Technische Universität München
Fakultät für Informatik
Lehrstuhl für Wirtschaftsinformatik (I 17)
Univ.-Prof. Dr. Helmut Krcmar

Performance-Modellierung und Simulation eines SAP-Netweaver-Portal-Systems

Manuel Mayer

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. Martin Bichler

Prüfer der Dissertation:

1. Univ.-Prof. Dr. Helmut Krcmar
2. Univ.-Prof. Dr. Hans Michael Gerndt

Die Dissertation wurde am 14. Januar 2013 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 24. April 2013 angenommen.

Zusammenfassung

Ziel: Im Rahmen dieser Dissertation soll ein Ansatz für die Modellierung und diskrete Ereignissimulation eines SAP-Netweaver-Portal-Systems entwickelt werden. Dabei werden unter Verwendung eines definierten Lastmusters sowie einer steigenden Anzahl von Benutzern Leistungsdaten ermittelt. Die Komponenten, die als wesentliche Einflussfaktoren für das Leistungsverhalten gelten, werden aus der Literatur abgeleitet, modelliert und mithilfe der erfassten Leistungskennzahlen parametrisiert. Der zu entwickelnde Ansatz konzentriert sich auf die Applikationsschicht der Portalinfrastruktur und setzt für die Modellierung die auf der Warteschlangentheorie basierenden „Layered Queueing Networks“ (LQN) ein. Im Zuge der Modellerstellung werden spezielle Architekturmerkmale eines SAP-Netweaver-Portal-Systems berücksichtigt und bestehende Ansätze entsprechend erweitert.

Methode: Die Arbeit basiert auf einem gestaltungsorientierten Ansatz. Ausgehend von den bestehenden Erkenntnissen aus der Wissensbasis und dem in einer Online-Umfrage festgestellten Bedarf in der Praxis werden systeminhärente Einflussfaktoren auf die Leistung des Systems analysiert und in einem LQN-Modell, dem zentralen Artefakt dieser Arbeit, abgebildet. Die Gestaltung des Artefakts wird neben der Mess- und Warteschlangentheorie von den ermittelbaren Daten aus der Leistungsmessung geleitet. Für die Bewertung der Tragfähigkeit des Ansatzes wird das Modell unter Verwendung eines Lastmusters aus der Praxis instanziiert. Anschließend werden die erzielten Simulationsergebnisse sowie die ermittelten Messwerte zum Vergleich herangezogen und die Ursachen für auftretende Differenzen analysiert.

Resultate: Durch die Berücksichtigung und Umsetzung spezieller Architekturmerkmale eines SAP-Netweaver-Portal-Systems konnten vielversprechende Simulationsergebnisse in Bezug auf das Antwortzeitverhalten erzielt werden. Auftretende Wartezeiten aufgrund von Rechenengpässen sowie ausgelasteten Applikations-Threads werden dem tatsächlichen Verhalten entsprechend berücksichtigt. Ebenso konnten durch die Modellierung des Sperrmanagements und der Tabellenpufferung die davon abhängigen Bearbeitungszeiten der Datenbankaufrufe abgebildet werden. Die Evaluation der Ergebnisse zeigt zudem, dass im Überlastbereich äußere Einflussfaktoren, die von dem Modell nicht erfasst werden, das Antwortzeitverhalten entscheidend mitbestimmen und infolgedessen die Simulationsgenauigkeit stark beeinträchtigen. Eine Ursachenanalyse verdeutlicht hierbei den hohen Wirkungsgrad der Garbage-Collector-Aktivitäten auf die erzielten Antwortzeiten.

Auswirkungen auf die Praxis: Das in dieser Arbeit entwickelte Modell stellt einen neuartigen Ansatz zur Ex-Ante-Leistungsanalyse eines SAP-Netweaver-Portal-Systems dar. Die Frage, wie sich das System bei einer zunehmenden Lastintensität aufgrund von ansteigenden Benutzerzahlen verhält, wird in der Praxis häufig gestellt und kann mit dem vorgestellten Vorgehen frühzeitig analysiert werden. Aufgrund der hohen Marktdurchdringung von SAP-Systemen und der starken Integration in die Geschäftsprozesse eines Unternehmens weist das erstellte Artefakt eine hohe Relevanz auf. Der Modellierungsansatz lässt sich ebenfalls auf architekturverwandte Systeme übertragen, sodass lediglich das abgebildete Lastmuster sowie die Parametrisierung der Modellkomponenten angepasst werden müssen. Zusätzlich kann die Leistungsvorhersage im Zuge der Kapazitätsplanung für eine Kosten-Leistungs-Analyse verwendet und benötigte Hardwareressourcen spezifiziert werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ABAP	Advanced Business Application Programming
ACID	Atomicity, Consistency, Isolation, Durability
AIX	Advanced Interactive Executive
AJAX	Asynchronous Javascript and XML
AME	Active Memory Expansion
API	Application Programming Interface
AS	Application Server
ASP	Application Service Provider
BYD	Business By Design
CCMS	Computing Center Management System
CEC	Cross Electronic Circuit
CEN	Central Monitoring System
CISC	Complex Instruction Set Computer
CLP	Collaboration Launch Pad
CTMC	Continuous Time Markov Chains
CPN	Coloured Petri Nets
CPU	Central Processing Unit
CRM	Customer Relationship Management
CSV	Comma Separated Value
DHTML	Dynamic Hypertext Markup Language
DIAG	Dynamic Information and Action Gateway
DSAG	Deutschsprachige SAP Anwendergruppe
DSR	Distributed Statistical Records
DTD	Document Type Definition
DTMC	Discrete Time Markov Chains
EJB	Enterprise Java Beans
ERP	Enterprise Ressource Planning
FCFS	First Come, First Served
FIFO	First In, First Out
FLOPS	Floating Point Operations per Second
GC	Garbage Collector
GRMG	Generic Request and Message Generator

GUI	Graphical User Interface
GUID	Global Unique Identifier
HANA	High Performance Analytic Appliance
HOL	Head-of-Line Priority
HTML	Hypertext Markup Language
HTTP	Hyper Text Transport Protocