hinsichtlich der Energieeffizienz die optimale *Cassandra*-Clustergröße bei 10 Clusterknoten liegt. Durch die erwähnten zusätzlichen Simulationen konnte diese Größe bestätigt werden.

Zudem konnte durch zusätzliche Simulationen auch der simulierte Netzwerkverkehr betrachtet werden. Wie in Szenario B von Experiment E beschrieben, bedeutet das, dass die Größe des simulierten *Cassandra*-Clusters schrittweise erhöht wurde, wobei auch die Größe der Datenbasis anwuchs. So konnte beobachtet werden, dass der simulierte Netzwerkverkehr im simulierten *Cassandra*-Cluster stetig, aber nicht linear zunahm. Das hat sich darin ausgedrückt, dass Netzwerkmarken im steigenden Maße nicht mehr innerhalb einer simulierten Sekunde verarbeitet wurden, sondern im Wartebereich der Warteschlangen-Stellen über mehrere simulierte Sekunden verblieben. Es konnte auch beobachtet werden, dass der Wartebereich der Warteschlangen-Stellen, die für die Modellierung des Computernetzwerkes des *Cassandra*-Clusters zuständig waren, über mehrere simulierte Sekunden komplett besetzt waren, sodass der Datenfluss der Netzwerkmarken komplett aussetzte. Diese Effekte hatten als Resultat auch Auswirkungen auf die Verarbeitung der Datenmarken, die in den Netzwerkmarken als Submarken enthalten waren. Sie erreichten nur verzögert die entsprechenden Zielstellen, wodurch insgesamt der gesamte Datenfluss der Datenmarken verzögert wurde. Daraus folgte, dass die simulierte Antwortzeit ingesamt anstieg.

So zeigte sich bei einer simulierten *Cassandra*-Clustergröße von 25 Clusterknoten, dass der simulierte Netzwerkverkehr derart immens anwuchs, dass es einen signifikanten Einfluss auf die simulierte Performance in Form der Antwortzeit hatte. Bei dieser simulierten Clustergröße ergab sich ein Wendepunkt. Das heisst, dass die Antwortzeiten mit jedem zusätzlich hinzugefügten, simulierten Clusterknoten sank. Es wurde bis zu einer *Cassandra*-Clustergröße von 30 Clusterknoten simuliert, wobei sich zeigte, dass die simulierte Antwortzeiten im Durchschnitt um 2.6 Prozent mit jedem zusätzlichen, simulierten Clusterknoten sanken.

6.3.2 Simulation von Experiment F (*Cassandra* mit 13 Clusterknoten und *YCSB*)

Die Simulation von Experiment F hat starke Parallelen zu der Simulation von Experiment E, wie sie in vorhergehenden Textabschnitt 6.3.1 beschrieben ist. Der Grund dafür ist, dass die beiden Experimente stark miteinander zusammenhängen. So wurde Experiment F durchgeführt, um die Annahme von Experiment E experimentell zu bestätigen, dass es hinsichtlich der Energieeffizienz eine optimale *Cassandra*-Clustergröße gibt. Dieses Ziel wird auch durch die Beschreibungen beider Experimente auf den Seiten 98 und 107 deutlich.

Modellanpassungen Wie eingangs erwähnt, basiert die Simulation von Experiment F in weiten Teilen auf der Simulation von Experiment E. Um textuelle Redundanzen zu vermeiden, wird in diesem Textabschnitt nur auf die Unterschiede eingegangen. Das bedeutet, dass grundsätzlich wieder ein hierarchisches QPN-Modell für die Simulation genutzt wurde, dass aus drei Hierarchieebenen besteht. Auf der untersten und mittleren Ebene wurden keine Änderungen hinsichtlich der verwendeten Modelle vorgenommen. Das bedeutet es wurden weiterhin Instanzen des erweiterten Modelles für das umgebende Betriebssystem und des Datenbanksystem verwendet, deren Anpassungen im vorhergehenden Textabschnitt beschrieben wurden. Da sich aber der Versuchsaufbau von Experiment F gegenüber Experiment E geändert hat, wurde auf oberste Ebene des hierarchischen QPN-Modelles ein neues QPN-Modell entworfen und genutzt. Dieses Modell repräsentiert den modifzierten Versuchsaufbau von Experiment F und ist in Abbildung 6.10 auf Seite 195 dargestellt.

Das QPN-Modell, das in Abbildung 6.10 dargestellt, zeigt die erforderlichen Stellen und Transitionen, um das zugrunde liegende Computernetzwerk zu modellieren, wie es der Versuchsaufbau von F vorgab. Schematisch ist der Versuchsaufbau in Abbildung 4.17 auf Seite 109 dargestellt und in Textabschnitt 4.2.2 beschrieben. Prinzipiell bestand der Versuchsaufbau aus zwei Bladecentern, die über einen Switch miteinander verbunden waren. Jedes Bladecenter verfügte dabei über eine Anzahl von Knoten, wordurch insgesamt 14 Knoten zur Verfügung standen. In Abbildung 6.10 sind diese Knoten als Stellen mit inneren Modell B_1 bis B_{14} repräsentiert.

Die Knoten eines Bladecenters konnten über die interne Backplanes Daten austauschen. Dies ist in Abbildung 6.10 durch die beiden Warteschlangen-Stellen BP_1 und BP_2 repräsentiert. Vergleicht man das QPN-Modell in Abbildung 6.10 mit dem in Abbildung 6.7 auf Seite 183, so erkennt man, dass letzteres in erstem enthalten ist[8].

Die Stelle mit innerem Modell *Sw* in Abbildung 6.10 verbindet die beiden Stellen BP_1 und BP_2 miteinander, wodurch Netzwerkmarken von den Stellen B_1 bis B_8 zu den Stellen B_9 bis B_{14} und umgekehrt gelangen können. Alle Transitionen, die in Abbildung 6.10 dargestellt sind, dienen dazu, Netzwerkmarken zwischen den genannten Stellen zu konsumieren, neu zu erzeugen und in die nächstfolgende Stelle abzulegen. Dabei wird die Richtung des Markenflusses beachtet, womit ein Up- und Downloadpfad modelliert wird. Insgesamt dient das QPN-Modell in Abbildung 6.10 also dazu, den Austausch von Daten zwischen den modellierten Clusterknoten des Versuchsaufbaus von Experiment F in Form eines Datenflusses von Netzwerkmarken nachzustellen.

In Abbildung 6.10 ist auch zu erkennen, dass die Stellen BP_1, BP_2 und *Sw* jeweils ein inneres Modell besitzen. Diese inneren Modellen folgen generell demselben Aufbau wie in Abbildung 6.8 auf Seite 183 dargestellt. In Abbildung 6.11 auf Seite 196 sind alle drei inneren Modelle

[8] Auf der rechten Seite von Abbildung 6.10 sind die Stellen mit innerem Modell B_1 bis B_8 zu erkennen, die mit der Stelle mit innerem Modell BP_1 über Transitionen verbunden sind. Diese Stellen sind auch auf Abbildung 6.7 auf Seite 183 zu sehen, allerdings sind sie horizontal angeordnet.

dargestellt. Sie unterscheiden sich prinzipiell nur in der Anzahl der Stellen mit Warteschlange. Das bedeutet, dass die Anzahl dieser Stellen im Falle von BP_1 und BP_2 (Teilabbildungen 6.11(a) und 6.11(c)) mit der Anzahl der modellierten Clusterknoten (Stellen B_1 bis B_{14} in Abbildung 6.10) korrespondiert. Das innere Modell von Stelle *Sw* enthält dagegen immer zwei Stellen mit Warteschlange, die in Teilabbildung 6.11(b) mit Sw_a und Sw_b bezeichnet sind. Diese beiden Stellen repräsentieren die Verbindung zwischen der Stelle *Sw* und BP_1 sowie *Sw* und BP_2, die in Abbildung 6.7 zu erkennen ist.

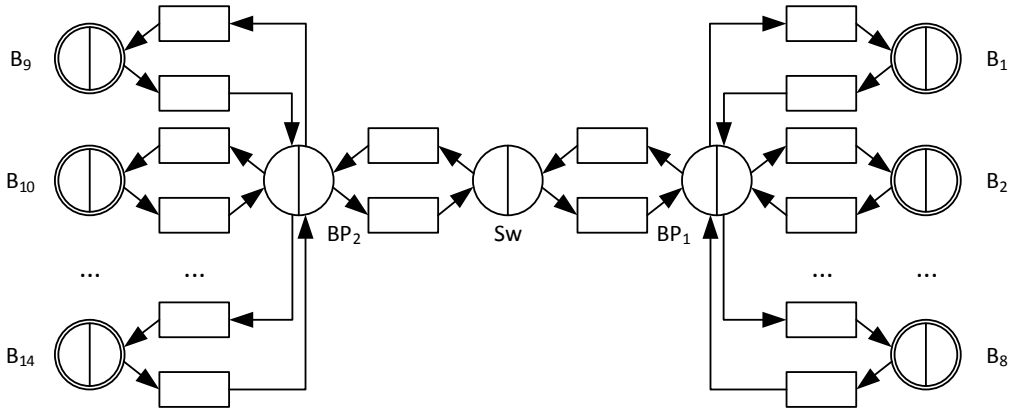
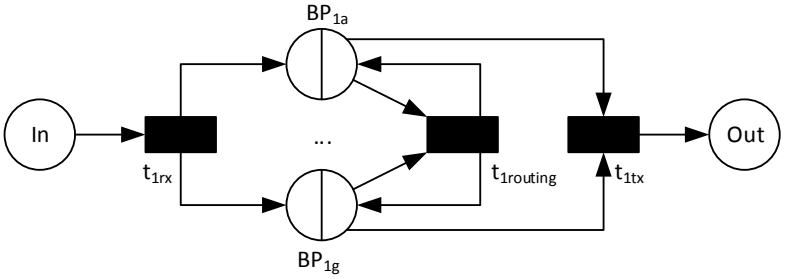


Abbildung 6.10: Oberste Ebene des hierarchischen QPN-Modells zur Simulation von Experiment F. Die Stelle B_1 repräsentiert den *YCSB*-Client, der auf einen *Cassandra*-Cluster zugreift. Die Clusterknoten werden durch die Stellen B_2 bis B_{14} repräsentiert. Die Stellen BP_1, BP_2 und *Sw* repräsentieren die Switche, mit denen die Clusterknoten untereinander Daten austauschen können.

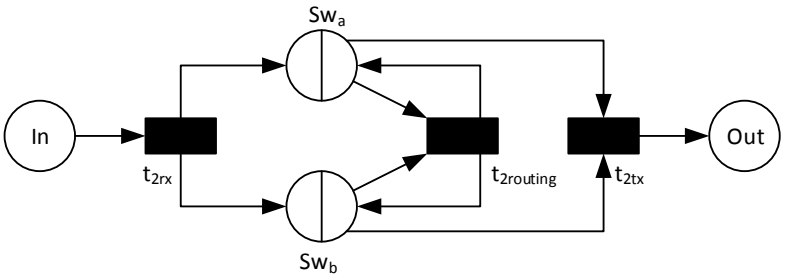
Die Funktionsweise der inneren Modelle, wie sie in Abbildung 6.11 dargestellt sind, unterscheiden sich nicht von der Funktionsweise des inneren Modelles von Abbildung 6.8 auf Seite 184. Das bedeutet unter anderem, dass ein- und ausgehende Netzwerkmarken eine Farbcodierung nutzen, um die Quelle und das Ziel einer Netzwerkmarke festzulegen. Für eine detaillierte Beschreibung der Arbeitsweise sowie der Semantiken sei auf den vorhergehenden Textabschnitt ab Seite 184 verwiesen.

Um Simulationen anhand des hierarchischen QPN-Modells durchzuführen, muss das in Abbildung 6.10 gezeigte QPN-Modell angepasst werden. Diese Anpassungen beziehen sich im wesentlichen auf die Konfigurarion der inneren Modelle der Stellen BP_1, BP_2 und *Sw*, um die technischen Charakteristiken des Versuchsaufbaus nachzubilden. Das bedeutet im genauerem, dass die Warteschlangen der Stellen BP_{1a} bis BP_{1g}, BP_{2a} bis BP_{2f} sowie Sw_a und Sw_b angepasst werden müssen, wie sie in drei Teilabbildungen von Abbildung 6.11 zu erkennen sind. Für diese Anpassungen können dieselben Einstellungen genutzt werden, wie sie bereits im vorhergehenden Textabschnitt beschrieben sind. Das bedeutet, dass die Warteschlangen der Stellen BP_{1a} bis BP_{1g} sowie BP_{2a} bis BP_{2f} auf eine maximale Kapazität von 98.426 Netzwerkmarken konfiguriert wurden. Das entspricht einem maximalen Netzwerkmarkendurchsatz plus 10 Prozent Überhangspuffer. Dies repräsentiert eine maximale Datenübertragungsrate von 1 GBit pro Sekunde, wie sie zwischen den Knoten eines Bladecenters im Versuchsaufbau von Experiment F zum Einsatz kam. Die Kapazität der Warteschlange von den Stellen Sw_a und Sw_b wurden jeweils auf 590.555 Netzwerkmarken festgelegt. Das entspricht ebenfalls einem maximalen

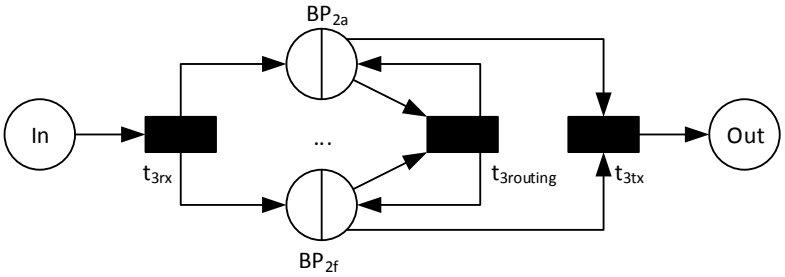
Netzwerkmarkendurchsatz plus 10 Prozent Überhangspuffer und repräsentiert die maximale Datenübertragungsrate von 6 GBit pro Sekunde zwischen den beiden modellierten Bladecentern des Versuchsaufbaus von Experiment F. Zusätzlich wurden die beiden genannten Stellen darauf konfiguriert, sechs Netzwerkmarken gleichzeitig verarbeiten zu können. Das modelliert den Umstand, dass die beiden Bladecenter des Versuchsaufbaus mittels Bündelung von sechs einzelnen Schnittstellen[9] miteinander verbunden waren. Hierbei wurde ein Lastverteilungsverfahren verwendet, dass alle sechs gebündelten Schnittstellen verwendete, wobei jede Schnittstelle einen maximalen Datendurchsatz von 1 GBit pro Sekunde bot ($6 \cdot 1\ \mathit{GBit} = 6\ \mathit{GBit}$).



(a) Inneres Modell von Stelle BP_1



(b) Inneres Modell von Stelle *Sw*



(c) Inneres Modell von Stelle BP_2

Abbildung 6.11: Innere Modelle der Stellen BP_1, BP_2 und *Sw*

Wie bereits erwähnt, wurden auf der mittleren und untersten Ebene des hierarchischen QPN-Modells zur Simulation von Experiment F keinerlei Änderungen gegenüber dem hierarchischen

[9] Das Verfahren wird als *link aggregation* bezeichnet. Hierbei werden mehrere Schnittstellen (*ports*, *links*) zu einer logischen Schnittstelle gebündelt. Auf dieser kann dann ein Lastverteilungsverfahren genutzt werden, das die Datenübertragungsrate vervielfacht.

QPN-Modells vorgenommen, das für die Simulation von Experiment E genutzt wurde. Das bedeutet genauer, dass die Anpassungen der erweiterten Modelle für das umgebende Betriebssystem und des Datenbanksystems komplett übernommen wurden, wie sie in vorhergehenden Textabschnitt beschrieben sind. Das betrifft zum Beispiel die initiale Markenbelegung, wie sie in Tabelle 6.10 auf Seite 186 dargestellt ist. Ein Unterschied ergab sich jedoch bei den Transitionsregelsätzen, die in den Instanzen des erweiterten Datenbanksystems auf unterster Ebene des hierarchischen QPN-Modells eingesetzt werden. Sie modellieren den Datenfluss beziehungsweise die Architektur des eingesetzten verteilten Datenbanksystems *Cassandra*. Sie wurden in einem ersten Schritt übernommen und in einem zweiten Schritt optimiert. Das heisst, Transitionsregeln wurde in den Transitionsregelsätzen neu angeordnet, wobei aber die ursprüngliche Semantik erhalten blieb. Durch die Neuanordnung konnten Transitionsregeln eingespart werden, die ansonsten eine Redundanz dargestellt hätten. Die Ersparnis hing dabei von der simulierten Arbeitslast des simulierten *YCSB*-Benchmarks ab, aber betrug im Durchschnitt rund 24 Prozent im Vergleich zu den originalen Transitionsregelsätzen.

Simulationsdurchführung Im Vergleich zur Durchführung der Simulation von Experiment E ergaben sich hinsichtlich der Simulation von Experiment F nur drei Änderungen. Diese Änderungen basieren auf der Beschreibung von Experiment F auf Seite 107 und besagen, dass nur Szenario A bei gleichzeitiger Benutzung der diskreten Gleichverteilung für eine erhöhte Anzahl von *Cassandra*-Clusterknoten gegenüber dem originalen Experiment E simuliert werden soll. Dadurch reduzierte sich die Anzahl der Testserien von 20 auf 11, die simuliert werden mussten (1 Verteilung $\times$ 1 Szenario $\times$ 11 *Cassandra*-Clustergrößen). Jede Testserie bestand dabei aus den Arbeitslasten `load`, `read`, `write` und `mixed`. Ingesamt wurden 132 einzelnen Simulationen im Rahmen der Simulation von Experiment F durchgeführt (11 Testserien $\times$ 4 Arbeitslasten $\times$ 3 Wiederholungen). Die Simulation der 11 Testserien wurde nach demselben Simulationsschema unter Beachtung der obigen Änderungen durchgeführt, wie sie auf Seite 188 beschrieben ist.

Zur Simulation der Testserien von Experiment F kam dieselbe dieselbe Simulationssoftware und derselbe Simulationsserver zum Einsatz, wie sie auch schon zur Simulation der Experimente B, C und E verwendet wurden. Das bedeutet, dass in gleicher Weise die QPN-Simulationssoftware *QPME* in Version 2 zur Erstellung des hierarchischen QPN-Modells und dessen Verwendung zur Simulation von Experiment F benutzt wurde. Die technischen Charakteristiken des genutzten Simulationsservers sind in Tabelle 6.6 auf Seite 173 aufgelistet.

Analog zu den Simulationen der Experimente B, C und E wurde auch bei den einzelnen durchgeführten Simulation der Testserien von Experiment F dieselben stochastischen Werte des hierarchischen QPN-Modells inklusive alle ausgeführten Instanzen der erweiterten Modelle des umgebenden Betriebssystems und des Datenbanksystems aufgezeichnet. Wie bereits in den vorhergehenden Textabschnitten erläutert, erlauben diese stochastischen Werte zusammen mit der simulierten Zeit, bis eine Simulation terminierte, einen direkten Vergleich mit den experimentellen Ergebnissen. Das ermöglicht auch hier die Berechnung der Genauigkeit gemäß der Definition auf Seite 165 für die Performance in Form der Antwortzeit und des Gesamtenergieverbrauches.

Aus den drei Wiederholungen einer simulierten Testserie wurden die Durchschnittswerte errechnet und mit den experimentell ermittelten Durchschnittswerten von Experiment F verglichen. Die daraus resultierenden prozentualen Differenzen, also die Genauigkeiten, sind in Tabelle 6.13 auf Seite 199 dargestellt. Diese Tabelle ist in drei Teiltabellen 6.13(a) bis 6.13(c) aufgeteilt. Jede dieser Teiltabellen hat denselben Aufbau: in jeder Zeile sind die Genauigkeitswerte für die simulierten Arbeitslasten `load`, `read`, `write` und `mixed` für die simulierten *Cassandra*-Clustergrößen (minimal 3 und maximal 14 Clusterknoten) dargestellt. Die letzte Spalte, die mit dem Symbol

"∅" bezeichnet ist, stellt den Durchschnittswert der gesamten jeweiligen Zeile dar. Das entspricht also dem Durchschnittswert der Genauigkeit einer simulierten Arbeitslast über alle simulierten *Cassandra*-Clustergrößen.

Simulationsergebnisse In Teiltabelle 6.13(a) auf Seite 199 sind die durchschnittlichen Genauigkeiten für die Antwortzeiten dargestellt. In dieser Teiltabelle kann festgestellt werden, dass die Genauigekeiten mit steigender simulierter *Cassandra*-Clustergröße ansteigen. Die Anstieg ist aber nur ein Trend und folgt keiner Gesetzmäßigkeit im Sinne einer Proportionalität. Vergleicht man die Genauigkeitswerte für die Antwortzeiten von Simulation E in Tabelle 6.11 auf Seite 190 mit denjenigen aus Teiltabelle 6.13(a), so ist dasselbe Trendverhalten erkennbar. Eine detaillierte Analyse der aufgezeichneten stochastischen Werte der Einzelsimulationen wurde durchgeführt, um dieses Verhalten zu ergründen. Das Ergebnis dieser Analyse war, dass mit zunehmender Komplexität des hierarchischen QPN-Modells auch die Menge an Marken zunahm, die es dann im Sinne des Datenflusses zu simulieren galt. Dabei bleibt die Eingabemenge bei der Simulation von Experiment F durch die alleinige Simulation von Szenario A zwar gleich, aber die Netzwerkmarken wurden dynamisch zur Simulationszeit erzeugt. Deren Anzahl ist stark von der Größe des simulierten *Cassandra*-Clusters abhängig. Diese Feststellung ist auch bei den zusätzlich durchgeführten Simulationen von Experiment E erkennbar, bei denen Simulationen *Cassandra*-Clustergrößen mit mehr als 25 Clusterknoten ausgeführt wurden. Insgesamt ist das Ergebnis der Analyse, dass die Genauigkeit steigt, je granularer das Modell ist.

Vergleicht man die Genauigkeiten für die Antwortzeit der modifizierenden mit denen der nichtmodifzierenden Arbeitslasten in Teiltabelle 6.13(a), also `load` und `write` gegenüber `read` und `mixed`, so ist auch erkennbar, dass erstgenannte höher ausfallen als letztgenannte. Der Unterschied beträgt im Mittel 3.28 Prozent. Das Verhalten von *Cassandra*, die Antwortzeiten bei modifizierenden Arbeitslasten durch massives Puffern drastisch zu senken, konnte in den Simulationen bestätigt werden. Zusätzlich werden die Simulationsergebnisse von Experiment E aus dem vorhergehenden Textabschnitt bestätigt, die zu einem ähnlichen Ergebnis kamen.

In Teiltabelle 6.13(b) werden in ähnlicher Weise wie in Teiltabelle 6.13(a) die Genauigkeiten für den Gesamtenergieverbrauch des simulierten *Cassandra*-Clusters widergegeben. Analog zum vorherigen Textabschnitt, in der die Simulationsergebnisse von Experiment E dargestellt sind, sei darauf hingewiesen, dass der Gesamtenergieverbrauch eine errechnete Größe ist, die sich als Produkt zwischen einem Basiswert pro allokierter und deallokierter Ressourcenmarke und dem mittleren Markendurchsatz der Ressourcenstellen zusammensetzt. Vergleicht man die Teiltabellen 6.13(a) und 6.13(b) miteinander, so ist auch in Teiltabelle 6.13(b) erkennbar, dass die dargestellten Genauigkeiten mit steigender simulierter *Cassandra*-Clustergröße ansteigen. Allerdings sind die Genauigkeitswerte für den Gesamtenergieverbrauch deutlich geringer als für die Antwortzeit. Die Gründe dafür sind dieselben, wie sie bereits detailliert im vorhergehenden Textabschnitt dargestellt wurden. Das bedeutet, dass nicht der Datenfluss innerhalb der Simulationen der Ursprung der Problematik ist, sondern die Energieverbrauchswerte der Ressourcenmarken. Es ergibt sich hinsichtlich der Genauigkeitswerte für den Gesamtenergieverbrauch wiederum der Umstand, dass präzise experimentelle Messungen der technischen Komponenten notwendig sind, um diese Basiswerte exakt zu bestimmen. Aufgrund des Versuchsaufbaus von Experiment F beziehungsweise der nicht dokumentierten, propritären Stromschienen in den genutzten Bladecenters war dieses Vorgehen aber nicht möglich.

	←				*Cassandra*-Clustergröße						→	
Arbeitslast	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**∅**
`load`	89.12	90.04	91.67	92.72	93.46	94.11	94.89	95.53	96.12	97.02	97.86	93.87
`read`	85.81	86.75	87.93	88.87	89.56	90.07	90.84	91.47	92.18	93.25	93.98	90.06
`write`	88.59	89.61	90.48	91.22	92.37	93.09	93.96	94.59	95.12	96.05	96.73	92.89
`mixed`	87.95	88.46	88.89	89.26	89.84	90.20	90.63	90.95	91.21	91.68	92.02	90.10

(a) Genauigkeit für die Antwortzeit

	←				*Cassandra*-Clustergröße						→	
Arbeitslast	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**∅**
`load`	77.19	78.02	79.76	80.85	81.59	82.34	83.11	83.98	84.52	85.23	86.08	82.06
`read`	65.44	66.58	67.61	68.33	68.89	70.51	71.23	71.91	72.67	73.62	74.28	70.09
`write`	75.02	75.95	77.73	78.75	79.44	80.29	81.01	81.96	82.42	83.11	83.97	79.96
`mixed`	70.23	71.26	72.67	73.54	74.16	75.40	76.12	76.93	77.54	78.36	79.12	75.03

(b) Genauigkeit für den Gesamtenergieverbrauch

	←				*Cassandra*-Clustergröße						→	
Arbeitslast	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**∅**
`load`	83.15	84.03	85.71	86.78	87.52	88.22	89.02	89.75	90.32	91.13	91.97	87.96
`read`	75.62	76.66	77.78	78.60	79.23	80.29	81.04	81.69	82.42	83.44	84.13	80.08
`write`	81.80	82.78	84.11	84.98	85.90	86.69	87.49	88.27	88.77	89.58	90.35	86.43
`mixed`	79.09	79.87	80.78	81.40	82.01	82.80	83.37	83.94	84.38	85.03	85.57	82.56

(c) Genauigkeit für die Energieeffizienz

Tabelle 6.13: Genauigkeiten der Simulation von Experiment F in Prozent

Das ist allerdings nur ein Teilaspekt. Hierzu vergleiche man die Genauigkeitswerte für den Gesamtenergieverbrauch aus Teiltabelle 6.13(b) (Simulation von Experiment F) mit denjenigen aus Tabelle 6.12 auf Seite 191 (Simulation von Experiment E). Da erstgenanntes Experiment auf letztgenanntem basiert, ist es von Interesse, aus welchem Grund die Genauigkeitswerte im Durchschnitt um fast 23 Prozent auf 76.79 Prozent angestiegen sind. Gemäß Definition 13 der Genauigkeit auf Seite 165 ist sie die prozentuale Differenz zwischen dem experimentell gemessenen und simulierten Gesamtenergieverbrauch. Eine große Ungenauigkeit ist somit ein Indikator dafür, dass entweder das simulierte QPN-Modell das Experiment nicht realitätsgetreu genug nachbildet oder die experimentellen Messungen nicht präzise durchgeführt wurden. In Anbetracht der relativ hohen Genauigkeitswerte für die Antwortzeit der beiden Simulationen für Experiment E und F kann nicht davon ausgegangen werden, dass die zugrunde liegenden QPN-Modelle der Grund für die vergleichsweise geringen Genauigkeitswerte für den Gesamtenergieverbrauch sind. Vergleicht man hingegen die Beschreibungen der Versuchsaufbauten sowie die Durchführung der Experimente E und F miteinander, so ergeben sich Unterschiede. Sie wirken sich auf die experimentell gemessenen Gesamtenergieverbrauchswerte und damit auf die Genauigkeit aus. So führte bei Experiment E das Fehlen des administrativen Zuganges zum Versuchsaufbau dazu, dass Clusterknoten zum Gesamtenergieverbrauch beigetragen haben, obwohl sie für die Durchführung der spezifischen Testserien nicht notwendig waren. Dieser Umstand wurde bei Experiment F nicht wiederholt. Als Result liegen die experimentell gemessenen Energieverbrauchswerte näher an den denjenigen, die per Simulation evaluiert wurden. Insgesamt stieg dadurch die Genauigkeit.

Ein weiterer Aspekt sind auch die optimierten Transitionsregelsätze, die bereits in diesem Textabschnitt erläutert wurden. Die Optimierungen haben dazu beigetragen, dass die mittlere Genauigkeit über alle Testserien von Experiment F um 11.8 Prozent auf 91.73 Prozent stieg, wenn man

sie mit denen von Experiment E vergleicht. Das bedeutet, dass im Durchschnitt die simulierten Antwortzeiten nur um 8.27 Prozent von den experimentellen abweichen. Das hierarchische QPN-Modell, das für die Simulation von Experiment F genutzt wurde, gilt auch für die Bestimmung der Genauigkeit des Gesamtenergieverbrauches. Es kann daher nicht ausgeschlossen werden, dass die optimierten Transitionsregelsätze und damit der Datenfluss, der durch die Simulation entsteht, einen positiven Einfluss auf die Genauigkeit des Gesamtenergieverbrauches hat.

Im Gegensatz zur Simulation von Experiment E, dessen Ergebnisse im vorherigen Textabschnitt 6.3.1 erläutert sind, wurde für die Simulation von Experiment F zusätzlich die Energieeffizienz der simulierten Testserien berechnet. Das ermöglicht den direkten Vergleich mit den experimentellen Ergebnissen hinsichlich der Energieeffizienz, wie sie in Textabschnitt 4.2.2 ab Seite 107 beschrieben sind. Zudem wird dadurch die Möglichkeit geschaffen, die Genauigkeit bezüglich der Energieeffizienz zu errechnen. Die Berechnung der simulierten Energieeffizienz erfolgte anhand von Gleichung 3.3 auf Seite 64 unter Nutzung der simulierten Antwortzeit und des simulierten Gesamtenergieverbrauches.

Die Genauigkeitswerte für die Energieeffizienz sind in Teiltabelle 6.13(c) auf Seite 199 verzeichnet. Diese Teiltabelle folgt dabei derselben Struktur wie die beiden anderen Teiltabellen 6.13(b) und 6.13(b). Das bedeutet, die dargestellten Genauigkeitswerte sind Durchschnittswerte der drei Wiederholungen einer Arbeitslast für eine simulierte *Cassandra*-Clustergröße. Die mittlere Genauigkeit der Energieeffizienz, also über alle Arbeitslasten und *Cassandra*-Clustergröße, beträgt 84.25 Prozent. Damit weichen im Durchschnitt die simulierten Genauigkeitswerte für die Energieeffizienz um 15.75 Prozent von den experimentellen ab.

Die experimentellen Werte für die Energieeffizienz von Experiment F wurden bereits in Abbildung 4.19 auf Seite 113 dargestellt. Diese Abbildung wurde um die simulierten Energieeffizienzwerte erweitert, um einen optischen Vergleich zu ermöglichen. Zusätzlich wurde der Energieeffizienzgewinn pro zusätzlichem simulierten *Cassandra*-Clusterknoten der Abbildung hinzugefügt. Alle Erweiterungen sind in Abbildung 6.12 auf Seite 201 dargestellt.

In Abbildung 6.12 sind insbesondere die beiden verbundenen, schwarzen Linien von Interesse. Wie bereits erwähnt zeigen sie den Gewinn an Energieeffizienz eines *Cassandra*-Clusters mit n gegenüber einem mit $n-1$ Clusterknoten. Dieser Gewinn kann als prozentualer Anstieg der Energieeffizienz für diese beiden Clustergrößen beschrieben werden. Vergleicht man die Linien des experimentellen und simulierten Energieeffizienzgewinns miteinander, so kann man feststellen, dass zweitere der ersteren folgt. Zudem ist erkennbar, dass die Differenz zwischen mit zunehmender simulierter *Cassandra*-Clustergröße abnimmt. Im Durchschnitt beträgt die Differenz 2.87 Prozent.

Wesentlich wichtiger ist jedoch, dass der simulierte Gewinn an Energieeffizienz dieselbe Aussage hinsichtlich der Optimums bestätigt. Das bedeutet, dass die Anzahl an *Cassandra*-Clusterknoten ermittelt wird, bei der die Performance in Form der Antwortzeit und die dafür aufgewendete elektrische Energie optimal aufeinander abgestimmt sind. In Experiment E wurde diese Anzahl durch lineare Regression mathematisch errechnet und in Experiment F experimentell bestätigt. Die Simulationsergebnisse von Experiment F, die in diesem Textabschnitt beschrieben wurden, zeigen eine relativ hohe Genauigkeit. Das bedeutet, dass das verwendete hierarchische QPN-Modell und die darin genutzten Instanzen der QPN-Modelle in der Lage sind, die experimentellen Ergebnisse recht akurat zu reproduzieren.

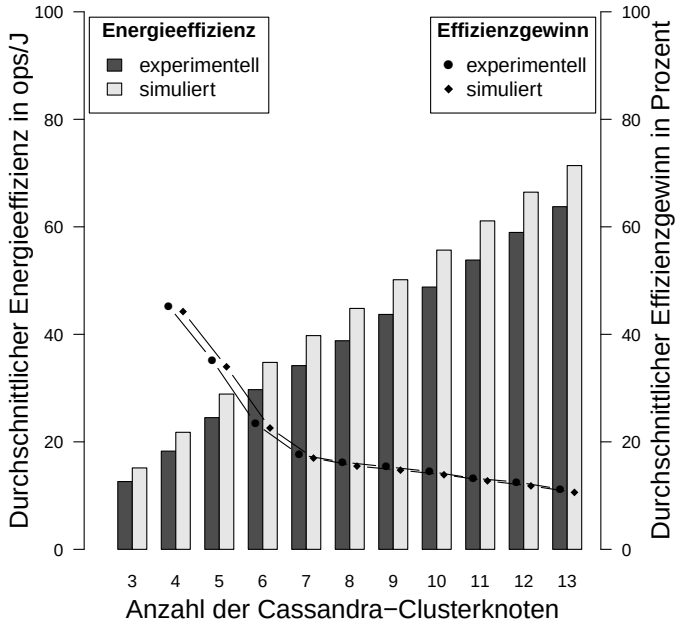


Abbildung 6.12: Durchschnittliche Energieeffizienz für die experimentellen und simulierten *Cassandra*-Clustergrößen aus Experiment F[10]

Das bedeutet im Umkehrschluss, dass es die angepassten QPN-Modelle ermöglichen, die Architektur des verteilten Datenbanksystems *Cassandra* sowie der technischen Plattform nachzubilden, auf der es eingesetzt wird. Sie zeigen auch, dass der Datenfluss einer Datenbankanwendung in Sinne eines Benchmarks abgebildet und simuliert werden kann.

6.4 Qualitative Analyse der Simulationsergebnisse

Die vorangegangenen zwei Textabschnitte 4.1 und 4.2 haben gezeigt, dass es möglich ist, sowohl ein zentral betriebenes als auch ein verteiltes Datenbanksystem zu simulieren, wobei die technische Plattform und das umgebende Betriebssystem in die Simulation mit einbezogen wurde.

Zur Bewertung der Simulationsergebnisse wurde der Begriff der Genauigkeit eingeführt, die den prozentualen Unterschied zwischen experimentellen und simulierten Metriken angibt. Zwei maßgebliche Genauigkeiten wurden in den letzten beiden Textabschnitten genutzt: die Genauigkeit für die Performance (zumeist in Form der Antwortzeit) und für den Energieverbrauch. Beide können verwendet werden, um schlussendlich die Genauigkeit für die Energieeffizienz gemäß Gleichung 3.3 auf Seite 64 zu berechnen.

In Textabschnitt 6.1 wurden die Qualitätsfaktoren für eine Simulation inklusive der zugrunde liegenden Modelle nach *Robinson* erläutert. Diese Faktoren sind in Tabelle 6.1 auf Seite 166 verzeichnet, wobei zusätzlich angegeben wurde, welche Relevanz diese Faktoren für diese Dis-

[10] Aus Darstellungsgründen wurde Abbildung 6.12 nicht numerisch annotiert. Die numerischen Werte sind im Anhang A.1.4 tabellarisch dargestellt.

sertation haben. Durch die Simulationsergebnisse, die in den beiden letzten Textabschnitten erläutert wurden, können die Qualitätsfaktoren bewertet werden. Im folgenden wurden nur die Qualitätsfaktoren ausgewertet, deren Relevanz in Tabelle 6.1 mindestens mit “hoch” eingestuft wurden:

- **Modell: Geschwindigkeit, Ästhetik und Benutzbarkeit** Hinsichtlich der Ausführungszeit, in der die Simulationen der QPN-Modelle durchgeführt wurden, muss eine recht hohe Qualität zugesprochen werden. Dazu muss aber bemerkt werden, dass eine hochperformante technische Plattform zur Ausführung der Simulationssoftware genutzt wurde. Infolgedessen waren die Simulationszeiten mit einer Spanne von wenigen Minuten bis knapp einer Stunde recht gering. Das ist nur ein Bruchteil der Zeitspanne, die es brauchte, um die Experimente durchzuführen, die schlussendlich simuliert wurden.
  Hinsichtlich der Benutzbarkeit kann festgestellt werden, dass die QPN-Modelle im besonderen Maß darauf ausgelegt sind, miteinander kombiniert und erweitert zu werden. Das bezieht sich auf die QPN-Modelle, die im Kapitel 5 eingeführt und beschrieben wurden. Die Erweiterbarkeit ergibt sich daraus, dass an zentralen Stellen immer die Möglichkeit gegeben ist, die grundlegenden Modelle durch innere Modelle zu verfeinern. Dadurch können spezielle Architekturen oder Strukturen von Datenbanksystemen, technischen Plattformen oder Betriebssystemen in ihrem Verhalten nachgebildet werden.
  Zusätzlich kann festgestellt werden, dass die QPN-Modelle im Bezug auf die Benutzbarkeit weder auf ein Datenbanksystem, eine technische Plattform oder eine spezifische Datenbankanwendung festgelegt sind. Somit sind die QPN-Modelle auch in der Lage, Datenbankanwendungen außerhalb der modellierten Datenbanksystem-Benchmarks zu simulieren. Damit sind sie insoweit anpassbar, eine Vielzahl an Kombinationsmöglichkeiten dieser drei Punkte zu modellieren. Insgesamt kann diesem Qualitätsfaktor ein hoher Erfüllungsgrad gegeben werden.
- **Vertrauen in das Modell** Dieser Qualitätsfaktor wird durch die Genauigkeit bestimmt. Das bedeutet, dass generell die Metriken, die per Simulation gewonnen werden, mit realen Metrikwerten verglichen werden, um so den Grad der Realitätstreue zu bestimmen. Dieser Qualitätsfaktor hat eine sehr hohe Bedeutung, da er maßgeblich die Akzeptanz, Plausibilität und Glaubwürdigkeit der Simulation begrenzt. Aus diesem Grund werden die Genauigkeiten, die in den letzten beiden Textabschnitten 4.1 und 4.2 gezeigt wurden, im weiteren Verlauf dieses Textabschnittes gesondert betrachtet.
- **Simulationsdaten: Verfügbarkeit und Genauigkeit** Durch die Verwendung von standardisierten und allgemein anerkannten Benchmarks *TPC-H*, *StarSchema Benchmark* und *Yahoo Cloud Serving Benchmark* stehen die Simulationsdaten öffentlich, nachprüfbar und vergleichbar zur Verfügung. Diese Benchmarks definieren sehr genau die zu messenden Metriken sowie den Versuchsaufbau und die Verfahren, die bei der Evaluation der experimentellen Ergebnisse zum Einsatz kommen müssen. Insgesamt kann dieser Qualitätsfaktor als sehr gut erfüllt angesehen werden.
- **Simulationssoftware: Benutzbarkeit, Eignung, Flexibilität, Glaubhaftigkeit** Die durchgehend verwendete Simulationssoftware *QPME* ist eine anerkannte Software, die in vielen wissenschaftlichen Publikationen dazu verwendet wurde, um komplexe QPN-Modelle zu gestalten und sie anschließend dazu zu verwenden, um Simulationen durchzuführen; unter anderem in [KD09, RK14, NRKR14, COK13].

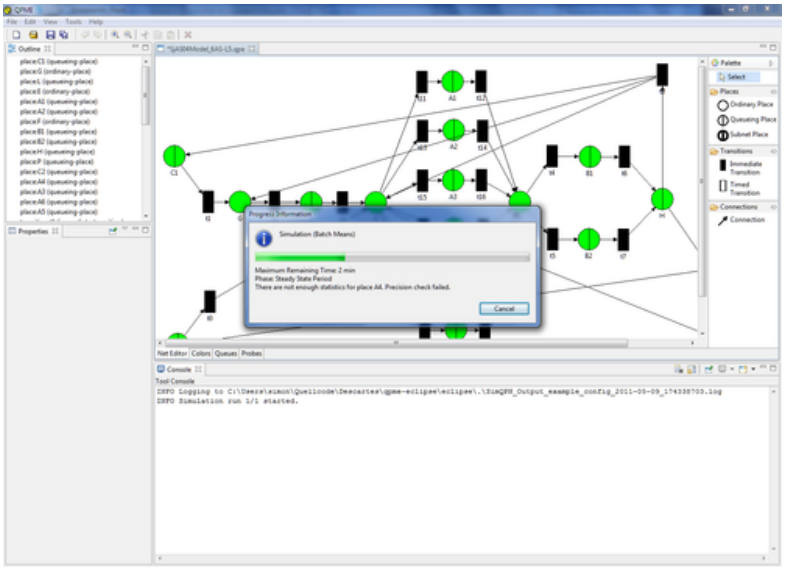


Abbildung 6.13: *Queuing Petri Net Modelling Environment* (QPME)[11]

Insbesondere durch die Möglichkeit, QPN-Modelle durch den bereitgestellten optischen Editor einfach und unkompliziert zu erstellen und zu konfigurieren, erleichtert die Modellierung maßgeblich. Dies ist ein herausstellendes Anwendungsmerkmal im Vergleich zu den weniger komfortablen Alternativen, in der QPN-Modelle textuell durch entsprechende Beschreibungssprachen definiert werden.
Zudem ist der Simulationskern sehr performant ausgelegt. Das wurde dadurch bewerkstelligt, dass er den Markenfluss ereignisgetrieben ausführt. Diese Architektur trägt dazu bei, die Simulationsdauer zu minimieren.

Der letzte Qualitätsfaktor *Kompetenz des Modellierenden* wurde nicht betrachtet und bewertet, da es außer dem Autor dieser Dissertation noch keine weiteren Personen existieren, die die QPN-Modelle verwenden, die im Rahmen dieser Dissertation eingeführt und erläutert wurden.

Zudem ist es schwierig, die Simulationsergebnisse der verangegangenen beiden Textabschnitte mit anderen Publikationen zu vergleichen. Eine Literaturrecherche dazu ergab, dass bisher keine anderen QPN-Modelle veröffentlicht wurden, die annähernd die Komplexität derjenigen erreichen, die im Rahmen dieser Dissertation beschrieben sind. So existieren Publikationen

- die Teilaspekte des umgebenden Betriebssystems modellieren, zum Beispiel die Auslastung der CPU oder die Auslastung des modellierten System im allgemeinen. Solche simplen Modelle wurden zum Beispiel von *Bontempi et al.*, *Lo et al.* und *Noorshams et al.* veröffentlicht [BK02b, LBE+98, NRKR14]. Das Modell von *Kounev und Buchmann* in [KB03] beinhaltet zusätzlich eine Abbildung des Ressourcenverbrauches in Form von Arbeits- und Massenspeicher bei der Ausführung von Anwendungen auf einem zentralen Serversystem.
- die Datenbanksysteme als simple Modelle abbilden, zum Beispiel in den Publikationen von *Osman et al.*, *Casale* und *Coulden et al.* [OAW08, Cas16, COK13]. Durch Simulation der Modelle wird versucht, Vorhersagen zur Performance zu treffen. Die Modelle enthalten allerdings keine Möglichkeit, den Ressourcenverbrauch abzubilden. Zudem entsprechen die

[11] Quelle: `https://se.informatik.uni-wuerzburg.de/tools/qpme/screenshots/`, Abruf 05.05.2016

simulierten Arbeitslasten keinen Standards. Das bedeutet, dass synthetische Arbeitslasten simuliert wurden, um die Modelle an sich zu evaluieren. Daher könne keine Angaben zur Genauigkeit gemacht werden, da eine vergleichbare und experimentelle Basis fehlt.

- in denen Modelle veröffentlicht wurden, die eine Kombination aus den beiden vorangegangenen Punkten darstellen. Das gilt zum Beispiel für die Publikation von *Kihl et al.* und *Dellkrantz et al.* in [KAR+12, DKR12]. Der Fokus lag hier eindeutig auf der Modellierung von verteilten Datenbanksystemen. Dadurch wurde es notwendig, auch die Umgebung für diese zu modellieren. Somit entstand ein kombiniertes Modell, das rudimentär einen Cluster repräsentiert, das wiederum gewisse Anleihen am Modell des umgebenden Betriebssystems hat. Simulationen des kombinierten Modelles verfolgten das Ziel, die Performance des verteilten Datenbanksystems zu evaluieren. Dafür wurden Teile des *TPC-C*-Benchmarks als Eingabemenge für das kombinierte Modell verwendet.
- die Computernetzwerke modellieren, unter anderem von *Rygielski und Kounev* sowie *Chang et al.* in [RK14, CPH10]. Sie evaluieren die Performance von verteilten Anwendungen in einem Computernetzwerk, wonach prinzipiell auch verteilte Datenbanksysteme modelliert werden können. Dementgegen war die Zielstellung der beiden Publikationen der Beleg, dass Computernetzwerke modelliert und simuliert werden können, wobei der Fokus auf anderen Metriken bei der Simulation lag. Dazu gehört nicht nur die Performance, sondern im wesentlichen der Durchsatz.

Die genannten Publikationen haben alle gemeinsam, dass Modelle gezeigt werden, deren Simulation die Absicht haben, die Performance zu evaluieren. Der Ressourcenverbrauch von Betriebsmitteln, wie sie vom Betriebssystem verwaltet und der technischen Plattform bereitgestellt werden, wird in den meisten Fällen komplett unbeachtet. Darüber hinaus wird allen Modellen in den erwähnten Veröffentlichungen der Energieverbrauch weder simuliert noch betrachtet. Allerdings bilden erst die Evaluation der Performance und die gleichzeitige Evaluation des Energieverbrauches die Grundlage zur Berechnung der Energieeffizienz. Dies ist auch in den genannten Publikationen erkennbar, die die Thematik Energieeffizienz nicht aufgreifen.

Insgesamt kann eine klare Abgrenzung der QPN-Modelle, die im Rahmen dieser Dissertation eingeführt und erläutert wurden, gegenüber den Modellen gezogen werden, die bis dato veröffentlicht wurden. Sie sind in der Lage, sowohl die Performance, den Ressourcenkonsum an Betriebsmitteln und den daraus resultierenden Energieverbrauch als auch die Energieeffizienz für verteilte und zentrale Datenbanksysteme nachzustellen und zu simulieren. Diese drei Aspekte ergeben sich aus dem Datenfluss, der durch die Simulation einer Datenbankanwendung entsteht, zum Beispiel (aus Vergleichsgründen) einem Benchmark. Die eingeführten und erläuterten Modelle erlauben eine Kombination untereinander und sind insoweit justierbar, dass alle Einflussgrößen auf die drei genannten Aspekte modellier- und simulierbar sind.

Die Publikation von *Garcia* in [Gar09] ist insofern hervorzuheben, da das dort vorgestellte Modell der Abgrenzung von allen genannten Publikationen der Literaturrecherche am nächsten kommt. Besondere Bedeutung erlangt die Publikation von *Garcia* dadurch, dass einleitend eine Zusammenfassung aller wissenschaftlichen Publikationen und Studien gegeben wird, die primär QPNs einsetzen, um die Performance eines Datenbanksystems zu evaluieren. Das von *Garcia* vorgeschlagene QPN-Modell umfasst eine vergleichsweise simple Repräsentation des zentralen Datenbanksystems *SQL Server 2000* sowie eine einfache Repräsentation der verwendeten technischen Plattform.

Genauer gesagt, es wurden nur die technischen Komponenten CPU und Massenspeicher modelliert. Eine Repräsentation des verwendeten, umgebenden Betriebssystems *Windows Server 2003* fand nicht statt. Als Datenbankanwendung wurde der *TPC-C*-Benchmark verwendet. Auch hier wurden die experimentellen Ergebnisse des *TPC-C*-Benchmarks genutzt, um sie mit denen aus den Simulationen zu vergleichen.

Zusammengefasst bedeutet die Analyse der Literaturrecherche, dass eine vergleichende Einschätzung der Qualität der QPN-Modelle aufgrund der Simulationsergebnisse der vorangegangenen zwei Textabschnitte und insbesondere der Genauigkeitsangaben nicht realistisch ist. Der Grund dafür ist, dass es schlicht keine Vergleichsmöglichkeiten gibt. Dementsprechend kann keine Aussage gemacht werden, wie "gut" oder wie "schlecht" die QPN-Modelle dieser Dissertation geeignet sind, die Abgrenzungseigenschaften zu erreichen.

Dennoch verbleibt die Option, die Genauigkeitsangaben der beiden Textabschnitte 6.2 und 6.3 miteinander zu vergleichen. Hier ergibt sich das Bild, dass die Genauigkeiten für die Performance höher ausfallen als die Genauigkeiten für den Energieverbrauch. Durch den mathematischen Zusammenhang zwischen beiden gemäß Gleichung 3.3 wirkt sich das auf die Genauigkeit der Energieeffizienz aus. Allerdings ist auch feststellbar, dass die Genauigkeiten durch bessere Kenntnis und Optimierungen der eingesetzten QPN-Modelle angestiegen sind. Dafür wurden die entsprechenden Genauigkeiten in Tabelle 6.14 zur besseren Übersicht zusammengefasst.

		← *Genauigkeit in Prozent* →		
Experiment		**Performance**	**Energieverbrauch**	**Energieeffizienz**
B	(TPC-H, *Postgresql*)	63.45	(nicht evaluiert)	78.09
C	(SSB, *Postgresql*)	80.23	(nicht evaluiert)	79.23
E	(YCSB, *Cassandra*)	79.93	53.75	(nicht evaluiert)
F	(YCSB, *Cassandra*)	91.73	76.79	84.25

Tabelle 6.14: Übersicht über die Genauigkeiten der simulierten Experimente

In Tabelle 6.14 sind für jede Simulation eines Experimentes die Genauigkeiten aufgeführt, wobei auch zur besseren Unterscheidung der verwendete und simulierte Benchmark und das Datenbanksystem vermerkt sind. Aufgrund der Zielstellung der Experimente und aus Vergleichsgründen sind nicht alle Genauigkeiten evaluiert worden. Dieser Umstand ist in Tabelle 6.14 mit der Eintrag "(nicht evaluiert)" notiert worden.

Dennoch ergibt sich mit den Genauigkeitsangaben für Experiment F die Situation, dass sich die experimentellen und simulierten evaluierten Metriken nur um wenige Prozent unterscheiden. Besonders die Genauigkeit für die Performance erreicht mit fast 92 Prozent das theoretische Maximum von 100 Prozent.

6.5 Kapitelzusammenfassung

Um die QPN-Modelle, die in Kapitel 5 eingeführt und beschrieben wurden, sowohl quantitativ also auch qualitativ zu evaluieren, wurden sie dafür genutzt, die Experimente aus Kapitel 4 nachzustellen. Dieses Vorgehen wurde aus dem Grund gewählt, um zu belegen, dass die QPN-Modelle grundsätzlich in der Lage sind, Datenbankanwendungen in Form von Benchmarks für reale Datenbanksysteme und die verwendete technische Plattform zu modellieren und zu simulieren.

Für die qualitative Evaluation ist es erforderlich, in welcher Form die Qualität oder Güte bemessen werden kann. Daher wurde in Textabschnitt 6.1 eine Literaturrecherche durchgeführt. Dessen Ergebnis mündete in der Definition der Genauigkeit. Sie gibt die prozentuale Differenz zwischer einer experimentell gewonnenen Metrik, zum Beispiel die Performance in Form der Antwortzeit, und derjenigen aus der Simulation an. Dadurch ergibt sich eine Bemessungsgrundlage, inwiefern die QPN-Modelle die realen Experimente aus Kapitel 4 reproduzieren können. Die Genauigkeit dient auch als Indikator dafür, inwiefern die eingesetzten QPN-Modelle die Charakteristiken der modellierten Kombination aus Datenbanksystem, der genutzten technischen Plattform und Datenbankanwendung umsetzen und simulieren können.

Für die Simulation der Experimente aus Kapitel 4 ist es natürlich notwendig, die QPN-Modelle an die eigentlichen Experimente anzupassen. Dafür wurden die Verfahrensweisen genutzt, wie sie in den Textabschnitten 5.4.2 bis 5.4.3 sowie 5.7.2 beschrieben wurden. Das beinhaltet dadurch die Anpassung der QPN-Modelle an die getesteten Datenbanksysteme *Postgresql* und *Cassandra* sowie an die verwendete technische Plattform und dem Benchmark. Diese Aspekte sind in den Textabschnitten 6.2 und 6.3 beschrieben.

Diese beiden Textabschnitte beschreiben auch, wie die Simulationen durchgeführt wurden und welche Ergebnisse beobachtet werden konnten. So zeigte sich zum Beispiel bei der Simulation von Experiment B, dass sich das beobachtete Pufferverhalten innerhalb der Simulation nicht zeigte. Als Folge daraus wiesen die Genauigkeiten für die Performance in Form der Antwortzeiten teils signifikante Unterschiede auf. Daraus resultierte auch, dass der Energieverbrauch und schlussendlich die Energieeffizienz zwar erfolgreich simuliert werden konnten, aber relativ geringe Genauigkeiten aufwiesen.

Dementgegen zeigen die Genauigkeiten für die Performance der übrigen Experimente C, E und F, dass die Charakteristiken der evaluierten Datenbanksysteme, der Versuchsaufbauten und Benchmarks in den Simulationen reproduziert werden konnten. Hier sei besonders die Simulation von Experiment F hervorzuheben. Durch verschiedene Optimierungen an den Simulationen konnte die Genauigkeit insofern gesteigert werden, dass die prozentuale Differenz zwischen experimentell gewonnener und simulierter Performance in Form der Antwortzeit unter 10 Prozent beträgt. Das bedeutet, dass Experiment F durchaus realitätsgetreu simuliert werden konnte.

Die Qualitätsfaktoren für Simulationen, die einleitend in Kapitel 6 beschrieben wurden, sind für die QPN-Modelle in Textabschnitt 6.4 zusammenfassend re-evaluiert worden. Diese Evaluation belegt, dass die relevanten Qualitätsfaktoren sehr gut erfüllt worden sind. Das bedeutet genauer, dass die Hauptfaktoren Akzeptanz, Plausibilität und Glaubwürdigkeit hinreichend belegt worden sind. Die Qualität der QPN-Modelle kann auch dadurch bemessen werden, indem man sie mit anderen publizierten QPN-Modellen vergleicht, die dieselbe Zielstellung verfolgen. Eine Literaturrecherche ergab, dass zwar QPN-Modelle existieren, sie aber nicht mit denjenigen vergleichbar sind, die im Rahmen dieser Dissertation eingeführt und für Simulationen genutzt worden sind. Dementsprechend können diese QPN-Modelle klar gegenüber den bisher publizierten QPN-Modellen abgegrenzt werden.

Die Literaturrecherche ergab auch, dass die bis dato publizierten QPN-Modelle nicht den Komplexitätsgrad derjenigen erreichen, die innerhalb von Kapitel 5 beschrieben wurden. Die gesteigerte Komplexität äußert sich darin, dass nicht nur deutlich aufwendigere QPN-Modelle konzipiert, sondern auch Markenquantitäten genutzt wurden, die um mehrere Größenordnung über denen liegen, die in den besagten, extern publizierten QPN-Modelle benutzt wurden.

7 Zusammenfassung und zukünftige Weiterentwicklungen

In dieser Dissertation wurde das Forschungsthema “Energieeffizienz von Datenbanken” betrachtet. Dieses Forschungsthema ist deswegen von Bedeutung und in den Vordergrund gerückt, da traditionelle Datenbanksysteme zunehmend nicht mit in der Lage sind, das immense Informationsaufkommen effizient zu verarbeiten. Wie in Kapitel 1 und 2 erläutert, führte das zur Erforschung und Einführung von neuen Technologien, Verfahren und auch Datenbanksystemen, die in das Umfeld von *Big Data* gezählt werden können. Wie in den Textabschnitten 2.3 und 2.4 beschrieben, können insbesondere *NoSQL*-Datenbanksysteme Vorteile hinsichtlich der Abfrageperformance und im Umgang mit Informationen bringen, sofern diese Informationen einem Datenmodell entsprechen, auf das sich das Datenbanksystem spezialisiert hat.

Auch wenn effizientere, spezialisierte Datenbanksysteme verfügbar sind und genutzt werden, täuscht es dennoch nicht über die Tatsache hinweg, dass zur Verarbeitung der Informationsmengen eine steigende Anzahl an Servern gebraucht werden. Das hat mittlerweile dazu geführt, dass der Energieverbrauch der Server gerade in großen und sehr großen Datenverarbeitungszentren zu einem Kostenfaktor geworden ist, der nicht unterschätzt werden darf. Aus diesem Grund ist zunehmend die Energieeffizienz von Datenbanksystemen beziehungsweise dessen Optimierung in den Fokus gerückt.

Der Hauptbeitrag dieser Dissertation zu diesem Aspekt ist die Konkretisierung des Optimierungsproblems hinsichtlich der Energieeffizienz von Datenbanksystemen sowie ein neuer Ansatz zur Lösung dieses Problems. Wie in Textabschnitt 3.3 beschrieben, definiert sich die Energieeffizienz als die Division der Performance und dem Energieverbrauch eines Datenbansystems im Bezug auf einen gegebenen Anwendungsfall. Das daraus resultierende Optimierungsproblem, beschrieben in Kapitel 6, ist keineswegs neu. Bezogen auf *denselben* Anwendungsfalls besagt es, dass zur Maximierung der Energieeffizienz eines Datenbanksystems entweder die Performance erhöht oder der Energieverbrauch gesenkt werden muss (oder im optimalsten Fall beides). Für die Konkretisierung des Optimierungsproblems müssen daher die beiden Gesichtspunkte Performance und Energieverbrauch eingehend betrachtet werden:

- Die Evaluierung und Optimierung der Performance von Datenbanksystem ist bereits Gegenstand der Forschung seit deren Aufkommen vor über 30 Jahren. Dementsprechend liegen bereits viele Forschungsergebnisse zu diesem Thema vor. In Textabschnitt 3.1 wurde dargestellt, dass standardisierte Benchmarks genutzt werden, um die Performance von Datenbanksystemen zu bemessen. Der Verteil dabei ist, dass die Performancemetriken der Benchmarks vergleich- und nachvollziehbar sind. Der Beitrag dieser Dissertation ist, dass diejenigen Faktoren zusammengetragen wurden, die *generell* für alle Datenbanksysteme

gelten und den potentiell größten Einfluss auf die Performance haben. Dafür wurden auch Experimente für zentral betriebene und verteilte Datenbanksysteme durchgeführt, deren Ergebnisse in den Textabschnitten 4.1 und 4.2 erläutert sind.

- Ein weiterer Beitrag ist die Identifikation und Evaluation der technischen Komponenten einer Plattform, die primär einen Einfluss auf den Energieverbrauch eines Datenbanksystems haben. Sie wurden in Textabschnitt 3.2 erläutert und ihr Einflusspotential in Experimenten bemessen. Die experimentellen Ergebnisse sind in Textabschnitt 4.1 dargestellt. Problematisch hierbei ist, dass nur zwei anerkannte Standards existieren, um den Energieverbrauch eines Datenbanksystems vergleichbar zu bemessen. Sie werden aber in der Praxis häufig ignoriert.
- Auch die Wahl der technischen Kernkomponenten kann einen Einfluss auf die Performance eines Datenbanksystems haben. Wie in Textabschnitt 3.3 dargelegt, treten Wechselwirkungen auf. Das heisst, die Faktoren mit einem Einfluss auf die Performance und auf den Energieverbrauch eines Datenbanksystems beeinflussen sich gegenseitig. Als Resultat beeinflussen auch die beschriebenen Wechselwirkungen die Energieeffizienz.

In Kapitel 5 wurden die Lösungsmöglichkeiten für das Optimierungsproblem betrachtet und Vor- und Nachteile diskutiert. Experimentelle Verfahren, also die Ausführung eines standardisierten und anerkannten Benchmarks bei gleichzeitiger Messung des Energieverbrauches, haben den Vorteil, vergleichbare experimentelle Ergebnisse hinsichtlich der drei Aspekte Performance, Energieverbrauch und -effizienz zu evaluieren. Zudem können Wechselwirkungen erkannt und auch das Skalierungsverhalten eines Datenbanksystems kann ermittelt werden. Dementgegen hat dieses Vorgehen aber den Nachteil, dass es Investitionen in Zeit und Hardware bedeutet, sofern man eine vergleichende Studie mit mehreren variierten Parametern realisieren möchte, zum Beispiel der technischen Plattformen, Betriebssystemen und steigenden Datenmengen. Analytische Verfahren versuchen, die Einflussfaktoren in Form eines Modells zu beschreiben und durch mathematische Verfahren zu berechnen. Sie haben den Nachteil, dass das Modell stark auf einen Anwendungsfall beschränkt ist.

In Kapitel 5 wurde daher auch gezeigt, dass die Evaluation der Energieeffizienz eines Datenbanksystems auf die Evaluation des Datenflusses innerhalb eines Datenbanksystems zurückgeführt werden kann. Eine mögliche Form ist die Simulation des Datenflusses mit *Queued Petri Nets* (QPNs). Sie haben den Vorteil, dass sie auf *Petri*-Netzen basieren, die bereits Gegenstand der Forschung seit knapp drei Jahrzehnten sind und gezeigt haben, das Performanceverhalten von Systemen aller Art nachzustellen. So wurden bereits QPN-Modelle erarbeitet und genutzt, um die Performance von technischen Komponenten und Teilaspekte von Betriebssystemen, Datenbanksystemen und Computernetzwerken zu simulieren. Die QPN-Modelle, die in Kapitel 5 eingeführt und erläutert wurden, basieren zum Teil auf diesen Modellen. Der Beitrag dieser Dissertation ist jedoch, dass diese QPN-Modelle völlig neu mit dem Ziel konzipiert wurden, alle oben genannten Einflussfaktoren abzubilden. Die grundlegenden QPN-Modelle können dabei hierarchisch kombiniert werden und erlauben, ein deutlich komplexeres QPN-Modell zu formen, um selbst diffizile Datenbanksysteme modellieren zu können, zum Beispiel verteilte Datenbanksysteme in einem Cluster. Dabei werden wesentliche Komponenten von modernen Datenbanksystemen modelliert: die technische Plattform mit ihren technischen Komponenten, das umgebende Betriebssystem zur Verwaltung von Betriebsmitteln und das Datenbanksystem mit ihren charakteristischen Eigenschaften. Im Falle von verteilten Datenbanksystemen wird zusätzlich das Computernetzwerk mit ihren spezifischen Topologien und Komponenten modelliert, etwa Switche und Router.

Die QPN-Modelle erlauben es, beliebige Datenbankanwendungen zu simulieren. Genauer gesagt, es wird der Datenfluss simuliert, der entsteht, wenn die Datenbankanwendung für eine Kombina-

tion aus technischer Plattform, Betriebssystem und Datenbanksystem ausgeführt wird. Anhand des simulierten Datenflusses können Rückschlüsse auf die Performance gezogen werden. Das kann zum Beispiel die simulierte Antwortzeit sein, also die simulierte Zeit, bis der Datenfluss terminiert. Dadurch, dass auch die technischen Kernkomponenten Teil der Modellierung sind, ist zugleich möglich, anhand des simulierten Datenflusses Aussagen über den Energieverbrauch der Kernkomponenten zu treffen. Im Vergleich zu anderen Modellen stellt das ein Novum dar. Insgesamt ist dadurch die Möglichkeit gegeben, die Energieeffizienz zu berechnen.

Es ist natürlich ratsam, als Datenbankanwendung einen Benchmark zu simulieren. Wie in Kapitel 6 beschrieben, ergibt sich hierdurch der Verteil, die experimenten Ergebnisse der Experimente B bis F dafür zu nutzen, die QPN-Modelle hinreichend zu validieren und ihre Qualität zu überprüfen. Dafür wurde in den Textabschnitten 6.2 und 6.3 detailliert beschrieben, wie die grundlegenden QPN-Modelle arrangiert und angepasst werden müssen, um die genannten Experimente zu simulieren. Als Resultat wurde dann die Genauigkeit berechnet. Sie gibt die prozentualen Differenz zwischen dem experimentell gewonnenen und simulierten Ergebnis einer Metrik an. Das Maximum ist 100 Prozent, womit indiziert wird, dass es keine Differenz gibt. Wie in Tabelle 6.14 auf Seite 205 dargestellt, wurden Genauigkeiten zwischen 63 und 92 Prozent für die Performance und zwischen 78 und 85 Prozent für die Energieeffizienz erreicht. In diesem Zusammenhang muss aber darauf hingewiesen werden, dass diese Genauigkeiten nur Indikatoren für die Qualität sind, da es schlichtweg keine Vergleichsmöglichkeiten gibt. Wie in Textabschnitt 6.4 beschrieben, ergab eine Literaturrecherche, dass keine anderen QPN-Modelle publiziert wurden, die dieselben Ziele verfolgen und annähernd dasselbe leisten wie denjenigen, die im Rahmen dieser Dissertation eingeführt und verwendet wurden.

Es stellt sich natürlich die Frage, aus welchen Gründen eine Simulation eines Benchmarks für ein Datenbanksystems, das auf einer technischen Plattform und Betriebssystem ausgeführt wird, Vorteile bringt. Der Hauptvorteil ist, dass es dadurch möglich ist, diesen Benchmark auszuführen ohne aufwendig in Zeit und technischen Komponenten zu investieren. Erstgenanntes bedeutet, dass es weniger Zeit bedarf, einen Benchmark zu simulieren anstatt ihn real auszuführen. Die Untersuchungsergebnisse dazu aus Textabschnitt 6.4 zeigen, dass die Simulationen bis zu 8.000 mal schneller die Benchmarkmetriken evaluieren können, wenn man sie mit den realen Experimenten vergleicht. Das ist eine enorme Zeitersparnis. Zweitgenanntes bedeutet, dass man eine Vielzahl von technischen Komponenten simulieren kann. Verglichen zu realen Experimenten entfallen dabei die Investitionskosten für diese Komponenten und der zeitliche Aufwand, den Versuchsaufbau mit jeder Komponente modifizieren zu müssen. Ein weiterer Vorteil ergibt sich dadurch, dass die QPN-Modelle fundamental auf *Petri*-Netzen basieren. Damit stehen weiterführende, formale Möglichkeiten offen, wie sie zum Beispiel in [PW08], [BK02a] und [JK07] beschrieben sind. Dazu gehören quantitative und qualitative Analysen der Modelle, zum Beispiel Erreichbarkeits-, Entscheidbarkeits- und Berechenbarkeitsanalysen.

Zusammengefasst bedeutet es, dass durch die QPN-Modelle, die im Rahmen dieser Dissertation eingeführt und erläutert wurden, eine praktikable Alternative geschaffen wurde, Datenbankwendungen in Form von Benchmarks für ein gegebenes Datenbanksystem zu simulieren, wobei alle wesentlichen Einflussfaktoren auf die Benchmarkmetriken mitsimuliert werden. Das Hauptziel dabei ist, die Investitionen in Zeit und technischen Komponenten bedeutend zu reduzieren. Wie erwähnt, stellen Datenbankbenchmarks ein anerkanntes und weithin genutztes Mittel dar, Aspekte wie die Performance, den Energieverbrauch und schlussendlich die Energieeffizienz von Datenbanksystemen zu evaluieren und zu vergleichen. Dabei schließen sich experimentell durchgeführte und simulierte Datenbankbenchmarks nicht aus; ihre Ergebnisse sind direkt vergleichbar. Diese Vergleichbarkeit im allgemeinen erlaubt es, die drei genannten Aspekte in existierenden Datenbanksystemen zu optimieren oder das Skalierungsverhalten zu ermitteln. Gleichzeitig

werden auch andere, darauf aufbauende Anwendungsszenarien möglich, zum Beispiel die Ressourcenplanung oder die Entscheidungsfindung. Beide werden im nachfolgenden Textabschnitt näher beschrieben.

Es soll auch angemerkt werden, dass die eingeführten QPN-Modelle zur Evaluation der Quantität und der Qualität dafür genutzt wurden, zwei etablierte und anerkannte Datenbanksysteme und drei Datenbanksystem-Benchmarks zu simulieren. Hier ist es natürlich wünschenswert, dass zusätzliche Datenbanksysteme und Benchmarks sowie technische Komponenten modelliert werden. Das würde die Glaubwürdigkeit, Plausibilität und Akzeptanz der QPN-Modelle steigern. Das ist insofern realistisch, da innerhalb dieser Dissertation detaillierte Vorgehensweisen und Beispiele erläutert wurden, um dieses Ziel zu ereichen.

Mögliche Weiterentwicklungen

In den Textabschnitten 4.1 und 4.2 wurden die Ergebnisse von Experimenten beschrieben, die das Ziel hatten, die drei Metriken Performance, Energieverbrauch und -effizienz zu evaluieren. Diese Experimente wurden dann in Kapitel 6 modelliert und die experimentellen Ergebnisse genutzt, um die Modelle zu validieren und die Genauigkeiten zu berechnen. Letzteres dient als entscheidende Grundlage zur Bemessung der Qualität der Modelle, zum Beispiel den Realitätsgrad der Simulationen. Eine genauere Betrachtung der Experimente zeigt allerdings, dass nicht die gesamte Palette an Datenbanksystemen, technischen Plattformen und Benchmarks experimentell untersucht beziehungsweise modelliert wurde. Das bedeutet insbesondere

- dass es durch die Experimente B bis F möglich war, jeweils ein QPN-Modell für zentral betriebene und verteilte Datenbanksysteme zu erarbeiten, zu simulieren und zu validieren. Nach Textabschnitt 2.4 sind neben diesen Datenbanksystemen auch *In-Memory*-Datenbanksysteme von Bedeutung. Aussichtsreiche Kandidaten dafür sind *SAP HANA* und *ExaSol*[1]. Eine mögliche Weiterentwicklung auf Basis dieser Dissertation wäre demnach die in Kapitel 5 beschriebene Vorgehensweise. Das bedeutet, dass zunächst diese Datenbanksysteme anhand von Benchmarks experimentell evaluiert werden. Aufgrund der experimentellen Ergebnisse kann dann eine Modellierung ihrer Architektur erfolgen. Die entstandenen Modelle können dann für Simulationen genutzt werden.

- dass es durch die oben genannten Experimente möglich wurde, die Charakteristiken von technischen Komponenten zu modellieren. Dazu zählen unter anderem Modelle für RAID-Systeme, Komponenten für Computernetzwerke (Switche) und CPU-Architekturen. Eine zusätzliche Weiterentwicklung könnte die Modellierung anderer technischer Komponenten beinhalten. Beispiele wären andere Computernetzwerktopologien und -techniken, etwa *Fibre Channel* oder *ATM*, aber auch spezielle Komponenten zur Beschleunigung von mathematischen Berechnungen, etwa Grafikkarten.

Der Vorteil dieser Weiterentwicklungen wäre es, dass damit ein gewisser Grundstock an Modellen entsteht, die beliebig miteinander kombiniert und hierarchisch aufgebaut werden können. Damit wäre es möglich, fast jede beliebige Kombination aus einem Datenbanksystem, einem Betriebssystem und der technischen Plattform zu modellieren. Solche Modelle ließen sich für

[1] Hierbei muss angemerkt werden, dass es im Rahmen dieser Dissertation auch Bestrebungen gab, diese Datenbanksysteme zu nutzen. Allerdings verhinderten Lizenzbestimmungen den Betrieb auf einem eigenen Server zum Zwecke der Durchführung von initialen Benchmarks.

Simulationen nutzen. Wie bereits eingangs beschrieben, können Simulationen dieser Modelle dabei helfen, Investitionen in Zeit und Hardware bedeutend zu senken.

Der beschriebende Grundstock an Modellen sowie darauf beruhende Simulationsergebnisse können dann in der Entwicklung eines Assistenzsystems münden, dass die Entscheidungsfindung sowie die Ressourcenplanung unterstützt. Dazu kann sich ein Assistenzsystem gedacht werden, dass drei Eingabeparameter zulässt: die Auswahl des Datenbanksystems, eine technische Plattform (bestehend aus den technischen Kernkomponenten und dem Betriebssysem) und einem Anwendungsfall (bestehend aus der Datenmenge und dessen Datenmodell). Sofern zwei dieser drei Parameter als konstant verbleiben, kann der verbleibende Parameter variiert werden. Eine mögliche Visualisierung dieses Assistenzsystems ist in Abbildung 7.1 dargestellt.

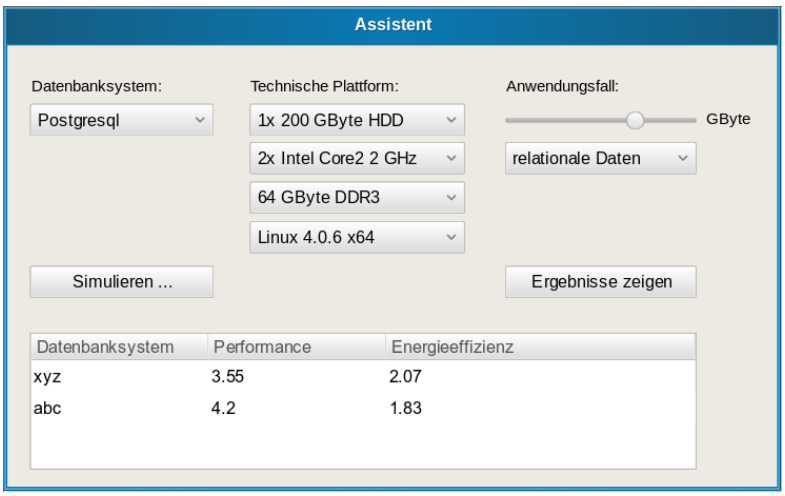


Abbildung 7.1: Mögliches Aussehen des Assistenzsystems

Dadurch sind drei Einsatzszenarien realisierbar:

1. **Entscheidungsfindung** Hierbei werden die beiden Parameter technische Plattform und Anwendungsfall fixiert. Demzufolge soll das Assistenzsystem das geeigneteste Datenbanksystem ermitteln, um den Anwendungsfall optimal zu unterstützen. Das Optimierungsziel kann dabei die Performance, die Energieeffizienz oder eine andere Metrik sein, etwa die Investitionskosten oder bauliche Vorkehrungen.
2. **Ressourcenplanung** Es werden die beiden Parameter Anwendungsfall und Datenbanksystem fixiert. Der verbleibende Parameter technische Plattform wird variiert. Somit soll die technische Plattform durch das Assistenzsystem dargestellt werden, die den gegebenen Anwendungsfall optimal unterstützt.
3. **Skalierungsverhalten** Bei diesem Szenario werden die beiden Parameter technische Plattform und Datenbanksystem fixiert. Dadurch wird der Parameter Anwendungsfall variiert. Das Assistenzsystem soll also zum Beispiel darstellen, wie das Datenbanksystem auf der gegebenen technischen Plattform hinsichtlich der Datenmenge skaliert.

Ein solches Assistenzsystem wäre für Betreiber von Datenverarbeitungszentren sowie für Anbieter und Entwickler von Datenbanksystem von Vorteil. Durch Ermittlung der optimalen Kombination der drei Eingabeparameter hinsichtlich eines Anwendungsfalls und mit dem Ziel, die

Energieeffizienz zu maximieren, kann die höchstmögliche Performance bei gleichzeitig niedrigstmöglichem Energieverbrauch ermittelt werden. Das stellt eine Möglichkeit dar, den Energieverbrauch zu senken ohne dabei die Performance zu vernachlässigen. Für Betreiber von Datenverarbeitungszentren zum Beispiel ist damit die Möglichkeit geschaffen, durch Ressourcenplanung die Kosten für Energie zu senken.

Literaturverzeichnis

[Aba12] ABADI, Daniel: Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story. In: *Computer* 45 (2012), Nr. 2, S. 37–42. http://dx.doi.org/10.1109/MC.2012.33. – DOI 10.1109/MC.2012.33. – ISBN 0018–9162 VO – 45

[AG08] ANGLES, Renzo ; GUTIERREZ, Claudio: Survey of graph database models. In: *ACM Computing Surveys* 40 (2008), Nr. 1, S. 1–39. http://dx.doi.org/10.1145/1322432.1322433. – DOI 10.1145/1322432.1322433. – ISBN 0360–0300

[AGMG+09] AGRAWAL, Rakesh ; GARCIA-MOLINA, Hector ; GEHRKE, Johannes ; GRUENWALD, Le ; HAAS, Laura M. ; HALEVY, Alon Y. ; HELLERSTEIN, Joseph M. ; IOANNIDIS, Yannis E. ; KORTH, Hank F. ; KOSSMANN, Donald ; MADDEN, Samuel ; AILAMAKI, Anastasia ; MAGOULAS, Roger ; OOI, Beng C. ; O'REILLY, Tim ; RAMAKRISHNAN, Raghu ; SARAWAGI, Sunita ; STONEBRAKER, Michael ; SZALAY, Alexander S. ; WEIKUM, Gerhard ; BERNSTEIN, Philip a. ; BREWER, Eric a. ; CAREY, Michael J. ; CHAUDHURI, Surajit ; DOAN, Anhai ; FLORESCU, Daniela ; FRANKLIN, Michael J.: The Claremont report on database research. In: *Communications of the ACM* 52 (2009), Nr. 6, S. 56. http://dx.doi.org/10.1145/1516046.1516062. – DOI 10.1145/1516046.1516062. – ISBN 00010782

[Ahl15] AHLERS, Ernst: Leistungsaufnahme des PCs richtig messen. In: *c't* 27 (2015), S. 174—-177

[Amd67] AMDAHL, Gene M.: Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In: *AFIPS Spring Joint Computer Conference, 1967*. Bd. 30, 1967. – ISBN 1558605398, S. 483–485

[APBC13] ARMSTRONG, Timothy G. ; PONNEKANTI, Vamsi ; BORTHAKUR, Dhruba ; CALLAGHAN, Mark: LinkBench : a Database Benchmark Based on the Facebook Social Graph. In: *SIGMOD '13 Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013. – ISBN 9781450320375, S. 1185 – 1196

[AS95] AHMAD, I. ; SALEH, K.: Specification and verification of cache coherence protocols using Petri nets. In: *International Journal of Electronics* 78 (1995), Nr. 5, S. 841–854. http://dx.doi.org/10.1080/00207219508926212. – DOI 10.1080/00207219508926212. – ISSN 0020–7217

[AS11] ARUN, R. ; SRIKANT, Y. N.: *Petri-Net based Performance Modeling for Effective DVFS for Multithreaded Programs Petri-Net based Performance Modeling for Effective DVFS for Multithreaded Programs*. 2011

[Axt01] AXTELL, Robert L.: Zipf distribution of U.S. firm sizes. In: *Science (New York, N.Y.)* 293 (2001), sep, Nr. 5536, S. 1818–20. http://dx.doi.org/10.1126/science.1062081. – DOI 10.1126/science.1062081. – ISSN 0036–8075

[BBK94] BAUSE, Falko ; BUCHHOLZ, Peter ; KEMPER, Peter: Hierarchically combined queueing Petri nets. Version: 1994. http://dx.doi.org/10.1007/BFb0033546. In:

11th International Conference on Analysis and Optimization of Systems Discrete Event Systems. London : Springer-Verlag, 1994. – DOI 10.1007/BFb0033546, S. 176–182

[BBN+13] BARU, Chaitanya ; BHANDARKAR, Milind ; NAMBIAR, Raghunath ; POESS, Meikel ; RABL, Tilmann: Benchmarking Big Data Systems and the BigData Top100 List. In: *Big Data* 1 (2013), Nr. 1, S. 60–64. http://dx.doi.org/10.1089/big.2013.1509. – DOI 10.1089/big.2013.1509. – ISSN 2167–6461

[Ben11] BENZ, Benjamin: Auf den Zahn gefühlt - Leistungsaufnahme von PC-Komponenten im Detail. In: *c't* 11 (2011), S. 136–143

[BFI94] BENNET, Kristin ; FERRIS, Michael C. ; IODANNIDIS, Yannis E.: A Genetic Algorithm for Database Query Optimization. In: *In Proceedings of the fourth International Conference on Genetic Algorithms*, 1994. – ISBN 0780318994, S. 400—-407

[BHCC07] BARROSO, Luiz A. ; HÖLZLE, Urs ; CASE, The ; COMPUTING, Energy-proportional: The Case for Energy-Proportional Computing. In: *Computer* 40 (2007), dec, Nr. December, S. 33–37. http://dx.doi.org/10.1109/MC.2007.443. – DOI 10.1109/MC.2007.443. – ISBN 0018–9162

[BK02a] BAUSE, Falko ; KITZINGER, Peter S.: *Stochastic Petri nets: An introduction to the theory*. 2nd edition. Friedrich Vieweg & Sohn Verlagsgesellschaft mbH, 2002. – 224 S. – ISBN 3528155353

[BK02b] BONTEMPI, G ; KRUIJTZER, W: A Data Analysis Method for Software Performance Prediction. In: *Proceedings of the conference on Design, automation and test in Europe*, ACM, 2002. – ISSN 1530–1591

[BKF14] BUCHHOLZ, Peter ; KRIEGE, Jan ; FELKO, Iryna: Phase-Type Distributions. Version: 1st editio, 2014. http://dx.doi.org/10.1007/978-3-319-06674-5. In: *Input Modeling with Phase-Type Distributions and Markov Models*. 1st editio. Springer International Publishing, 2014 (0). – DOI 10.1007/978–3–319–06674–5. – ISBN 978–3–319–06673–8, S. 5–29

[BPR88] BLAHA, Michael R. ; PREMERLANI, William J. ; RUMBALIGH, James E.: Relational Database Design using an Object-Oriented Methodology. In: *Communications of the ACM* 31 (1988), Nr. 4, S. 414–427

[BPSM+08] BRAY, Tim ; PAOLI, Jean ; SPERBERG-MCQUEEN, C.M. ; MALER, Eve ; YERGEAU, Francois: *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. https://www.w3.org/TR/xml/. Version: 2008

[Bre12] BREWER, E.: CAP twelve years later: How the "rules"have changed. In: *Computer* 45 (2012), feb, Nr. 2, S. 23–29. http://dx.doi.org/10.1109/MC.2012.37. – DOI 10.1109/MC.2012.37. – ISSN 0018–9162

[BS10] BAKKUM, Peter ; SKADRON, Kevin: Accelerating SQL database operations on a GPU with CUDA. In: *Proceedings of the 3rd Workshop on General- ...*, 2010. – ISBN 9781605589350, S. 94–103

[BSMM99] BRONSTEIN, I.N. ; SEMENDJAEV, K.A. ; MUSIOL, H. ; MÜHLIG, G.: *Taschenbuch der Mathematik*. 4. Auflage. Frankfurt am Main, Thun : Verlag Harri Deutsch, 1999. – 1151 S. – ISBN 3–8171–2004–4

[BYF+09] BAKHODA, Ali ; YUAN, George L. ; FUNG, Wilson W L. ; WONG, Henry ; AAMODT, T. M.: Analyzing CUDA workloads using a detailed GPU simulator. In: *ISPASS 2009 - International Symposium on Performance Analysis of Systems and Software* (2009), S. 163–174. http://dx.doi.org/10.1109/ISPASS.2009.4919648. – DOI 10.1109/ISPASS.2009.4919648. – ISBN 9781424441846

[Cas16] CASALE, Giuliano: Performance Engineering for In-Memory Databases. In: *Companion Publication for ACM/SPEC on International Conference on Performance Engineering - ICPE '16 Companion*. New York, New York, USA : ACM Press, 2016. – ISBN 9781450341479, S. 13–13

[CGK10] CHEN, Yanpei ; GANAPATHI, Archana ; KATZ, Randy H.: To compress or not to compress - compute vs. IO tradeoffs for mapreduce energy efficiency. In: *Proceedings of the first ACM SIGCOMM workshop on Green networking - Green Networking '10*, 2010. – ISBN 9781450301961, S. 23–28

[Che76] CHEN, PPS: The entity-relationship model—toward a unified view of data. In: *ACM Transactions on Database Systems (TODS)* 1 (1976), Nr. 1, S. 9–36. http://dx.doi.org/10.1145/320434.320440. – DOI 10.1145/320434.320440. – ISBN 0471164984

[CII+10] CHUN, Byung-Gon ; IANNACCONE, Gianluca ; IANNACCONE, Giuseppe ; KATZ, Randy ; LEE, Gunho ; NICCOLINI, Luca: An energy case for hybrid datacenters. In: *ACM SIGOPS Operating Systems Review* Bd. 44, 2010. – ISSN 01635980, S. 76

[Cod70] CODD, Edgar F.: A Relational Model of Data for Large Shared Data Banks. In: *Communications of the ACM* 13 (1970), Nr. 6, S. 377–387. – ISSN 0724–6811

[COK13] COULDEN, David ; OSMAN, Rasha ; KNOTTENBELT, William J.: Performance modelling of database contention using queueing petri nets. In: *Proceedings of the ACM/SPEC international conference on International conference on performance engineering - ICPE '13*, 2013. – ISBN 9781450316361, S. 331

[Col15] COLGAN, Maria (.: Oracle Database In-Memory / Oracle. 2015 (July). – Forschungsbericht

[CP15] CALERO, Coral (Hrsg.) ; PIATTINI, Mario (Hrsg.): *Green in Software Engineering*. Cham : Springer International Publishing, 2015. http://dx.doi.org/10.1007/978-3-319-08581-4. http://dx.doi.org/10.1007/978-3-319-08581-4. – ISBN 978–3–319–08580–7

[CPH10] CHANG, Xin ; PANG, Huanli ; HU, Liang: Distributed computer network model base on petri nets. In: *2010 International Conference on Computer, Mechatronics, Control and Electronic Engineering* Bd. 1, IEEE, aug 2010. – ISBN 978–1–4244–7957–3, S. 200–203

[CST+10] COOPER, Brian F. ; SILBERSTEIN, Adam ; TAM, Erwin ; RAMAKRISHNAN, Raghu ; SEARS, Russell: Benchmarking cloud serving systems with YCSB. In: *Proceedings of the 1st ACM symposium on Cloud computing* ACM, 2010, S. 143–154

[DaC09] The GREEN-NET framework: Energy efficiency in large scale distributed systems. In: *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on* (2009), S. 1–8. http://dx.doi.org/10.1109/IPDPS.2009.5160975. – DOI 10.1109/IPDPS.2009.5160975. ISBN 1530–2075

[Dat] DATASTAX: *Understanding the architecture*. https://www.datastax.com/documentation/cassandra/2.0/cassandra/architecture/architectureTOC.html

[DBCrW92] DIETRICH, S. W. ; BROWN, M. ; CORTES-RELLO, E. ; WUNDERLIN, S.: A practitioner's introduction to database performance benchmarks and measurements. In: *Computer Journal* 35 (1992), Nr. 4, S. 322–331. http://dx.doi.org/10.1093/comjnl/35.4.322. – DOI 10.1093/comjnl/35.4.322. – ISSN 00104620

[DCPU11] DUGGAN, Jennie ; CETINTEMEL, Ugur ; PAPAEMMANOUIL, Olga ; UPFAL, Eli: Performance prediction for concurrent database workloads. In: *Proceedings of the 2011 international conference on Management of data - SIGMOD '11*. New York, New York, USA : ACM Press, 2011. – ISBN 9781450306614, S. 337

[Dev01] DEVARAKONDA, Ramakanth S.: Object-relational Database Systems - the Road Ahead. In: *Crossroads* 7 (2001), Nr. 3, S. 15–18. http://dx.doi.org/10.1145/367884.367895. – DOI 10.1145/367884.367895. – ISBN 0749236914

[DG04] DEAN, Jeffrey ; GHEMAWAT, Sanjay: MapReduce: Simplied Data Processing on Large Clusters. In: *Proceedings of 6th Symposium on Operating Systems Design and Implementation*, 2004. – ISBN 9781595936868, S. 137–149

[DGG16] DE MAURO, Andrea ; GRECO, Marco ; GRIMALDI, Michele: A formal definition of Big Data based on its essential features. In: *Library Review* 65 (2016), apr, Nr. 3, S. 122–135. http://dx.doi.org/10.1108/LR-06-2015-0061. – DOI 10.1108/LR–06–2015–0061. – ISSN 0024–2535

[DHJ+07] DECANDIA, Giuseppe ; HASTORUN, Deniz ; JAMPANI, Madan ; KAKULAPATI, Gunavardhan ; LAKSHMAN, Avinash ; PILCHIN, Alex ; SIVASUBRAMANIAN, Swaminathan ; VOSSHALL, Peter ; VOGELS, Werner: Dynamo: Amazon's Highly Available Key-value Store. In: *Proceedings of the Symposium on Operating Systems Principles*, 2007. – ISBN 9781595935915, S. 205–220

[DKL+13] DIMITROV, Martin ; KUMAR, Karthik ; LU, Patrick ; VISWANATHAN, Vish ; WILLHALM, Thomas: Memory system characterization of big data workloads. In: *IEEE International Conference on Big Data*, IEEE, 2013. – ISBN 9781479912926, S. 15–22

[DKR12] DELLKRANTZ, Manfred ; KIHL, Maria ; ROBERTSSON, Anders: Performance Modeling and Analysis of a Database Server with Write-Heavy Workload. In: *Service-Oriented and Cloud Computing*. Springer, 2012, S. 184–191

[DL07] DONG, Hongbin ; LIANG, Yiwen: Genetic algorithms for large join query optimization. In: *Proceedings of the 9th annual conference on Genetic and evolutionary computation - GECCO '07*, 2007. – ISBN 9781595936974, S. 1211

[DM02] DOLAN, Elizabeth D. ; MORÉ, Jorge J.: Benchmarking Optimization Software with Performance Profiles. In: *Mathematical Programming* 91 (2002), Nr. 2, S. 201–213

[Dor14] DORENBURG, Jürgen: *Kühlung von Rechenzentren*. Karlsruhe : cci Dialog GmbH, 2014. – 72 S. – ISBN 9783922420316

[DSK+13] DEDE, Elif ; SENDIR, Bedri ; KUZLU, Pinar ; HARTOG, Jessica ; GOVINDARAJU, Madhusudhan: An Evaluation of Cassandra for Hadoop. In: *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on* (2013), S. 494–501

[DV13] DAN MIRONESCU, Ion ; VINTAN, Lucian: Performance prediction for parallel applications running on HPC architectures through Petri net modelling and simulation. In: *2013 IEEE 9th International Conference on Intelligent Computer Communication and Processing (ICCP)* (2013), Nr. April 2016, S. 267–270. http://dx.doi.org/10.1109/ICCP.2013.6646119. – DOI 10.1109/ICCP.2013.6646119. ISBN 978–1–4799–1494–4

[EN09] ELMASRI, Ramez A. ; NAVATHE, Shamkant B.: *Grundlagen von Datenbanksystemen*. 3. Auflage. Pearson Studium, 2009. – ISBN 386894012X

[FFA+12] FERDMAN, Michael ; FALSAFI, Babak ; ADILEH, Almutaz ; KOCBERBER, Onur ; VOLOS, Stavros ; ALISAFAEE, Mohammad ; JEVDJIC, Djordje ; KAYNAK, Cansu ; POPESCU, Adrian D. ; AILAMAKI, Anastasia: Clearing the clouds - A Study of Emerging Scale-out Workloads on Modern Hardware. In: *ACM SIGARCH Computer Architecture News* 40 (2012), apr, Nr. 1, S. 37. http://dx.doi.org/10.1145/2189750.2150982. – DOI 10.1145/2189750.2150982. – ISBN 9781450307598

[FSV05] FINK, Andreas ; SCHNEIDEREIT, Gabriele ; VOSS, Stefan: *Grundlagen der Wirtschaftsinformatik*. 5. Auflage. Oldenbourg Wissenschaftsverlag, 2005. – 2–6; 178–179 S.

[Gar09] GARCIA, Daniel F.: Performance Modeling and Simulation of Database Servers. In: *The Online Journal on Electronics and Electrical Engineering (OJEEE)* 2 (2009), Nr. 1, S. 183–188

[GC15] GELENBE, Erol ; CASEAU, Yves: The impact of information technology on energy consumption and carbon emissions. In: *Ubiquity* 2015 (2015), jun, Nr. June, S. 1–15. http://dx.doi.org/10.1145/2755977. – DOI 10.1145/2755977. – ISSN 15302180

[Gei14] GEISLER, Frank: *Datenbanken - Grundlagen und Design*. 5. Auflage. Heidelberg : mitp, 2014. – 550 S.

[GGKM06] GOVINDARAJU, Naga ; GRAY, Jim ; KUMAR, Ritesh ; MANOCHA, Dinesh: GPUTeraSort: high performance graphics co-processor sorting for large database management. In: *SIGMOD 2006*, 2006. – ISBN 1–59593–434–0, S. 325 – 336

[GHR11] GONTERMANN, Tobias ; HERMANN, Adam ; ROSSELLI, Marten: *Green IT und Datenbanken*, Goethe Universität, Diplomarbeit, 2011. – 205 S.

[GL02] GILBERT, Seth ; LYNCH, Nancy: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. In: *ACM SIGACT News* 33 (2002), jun, Nr. 2, S. 51. http://dx.doi.org/10.1145/564585.564601. – DOI 10.1145/564585.564601. – ISSN 01635700

[Gla11] GLANZ, James: *Google Details, and Defends, Its Use of Electricity*. New York, nov 2011

[GM16] GREGORY, Adam ; MAJUMDAR, Shikharesh: A Constraint Programming Based Energy Aware Resource Management Middleware for Clouds Processing MapReduce Jobs with Deadlines. In: *Companion Publication for ACM/SPEC on International Conference on Performance Engineering - ICPE '16 Companion*. New York, New York, USA : ACM Press, 2016. – ISBN 9781450341479, S. 15–20

[Gre01] GREINER, Stefan: *Modeling and analysis of operating systems using extended QN techniques and Petri nets*. Erlangen, Diss., 2001

[GRH+13] GHAZAL, Ahmad ; RABL, Tilmann ; HU, Minqing ; RAAB, Francois ; POESS, Meikel ; CROLETTE, Alain ; JACOBSEN, Hans-Arno: Bigbench: Towards an industry standard benchmark for big data analytics. In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013. – ISBN 9781450320375, S. 1197–1208

[HAM06] HARIZOPOULOS, Stavros ; ABADI, Daniel J. ; MADDEN, Samuel: Performance Tradeoffs in Read-Optimized Databases. In: *Proceedings of the VLDB Endowment* (2006), S. 487–498. http://dx.doi.org/10.1145/1142473.1142548. – DOI 10.1145/1142473.1142548. ISBN 1595933859

[HAMS08] HARIZOPOULOS, Stavros ; ABADI, Dj ; MADDEN, Samuel ; STONEBRAKER, Michael: OLTP through the looking glass, and what we found there. In: *Sigmod 2008* pages (2008), S. 981. http://dx.doi.org/10.1145/1376616.1376713. – DOI 10.1145/1376616.1376713. – ISBN 9781605581026

[Hew11] HEWITT, Eben: *Cassandra - the definite guide*. 2nd. O'Reilly, 2011. – 304 S. – ISBN 978–1–449–39041–9

[HHD+10] HUANG, Shengsheng ; HUANG, Jie ; DAI, Jinquan ; XIE, Tao ; HUANG, Bo: The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis. In: *Proc. of the ICDEW - Intl. Conf. on Data Engineering Workshops*, 2010. – ISBN 978–1–4244–6522–4, S. 41–51

[HKZ+11] HINDMAN, Benjamin ; KONWINSKI, Andy ; ZAHARIA, Matei ; ALI GHODSI, Anthony D. J. ; KATZ, Randy ; SHENKER, Scott ; STOICA, Ion: Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In: *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation* Bd. 11. Boston, USA : NSDI, 2011, S. 295–308

[INI+14] IVANOV, Todor ; NIEMANN, Raik ; IZBEROVIC, Sead ; ROSSELLI, Marten ; TOLLE, Karsten ; ZICARI, Roberto V.: Benchmarking DataStax Enterprise/Cassandra with HiBench. In: *arXiv preprint* (2014), nov

[INI+15] IVANOV, Todor ; NIEMANN, Raik ; IZBEROVIC, Sead ; ROSSELLI, Marten ; TOLLE, Karsten ; ZICARI, Roberto: Performance Evaluation of Enterprise Big Data Platforms with HiBench. In: *9th IEEE International Conference on Big Data Science and Engineering*. Helsinki : IEEE, 2015

[Jen81] JENSEN, Kurt: Coloured petri nets and the invariant-method. In: *Theoretical Computer Science* 14 (1981), Nr. 3, S. 317–336. http://dx.doi.org/10.1016/0304-3975(81)90049-9. – DOI 10.1016/0304–3975(81)90049–9. – ISBN 978–3–642–00283–0

[JK07] JENSEN, Kurt ; KRISTENSEN, Lars M.: *Coloured Petri Nets*. 2007. – 213–254 S. http://dx.doi.org/10.1007/b95112. http://dx.doi.org/10.1007/b95112. – ISBN 9783642002830

[KAEN11] KAUSHIK, Rini T. ; ABDELZAHER, Tarek ; EGASHIRA, Ryota ; NAHRSTEDT, Klara: Predictive data and energy management in GreenHDFS. In: *2011 International Green Computing Conference and Workshops*, IEEE, jul 2011. – ISBN 978–1–4577–1222–7, S. 1–9

[KAR+12] KIHL, Maria ; AMANI, Payam ; ROBERTSSON, Anders ; RADU, Gabriela ; DELLKRANTZ, Manfred ; ASPERNÄS, Bertil: Performance Modeling of Database Servers in a Telecommunication Service Management System. In: *The Seventh International Conference on Digital Telecommunications* (2012), Nr. c, S. 124–129. ISBN 9781612081939

[Kau10] KAUSHIK, Rini T.: GreenHDFS : Towards An Energy-Conserving , Storage-Efficient , Hybrid Hadoop Compute Cluster. In: *Proceedings of the USENIX Annual Technical Conference*, UseNix, 2010

[KB03] KOUNEV, Samuel ; BUCHMANN, Alejandro: Performance modelling of distributed e-business applications using Queuing Petri Nets. In: *Performance Analysis of Systems and Software, 2003. ISPASS. 2003 IEEE International Symposium on* IEEE, 2003. – ISBN 0–7803–7756–7, S. 143–155

[KCJ98] KRISTENSEN, Lars M. ; CHRISTENSEN, Søren ; JENSEN, Kurt: The practitioner ' s guide to coloured Petri nets. In: *International Journal* 2 (1998), Nr. 2, S. 98–132. http://dx.doi.org/10.1007/s100099800003. – DOI 10.1007/s100099800003. – ISSN 14332779

[KD09] KOUNEV, Samuel ; DUTZ, Christofer: QPME - A Performance Modeling Tool Based on Queueing Petri Nets. In: *ACM SIGMETRICS Performance Evaluation Review (PER), Special Issue on Tools for Computer Performance Modeling and Reliability Analysis* 36 (2009), S. 46–51

[KE11] KEMPER, Alfons ; EICKLER, Andre: *Datenbanksysteme: eine Einführung*. 8. Auflage. München : Oldenbourg Wissenschaftsverlag, 2011. – 792 S. – ISBN 9783486598346

[KK03] KAASHOEK, M. F. ; KARGER, David: Koorde: A Simple Degree-Optimal Distributed Hash Table. In: *Peer-to-Peer Systems II* 2735 (2003), Nr. 1, S. 98–107. http://dx.doi.org/10.1007/b11823. – DOI 10.1007/b11823. – ISBN 9783540407249

[KKR14] KUHLENKAMP, Jörn ; KLEMS, Markus ; RÖSS, Oliver: Benchmarking Scalability and Elasticity of Distributed Database Systems. In: *Proceedings of the VLDB Endowment* 7 (2014), aug, Nr. 13, S. 1219–1230. http://dx.doi.org/10.14778/2732977.2732995. – DOI 10.14778/2732977.2732995. – ISSN 21508097

[KLL+97] KARGER, David ; LEHMAN, Eric ; LEIGHTON, Tom ; PANIGRAHY, Rina ; LEVINE, Matthew ; LEWIN, Daniel: Consistent hashing and random trees. In: *Proceedings of the 29th ACM symposium on Theory of computing*. New York, New York, USA : ACM Press, 1997. – ISBN 0897918886, S. 654–663

[Kou08] Performance Evaluation: Metrics, Models and Benchmarks. In: KOUNEV, Samuel (Hrsg.) ; GORTON, Ian (Hrsg.) ; SACHS, Kai (Hrsg.): *SPEC International Performance Evaluation Workshop 2008* Bd. 5119. Berlin, Heidelberg : Springer Berlin Heidelberg, 2008 (Lecture Notes in Computer Science). – ISBN 978–3–540–69813–5

[KV08] KORTE, Bernhard ; VYGEN, Jens: *Kombinatorische Optimierung: Theorie und Optimierung*. 4. Auflage. Berlin, Heidelberg : Springer Berlin Heidelberg, 2008. – 681 S. – ISBN 978–3–540–76918–7

[LBE+98] LO, Jack L. ; BARROSO, Luiz A. ; EGGERS, Susan J. ; GHARACHORLOO, Kourosh ; LEVY, Henry M. ; PAREKH, Sujay S.: An Analysis of Database Workload Performance on Simultaneous Multithreaded Processors. In: *Proceedings of the 25th Annual International Symposium on Computer Architecture* 26 (1998), Nr. June, S. 39–50. http://dx.doi.org/10.1145/279361.279367. – DOI 10.1145/279361.279367. – ISBN 0–8186–8491–7

[LH11] LLADÓ, Catalina M. ; HARRISON, Peter G.: A PMIF with petri net building blocks. In: *Proceeding of the second joint WOSP/SIPEW international conference on Performance engineering - ICPE '11* Bd. 36, 2011. – ISBN 9781450305198, S. 103

[LHP+12] LANG, Willis ; HARIZOPOULOS, Stavros ; PATEL, Jignesh M. ; SHAH, Mehul A. ; TSIROGIANNIS, Dimitris: Towards energy-efficient database cluster design. In: *Proceedings of the VLDB Endowment* 5 (2012), jul, Nr. 11, S. 1684–1695. http://dx.doi.org/10.14778/2350229.2350280. – DOI 10.14778/2350229.2350280. – ISSN 21508097

[Lin13] LIN, Jimmy: Mapreduce is Good Enough? If All You Have is a Hammer, Throw Away Everything That's Not a Nail! In: *Big Data* 1 (2013), Nr. 1, S. 28–37. http://dx.doi.org/10.1089/big.2012.1501. – DOI 10.1089/big.2012.1501. – ISBN 2167–6461\r2167–647X

[LK10] LEVERICH, Jacob ; KOZYRAKIS, Christos: On the energy (in)efficiency of Hadoop clusters. In: *ACM SIGOPS Operating Systems Review* Bd. 44, 2010. – ISSN 01635980, S. 61

[LKP11] LANG, Willis ; KANDHAN, Ramakrishnan ; PATEL, Jignesh M.: Rethinking Query Processing for Energy Efficiency: Slowing Down to Win the Race. In: *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 34 (2011), Nr. 1, S. 12

[LP09] LANG, Willis ; PATEL, Jignesh: Towards Eco-friendly Database Management Systems. In: *4th Biennial Conference on Innovative Data Systems Research (CIDR)*. Asilomar, California, USA : CIDR, 2009

[LP10a] LAKSHAM AVINASH ; PRASHANT MALIK: Cassandra: a decentralized structured storage system. In: *ACM SIGOPS Operating Systems Review*, 2010. – ISBN 9781605583969, S. 1–6

[LP10b] LANG, Willis ; PATEL, Jignesh M.: Energy management for MapReduce clusters. In: *Proceedings of the VLDB Endowment* Bd. 3, 2010. – ISSN 2150–8097, S. 129–139

[LZGS84] LAZOWSKA, Edward D. ; ZAHORJAN, John ; GRAHAM, G. S. ; SEVCIK, Kenneth G.: *Quantitative system performance : computer system analysis using queueing network models*. Englewood Cliffs, NJ : Prentice-Hall, 1984. – 417 S. – ISBN 0137469756

[MCE+02] MAGNUSSON, P.S. ; CHRISTENSSON, M. ; ESKILSON, J. ; FORSGREN, D. ; HALLBERG, G. ; HOGBERG, J. ; LARSSON, F. ; MOESTEDT, A. ; WERNER, B.: Simics: A full system simulation platform. In: *Computer* 35 (2002), Nr. 2, S. 50–58. http://dx.doi.org/10.1109/2.982916. – DOI 10.1109/2.982916. – ISSN 00189162

[Nam11] Performance Evaluation, Measurement and Characterization of Complex Systems. In: NAMBIAR, Raghunath (Hrsg.) ; POESS, Meikel (Hrsg.): *Second TPC Technology Conference 2010* Bd. 6417. Berlin, Heidelberg : Springer Berlin Heidelberg, 2011 (Lecture Notes in Computer Science). – ISBN 978–3–642–18205–1

[Neu93] NEUMANN, J. von: First draft of a report on the EDVAC. In: *IEEE Annals of the History of Computing* 15 (1993), Nr. 4, S. 27–75. http://dx.doi.org/10.1109/85.238389. – DOI 10.1109/85.238389. – ISSN 1058–6180

[NI15a] NIEMANN, Raik ; IVANOV, Todor: Evaluating the Energy Efficiency of Data Management Systems. In: *2015 IEEE/ACM 4th International Workshop on Green and Sustainable Software*, IEEE, may 2015. – ISBN 978–1–4673–7049–3, S. 22–28

[NI15b] NIEMANN, Raik ; IVANOV, Todor: Modelling the Performance, Energy Consumption and Efficiency of Data Management Systems. In: W., Cunningham D. (Hrsg.) ; HOFSTEDT, Petra (Hrsg.) ; MEER, Klaus (Hrsg.) ; SCHMITT, Ingo (Hrsg.): *INFORMATIK 2015: Informatik, Energie und Umwelt*. Cottbus : Bonner Köllen Verlag, 2015, S. 1183–1194

[Nie15] NIEMANN, Raik: Evaluating the Performance and Energy Consumption of Distributed Data Management Systems. In: *2015 IEEE 10th International Conference on Global Software Engineering Workshops*, IEEE, 2015. – ISBN 978–1–4799–9874–6, S. 27–34

[Nie16] NIEMANN, Raik: Towards the Prediction of the Performance and Energy Efficiency of Distributed Data Management Systems. In: *Proceedings of the 7th International Conference on Performance Engineering*. Delft : ACM/SPEC, 2016

[NJ00] NICOLA, M. ; JARKE, M.: Performance modeling of distributed and replicated databases. In: *IEEE Transactions on Knowledge and Data Engineering* 12 (2000), Nr. 4, S. 645–672. http://dx.doi.org/10.1109/69.868912. – DOI 10.1109/69.868912. – ISBN 1041–4347

[NKZG13] NIEMANN, Raik ; KORFIATIS, Nikolaos ; ZICARI, Roberto ; GÖBEL, Richard: Evaluating the Energy Efficiency of OLTP Operations. Version: 2013. http://dx.doi.org/10.1007/978-3-642-40511-2. In: CUZZOCREA, Alfredo (Hrsg.) ; KITTL, Christian (Hrsg.) ; SIMOS, Dimitris E. (Hrsg.) ; WEIPPL, Edgar (Hrsg.) ; XU, Lida (Hrsg.): *Availability, Reliability, and Security in Information Systems and HCI* Bd. 8127. Berlin, Heidelberg : Springer, 2013. – DOI 10.1007/978–3–642–40511–2. – ISBN 978–3–642–40510–5, S. 28–43

[NPD+15] NAMBIAR, Raghunath ; POESS, Meikel ; DEY, Akon ; CAO, Paul ; MAGDON-ISMAIL, Tariq ; QI REN, Da ; BOND, Andrew: Introducing TPCx-HS: The First Industry Standard for Benchmarking Big Data Systems. In: NAMBIAR, Raghunath (Hrsg.) ; POESS, Meikel (Hrsg.): *Performance Characterization and Benchmarking*. Springer, 2015, S. 1–12

[NPG15] NIEMANN, Raik ; PFINGST, Udo ; GÖBEL, Richard: Performance Evaluation of netfilter: A Study on the Performance Loss When Using netfilter as a Firewall. In: *ArXiv preprint* (2015)

[NRKR14] NOORSHAMS, Qais ; ROSTAMI, Kiana ; KOUNEV, Samuel ; REUSSNER, Ralf: Modeling of I/O Performance Interference in Virtualized Environments with Queueing Petri Nets. In: *2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*, 2014. – ISBN 978–1–4799–5610–4, S. 331–336

[NW03] NAOR, Moni ; WIEDER, Udi: A simple fault tolerant distributed hash table. In: *Peer-to-Peer Systems II* (2003), S. 1–6. – ISBN 978–3–540–40724–9, 978–3–540–45172–3

[OAW08] OSMAN, Rasha ; AWAN, Irfan ; WOODWARD, Me: Queuing networks for the performance evaluation of database designs. In: *24th UK Performance Engineering Workshop (UKPEW 2008), Dept of Computing, Imperial College London* (2008), S. 172–183

[OOC09] O'NEIL, Pat ; O'NEIL, Betty ; CHEN, Xuedong: *Star Schema Benchmark Revision3*. Boston, 2009

[Öre84] ÖREN, Tuncer I.: Quality Assurance in Modelling and Simulation: A Taxonomy. Version: 1984. http://dx.doi.org/10.1007/978-3-642-82144-8. In: ÖREN, Tuncer I. (Hrsg.) ; ZEIGLER, Bernard P. (Hrsg.) ; ELZAS, Maurice S. (Hrsg.): *Simulation and Model-Based Methodologies: An Integrative View*. Berlin, Heidelberg : Springer Berlin Heidelberg, 1984. – DOI 10.1007/978–3–642–82144–8. – ISBN 978–3–642–82146–2, S. 477–517

[PNV+10] POESS, Meikel ; NAMBIAR, Raghunath O. ; VAID, Kushagra ; STEPHENS JR., John M. ; HUPPLER, Karl R. ; HAINES, Evan: Energy benchmarks: a detailed analysis. In: *1st International Conference on Energy-Efficient Computing and Networking*, 2010. – ISBN 978–145030042–1, S. 131–140

[PPR+09] PAVLO, Andrew ; PAULSON, Erik ; RASIN, Alexander ; ABADI, Daniel J. ; DEWITT, David J. ; MADDEN, Samuel ; STONEBRAKER, Michael: A comparison of approaches to large-scale data analysis. In: *Proceedings of the 35th SIGMOD international conference on Management of data*, 2009. – ISBN 9781605585512, S. 165–178

[PPR+11] PATIL, Swapnil ; POLTE, Milo ; REN, Kai ; TANTISIRIROJ, Wittawat ; XIAO, Lin ; LÓPEZ, Julio ; GIBSON, Garth ; FUCHS, Adam ; RINALDI, Billie: YCSB++: benchmarking and performance debugging advanced features in scalable table stores. In: *Proceedings of the 2nd ACM Symposium on Cloud Computing*. New York, New York, USA : ACM Press, oct 2011. – ISBN 9781450309769, S. 9

[Pri08] PRITCHETT, Dan: Base: an Acid Alternative. In: *Queue* 6 (2008), Nr. 3, S. 48–55. http://dx.doi.org/10.1145/1394127.1394128. – DOI 10.1145/1394127.1394128. – ISSN 15427730

[PW08] PRIESE, Lutz ; WIMMEL, Harro: *Petri-Netze*. 2008. – 1–376 S. http://dx.doi.org/10.1007/978-3-540-76971-2. http://dx.doi.org/10.1007/978-3-540-76971-2. – ISBN 978–3–540–76970–5

[QZ13] QIN, Xiongpai ; ZHOU, Xiaoyun: A survey on benchmarks for big data and some more considerations. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8206 LNCS (2013), S. 619–627. – ISBN 9783642412776

[RGVS+12] RABL, Tilmann ; GÓMEZ-VILLAMOR, Sergio ; SADOGHI, Mohammad ; MUNTÉS-MULERO, Victor ; JACOBSEN, Hans-Arno ; MANKOVSKII, Serge: Solving Big Data Challenges for Enterprise Application Performance Management. In: *Proceedings of the VLDB Endowment* Bd. 5, VLDB Endowment, 2012. – ISSN 21508097, S. 1724–1735

[RHWG95] ROSENBLUM, Mendel ; HERROD, Stephen A. ; WITCHEL, Emmett ; GUPTA, Anoop: Complete Computer System Simulation: The SimOS Approach. In: *IEEE Parallel and Distributed Technology* Bd. 3, 1995. – ISSN 10636552, S. 34–43

[Rie13a] RIESS, Ulrike: BigData bestimmt die IT-Welt. In: *BigData (Sonderbeilage der c't)* (2013), S. 4–9

[Rie13b] RIESS, Ulrike: Die neuen Datenbanken. In: *BigData (Sonderbeilage der c't)* (2013), S. 22–24

[RK14] RYGIELSKI, Piotr ; KOUNEV, Samuel: Data Center Network Throughput Analysis using Queueing Petri Nets. In: *Distributed Computing Systems Workshops (ICDCSW), 2014 IEEE 34th International Conference on* IEEE, 2014, S. 100–105

[RNI+15] ROSSELLI, Marten ; NIEMANN, Raik ; IVANOV, Todor ; TOLLE, Karsten ; ZICARI, Roberto: Benchmarking the Availability and Fault Tolerance of Cassandra. In: *6th SPEC Workshop on Big Data Benchmarking*. Toronto : SPEC, 2015

[Rob02] ROBINSON, Stewart: General concepts of quality for discrete-event simulation. In: *European Journal of Operational Research* 138 (2002), Nr. 1, S. 103–117. http://dx.doi.org/10.1016/S0377-2217(01)00127-8. – DOI 10.1016/S0377–2217(01)00127–8. – ISBN 4424765245

[RRB+15] ROBERTS, Tom ; RICCIO, Karen ; BRADBURY, Danny ; DUTTON, Gail ; KLEYMAN, Bill ; ROBB, Drew: AFCOM's 2015 State-of-the-Data Center Survey / AFCOM. New York, 2015. – Forschungsbericht. – 28 S.

[RWE13] ROBINSON, Ian ; WEBBER, Jim ; EIFREM, Emil: *Graph databases*. Sebastopol. CA : O'Reilly, 2013. – ISBN 978–1–4493–5626–2

[RZK13] RYGIELSKI, Piotr ; ZSCHALER, Steffen ; KOUNEV, Samuel: A meta-model for performance modeling of dynamic virtualized network infrastructures. In: *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*. New York, New York, USA : ACM Press, apr 2013. – ISBN 9781450316361, S. 327

[SD95] SU, Ching-Long ; DESPAIN, Alvin M.: Cache Designs for Energy Efficiency. In: *Proceedings of the 28th Annual Hawaii International Conference on System Sciences* Bd. 00, 1995. – ISBN 0–8186–6930–6, S. 306–315

[SDM+89] SOEDIONO, Budi ; DATTA, Anindya ; MUKHERJEE, Sarit ; KONANA, Prabhudev ; VIGUIER, Igor R. ; BAJAJ, Akhilesh: Multiclass Transaction Scheduling and Overload Management in Firm Real-Time Database Systems. In: *Information Systems* 53 (1989), Nr. 1, S. 160. http://dx.doi.org/10.1016/S0306-4379(96)00003-8. – DOI 10.1016/S0306–4379(96)00003–8. – ISBN 9788578110796

[SEM+14] SCOTT, Grant ; ENGLAND, Matthew ; MELKOWSKI, Kevin ; FIELDS, Zachary ; ANDERSON, Derek T.: GPU-based PostgreSQL extensions for scalable high-throughput pattern matching. In: *Proceedings - International Conference on Pattern Recognition*, 2014. – ISBN 9781479952083, S. 1880–1885

[SH11] SCHALL, D ; HUDLET, V: WattDB: An energy-proportional cluster of wimpy nodes. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2011. – ISBN 978–1–4503–0661–4, S. 1229–1231

[SH13] SCHALL, Daniel ; HÄRDER, Theo: Energy-proportional Query Execution using a Cluster of Wimpy Nodes. In: *Proc. SIGMOD Workshop "Data Management on New Hardware (DaMoN)*, 2013. – ISBN 9781450321969

[SHH10] SCHALL, Daniel ; HUDLET, Volker ; HÄRDER, Theo: Enhancing energy efficiency of database applications using SSDs. In: *Proceedings of the Third C* Conference on Computer Science and Software Engineering - C3S2E '10*. New York, New York, USA : ACM Press, may 2010. – ISBN 9781605589015, S. 1–9

[SLM16] SCHROEDER, Bianca ; LAGISETTY, Raghav ; MERCHANT, Arif: Flash Reliability in Production: The Expected and the Unexpected. In: *14th USENIX Conference on File and Storage Technologies (FAST '16)*, 2016. – ISBN 9781931971287, S. 67–80

[SM08] SANGYEUN CHO ; MELHEM, R.G.: Corollaries to Amdahl's Law for Energy. In: *IEEE Computer Architecture Letters* 7 (2008), jan, Nr. 1, S. 25–28. http://dx.doi.org/10.1109/L-CA.2007.18. – DOI 10.1109/L–CA.2007.18. – ISSN 1556–6056

[SMA+07] STONEBRAKER, Michael ; MADDEN, Samuel ; ABADI, Daniel J. ; HARIZOPOULOS, Stavros ; HACHEM, Nabil ; HELLAND, Pat: The End of an Architectural Era (It's Time for a Complete Rewrite). In: *Vldb* 12 (2007), Nr. 2, S. 1150–1160. http://dx.doi.org/10.1080/13264820701730900. – DOI 10.1080/13264820701730900. – ISBN 9781595936493

[SPÖ+10] SHEN, Heng T. (Hrsg.) ; PEI, Jian (Hrsg.) ; ÖZSU, M. T. (Hrsg.) ; ZOU, Leu (Hrsg.) ; LU, Jiaheng (Hrsg.) ; LING, Tok-Wang (Hrsg.) ; YU, Ge (Hrsg.) ; ZHUNAG, Yi (Hrsg.) ; SHAO, Jie (Hrsg.): *Web-age Information Management*. Springer, 2010. – 274 S. – ISBN 978–3–642–16719–5

[Sta09] STATISTISCHES BUNDESAMT: *Informationsgesellschaft in Deutschland*. Wiesbaden : Statistisches Bundesamt, 2009. – 73 S. – ISBN 9783824608683

[SVSR13] SHUTE, Jeff ; VINGRALEK, R ; SAMWEL, Bart ; RAE, Ian: F1: A distributed SQL database that scales. In: *Proceedings of the ...* Bd. 6, 2013. – ISSN 2150–8097, S. 1068–1079

[SWK+02] SCHMIDT, Albrecht ; WAAS, Florian ; KERSTEN, Martin ; CAREY, Michael J. ; MANOLESCU, Ioana ; BUSSE, Ralph: XMark: A benchmark for XML data management. In: *Proceedings of the 28th international conference on Very Large Data Bases*, 2002. – ISBN 1–55860–869–9, S. 974–985

[TB14] TANENBAUM, Andrew S. ; BOS, Herbert: *Modern operating systems*. 4th revise. Prentice Hall International, 2014. – ISBN 978–1292061429

[THS10] TSIROGIANNIS, Dimitris ; HARIZOPOULOS, Stavros ; SHAH, Mehul a.: Analyzing the Energy Efficiency of a Database Server. In: *the 2010 International Conference*, 2010. – ISBN 9781450300322, S. 231

[Tol06] TOLLE, Karsten: *Semantisches Web und Kontext – Speicherung von und Anfragen auf RDF-Daten unter Berücksichtigung des Kontextes*. Frankfurt, Goethe-Universität, Dissertation, 2006. – 138 S.

[UB93] ULUSOY, Özgür ; BELFORD, Geneva G.: Real-time transaction scheduling in database systems. In: *Information Systems* 8 (1993), S. 559–580

[VBSK09] VASIĆ, Nedeljko ; BARISITS, Martin ; SALZGEBER, Vincent ; KOSTIC, Dejan: Making cluster applications energy-aware. In: *Proceedings of the 1st workshop on Automated control for datacenters and clouds - ACDC '09*, 2009. – ISBN 9781605585857, S. 37

[Vos94] VOSSEN, Gottfried: *Datenmodelle, Datenbanksprachen und Datenbank-Management-Systeme*. 2. Auflage. Bonn : Addison-Wesley, 1994. – 686 S. – ISBN 3–89319–566–1

[WD14] WHITNEY, Josh ; DELFORGE, Pierre: Data Center Efficiency Assessment: Scaling Up Energy Efficiency Across the Data Center Industry : Evaluating Key Drivers and Barriers / Natural Resource Defense Council. 2014 (August). – Forschungsbericht

[WFP07] WOODSIDE, Murray ; FRANKS, Greg ; PETRIU, Dorina C.: The Future of Software Performance Engineering. In: *Future of Software Engineering (FOSE '07)*, 2007. – ISBN 0–7695–2829–5, S. 171–187

[WFXS11] WANG, Jun ; FENG, Ling ; XUE, Wenwei ; SONG, Zhanjiang: A survey on energy-efficient data management. In: *ACM SIGMOD Record* 40 (2011), Nr. 2, S. 17. http://dx.doi.org/10.1145/2034863.2034867. – DOI 10.1145/2034863.2034867. – ISBN 0163–5808

[Wöl13] WÖLTCHE, Adrian: *Analyse indexbasierter und sequentieller Suchverfahren als Basisalgorithmen für In-Memory-Datenbanken*, Hochschule für angewandte Wissenschaften Hof, Master Thesis, 2013. – 98 S.

[XTW10] XU, Zichen ; TU, Yi-Cheng ; WANG, Xiaorui: Exploring power-performance tradeoffs in database systems. In: *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, 2010. – ISBN 978–1–4244–5445–7, S. 485–496

[XYB+13] XIONG, Wen ; YU, Zhibin ; BEI, Zhendong ; ZHAO, Juanjuan ; ZHANG, Fan ; ZOU, Yubin ; BAI, Xue ; LI, Ye ; XU, Chengzhong: A characterization of big data benchmarks. In: *Proceedings - 2013 IEEE International Conference on Big Data, Big Data 2013*, IEEE, 2013. – ISBN 9781479912926, S. 118–125

[YDHP07] YANG, Hung-chih ; DASDAN, Ali ; HSIAO, Ruey-Lung ; PARKER, D. S.: Map-reduce-merge: simplified relational data processing on large clusters. In: *Proceedings of the 2007 ACM SIGMOD international conference on Management of data - SIGMOD '07*, 2007. – ISBN 9781595936868, S. 1029

[YL06] YI, J.J. ; LILJA, D.J.: Simulation of computer architectures: simulators, benchmarks, methodologies, and recommendations. In: *IEEE Transactions on Computers* Bd. 55, IEEE, 2006. – ISSN 0018–9340, S. 268–280

[YM13] YUVENTI, Jumie ; MEHDIZADEH, Roshan: A critical analysis of Power Usage Effectiveness and its use in communicating data center energy consumption. In: *Energy and Buildings* 64 (2013), S. 90–94. http://dx.doi.org/10.1016/j.enbuild.2013.04.015. – DOI 10.1016/j.enbuild.2013.04.015. – ISSN 03787788

[Zib16] ZIBITSKER, Boris: Big Data Applications Performance Assurance. In: *Companion Publication for ACM/SPEC on International Conference on Performance Engineering - ICPE '16 Companion*. New York, New York, USA : ACM Press, 2016. – ISBN 9781450341479, S. 31–31

[ZZ10] ZHENG, Hongzhong ; ZHU, Zhichun: Power and Performance Trade-Offs in Contemporary DRAM System Designs for Multicore Processors. In: *IEEE Transactions on Computers* 59 (2010), aug, Nr. 8, S. 1033–1046. http://dx.doi.org/10.1109/TC.2010.108. – DOI 10.1109/TC.2010.108. – ISSN 0018–9340

A Anhang

A.1 Zusätzliche experimentelle Ergebnisse

A.1.1 Experimentelle Ergebnisse für Experiment B

Die nachfolgende Grafik und Tabelle stellen die experimentellen Ergebnisse für den *TPC-H*-Benchmark aus Experiment B (vgl. Seite 78) dar.

	← *Arbeitsspeichergröße* →		
Verbindungsart	**1 GByte**	**2.5 GByte**	**5 GByte**
1 GByte Datenbankgröße			
Einzelverbindung	$6.285506 \cdot 10^{-10}$	$6.161519 \cdot 10^{-10}$	$6.671073 \cdot 10^{-10}$
Mehrfachverbindung	$6.913648 \cdot 10^{-10}$	$6.464351 \cdot 10^{-10}$	$7.124907 \cdot 10^{-10}$
5 GByte Datenbankgröße			
Einzelverbindung	$2.609586 \cdot 10^{-12}$	$1.467117 \cdot 10^{-12}$	$1.501824 \cdot 10^{-12}$
Mehrfachverbindung	$2.866488 \cdot 10^{-12}$	$1.785889 \cdot 10^{-12}$	$1.449710 \cdot 10^{-12}$
10 GByte Datenbankgröße			
Einzelverbindung	$2.568826 \cdot 10^{-13}$	$2.122201 \cdot 10^{-13}$	$8.898718 \cdot 10^{-14}$
Mehrfachverbindung	$2.681026 \cdot 10^{-13}$	$2.208022 \cdot 10^{-13}$	$1.026758 \cdot 10^{-13}$

Tabelle A.1: Mittlere Energieeffizienz des *TPC-H*-Benchmarks

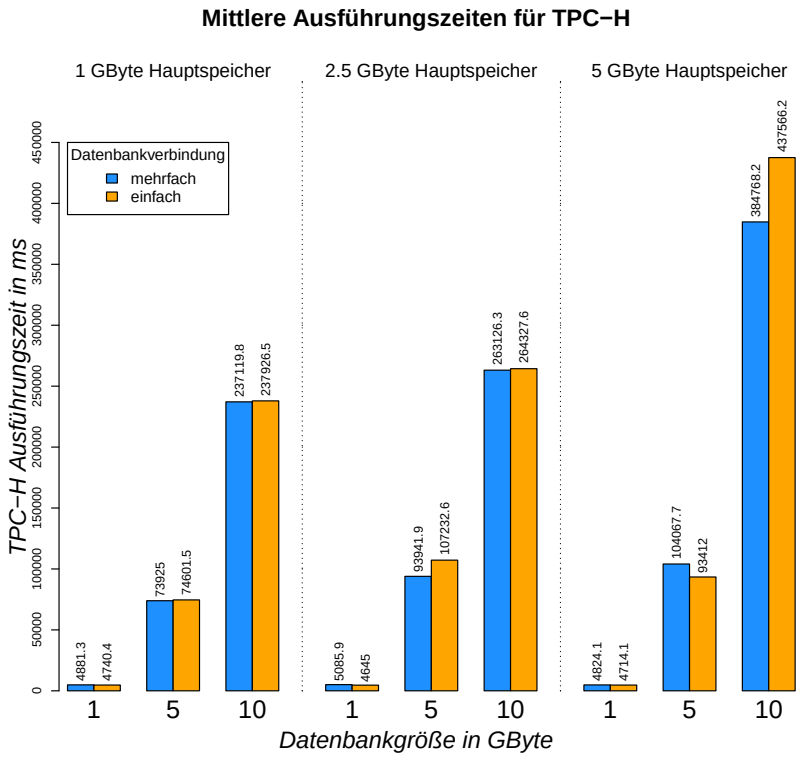


Abbildung A.1: Mittlere Antwortzeiten des *TPC-H*-Benchmarks

A.1.2 Experimentelle Ergebnisse für Experiment C

Die nachfolgende Grafik und Tabelle stellen die experimentellen Ergebnisse für den *StarSchema*-Benchmark aus Experiment C (vgl. Seite 84) dar.

	← *Arbeitsspeichergröße* →		
Verbindungsart	**1 GByte**	**2.5 GByte**	**5 GByte**
1 GByte Datenbankgröße			
Einzelverbindung	$2.064524 \cdot 10^{-8}$	$2.046993 \cdot 10^{-8}$	$2.243907 \cdot 10^{-8}$
Mehrfachverbindung	$1.996951 \cdot 10^{-8}$	$2.032002 \cdot 10^{-8}$	$1.898967 \cdot 10^{-8}$
5 GByte Datenbankgröße			
Einzelverbindung	$5.980822 \cdot 10^{-10}$	$6.130906 \cdot 10^{-10}$	$5.700113 \cdot 10^{-10}$
Mehrfachverbindung	$7.150536 \cdot 10^{-10}$	$7.116695 \cdot 10^{-10}$	$6.882578 \cdot 10^{-10}$
10 GByte Datenbankgröße			
Einzelverbindung	$1.756796 \cdot 10^{-11}$	$1.754061 \cdot 10^{-11}$	$1.762180 \cdot 10^{-11}$
Mehrfachverbindung	$1.925416 \cdot 10^{-11}$	$1.970282 \cdot 10^{-11}$	$1.976115 \cdot 10^{-11}$

Tabelle A.2: Mittlere Energieeffizienz des *StarSchema*-Benchmarks

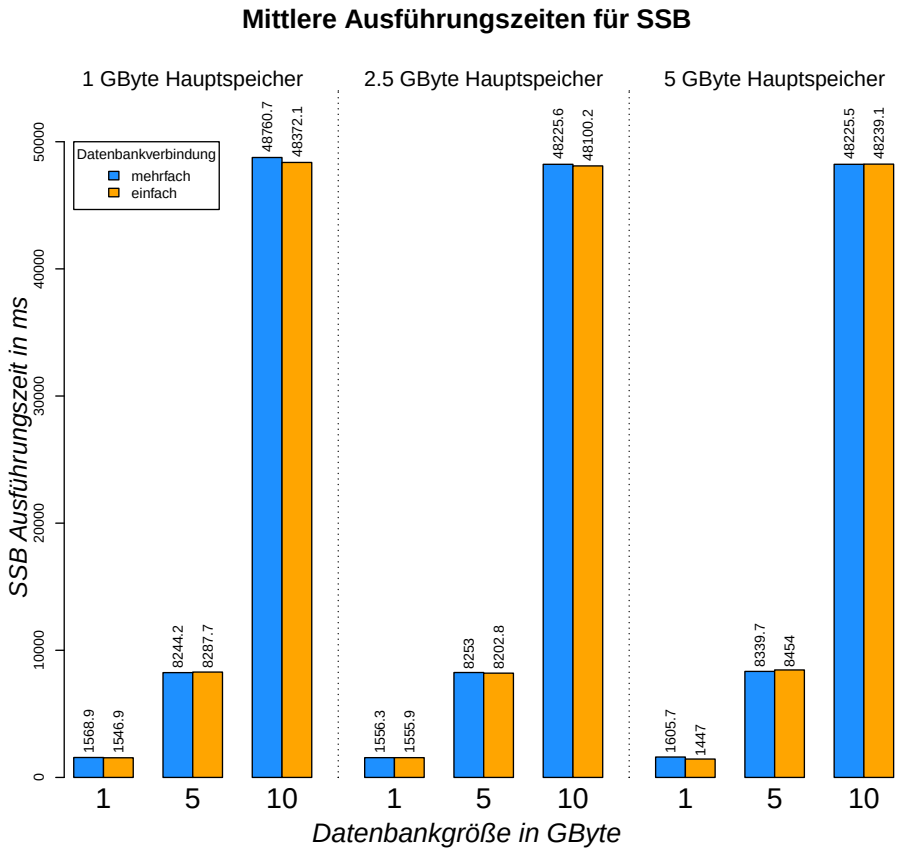
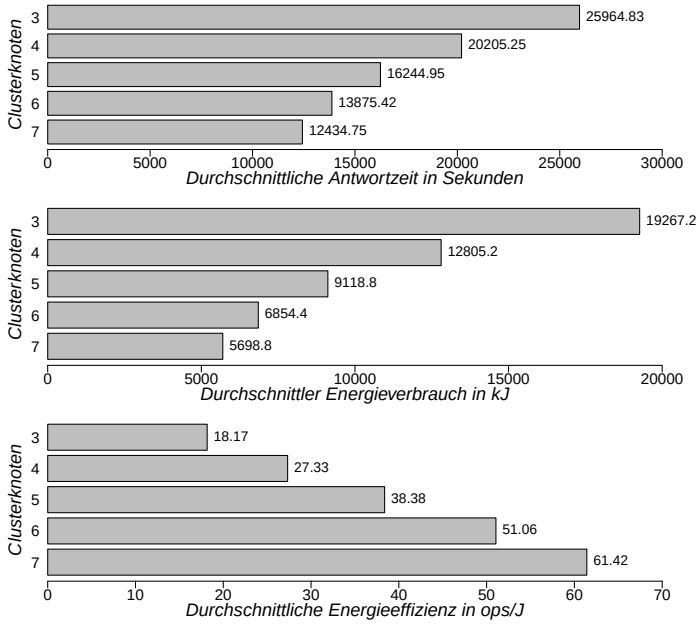


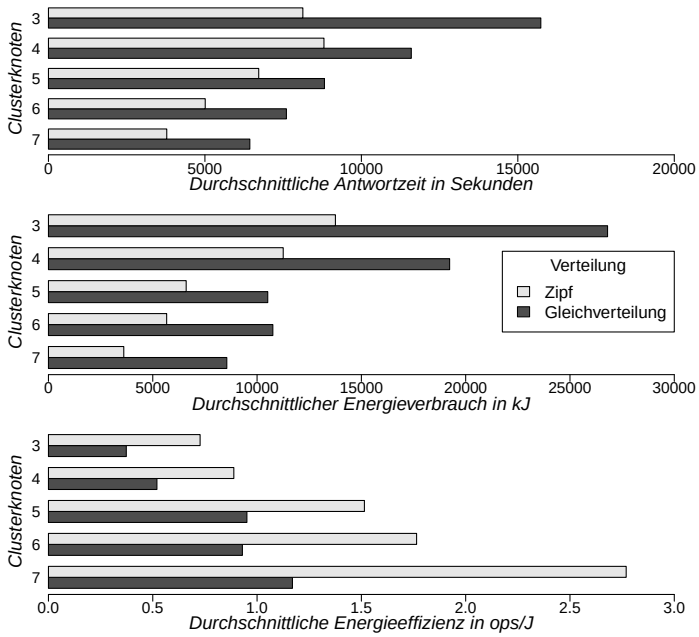
Abbildung A.2: Mittlere Antwortzeiten des *StarSchema*-Benchmarks

A.1.3 Experimentelle Ergebnisse für Experiment E

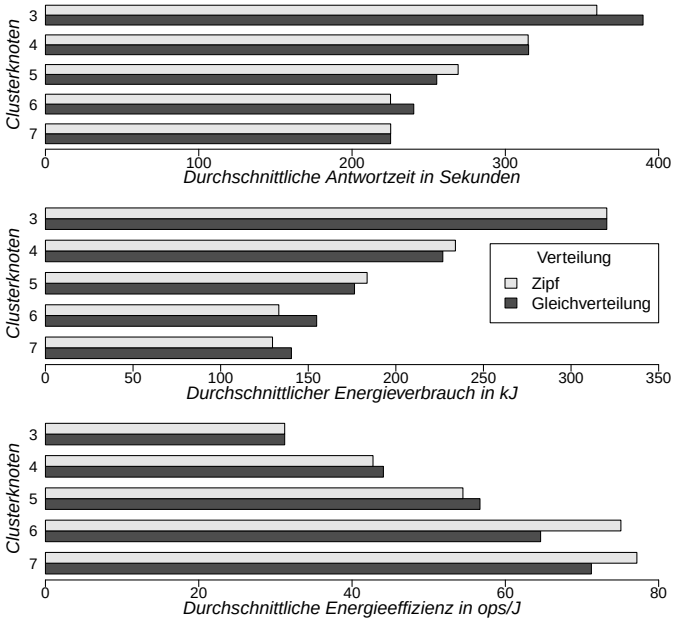
Die nachfolgenden Grafiken stellen die experimentellen Ergebnisse für Experiment E (vgl. Seite 98) für den Anwendungsfall A (fixe Datenmenge) dar.



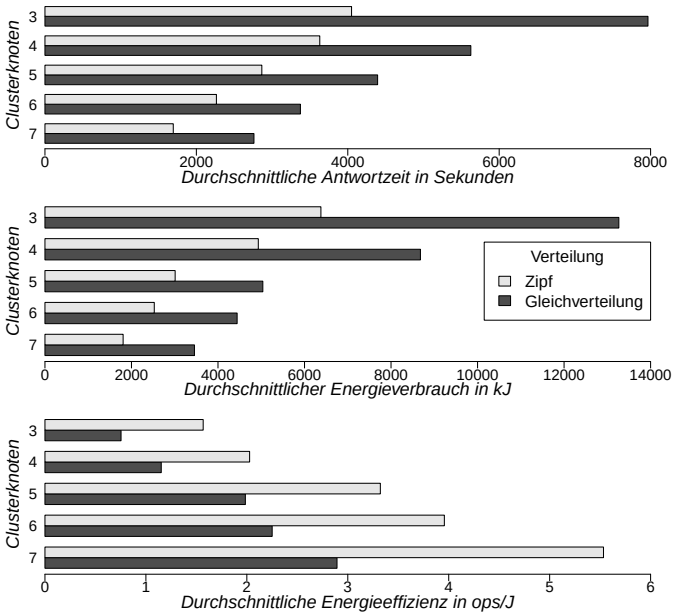
(a) Arbeitslast `load`



(b) Arbeitslast `read`



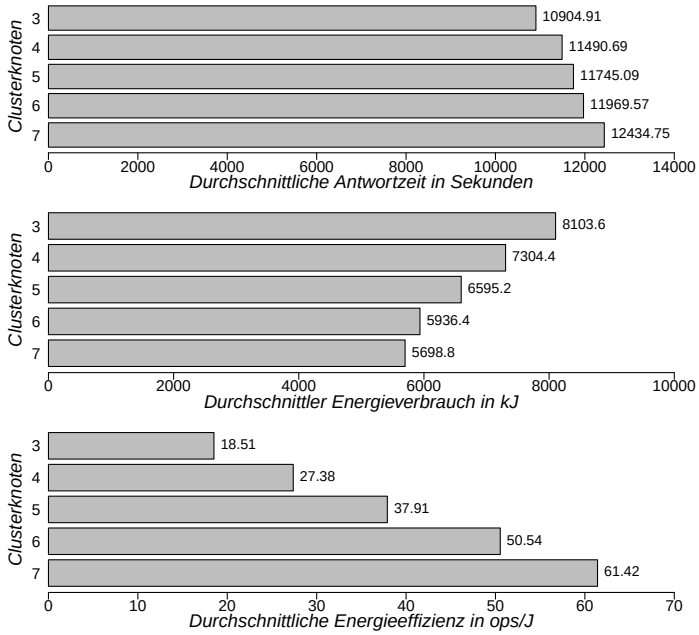
(c) Arbeitslast `write`



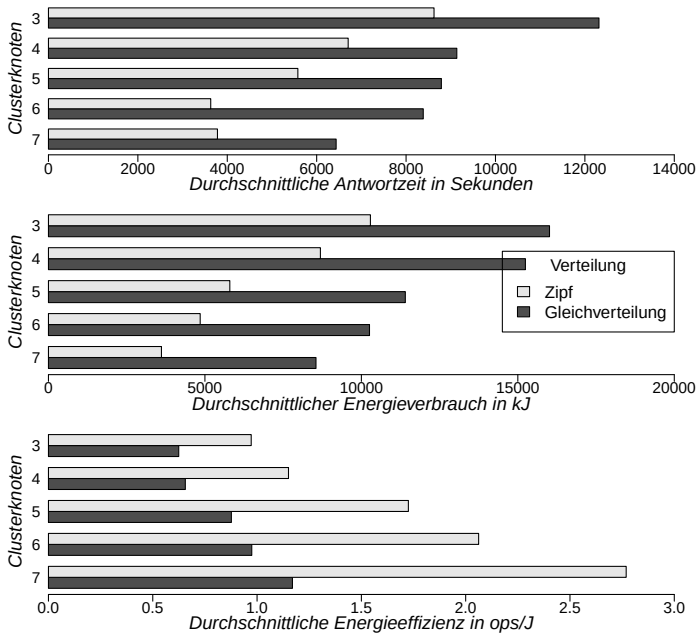
(d) Arbeitslast `mixed`

Abbildung A.3: Experimentelle Ergebnisse für Experiment E (Anwendungsfall A, fixe Datenmenge)

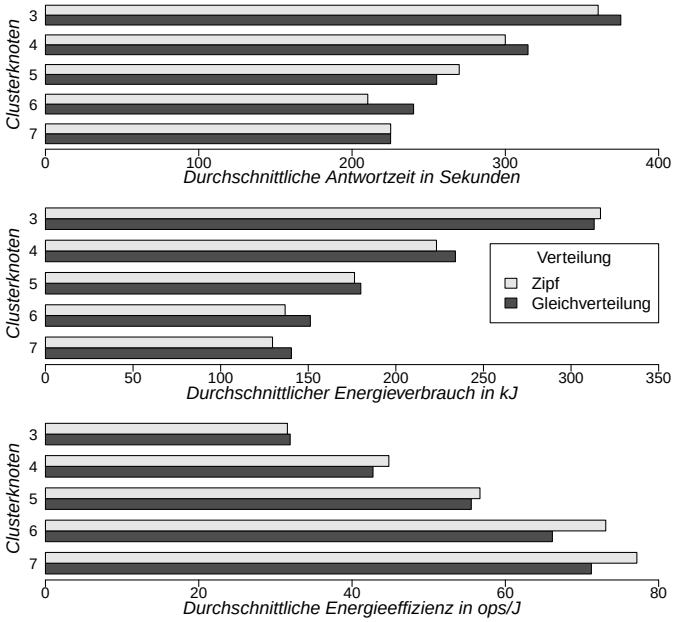
Die nachfolgenden Grafiken stellen die experimentellen Ergebnisse für Experiment E (vgl. Seite 98) für den Anwendungsfall B (stufenartig ansteigende Datenmenge) dar.



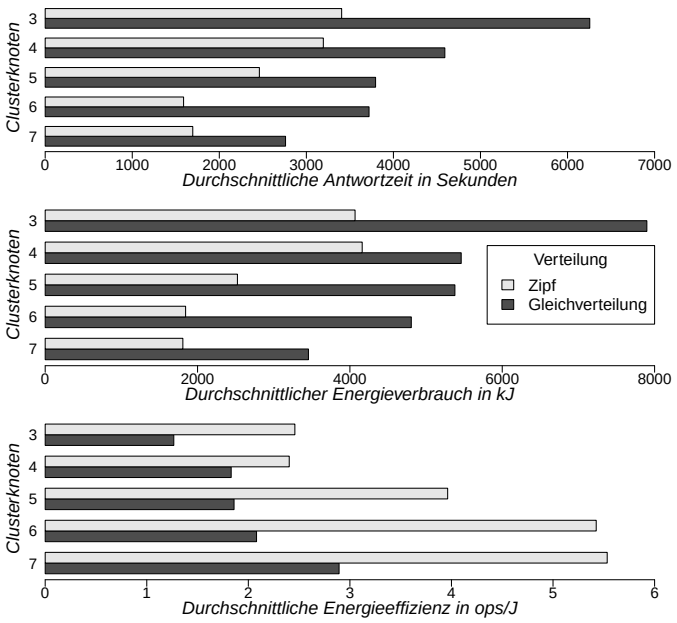
(a) Arbeitslast `load`



(b) Arbeitslast `read`



(c) Arbeitslast `write`



(d) Arbeitslast `mixed`

Abbildung A.4: Experimentelle Ergebnisse für Experiment E (Anwendungsfall B, stufenartig ansteigende Datenmenge)

A.1.4 Experimentelle Ergebnisse für Experiment F

Die nachfolgende Tabelle stellt die experimentellen Werte für Abbildung 4.19 und 6.12 numerisch dar.

	← *Cassandra*-Clustergröße →					
Genauigkeit	**3**	**4**	**5**	**6**	**7**	**8**
experimentell	12.61	18.27	24.50	29.71	34.18	38.80
→ Gewinn		44.87	34.09	21.25	15.04	13.53
simuliert	15.14	21.77	28.89	34.77	39.76	44.82
→ Gewinn		43.77	32.67	20.37	14.33	12.72

	← *Cassandra*-Clustergröße →				
Genauigkeit	**9**	**10**	**11**	**12**	**13**
experimentell	43.70	48.79	53.82	58.95	63.74
→ Gewinn	12.61	11.65	10.29	9.53	8.13
simuliert	50.15	55.66	61.10	66.44	71.39
→ Gewinn	11.90	10.98	9.75	8.74	7.44

Tabelle A.3: Experimentelle und simulierte Genauigkeit für Experiment F in Prozent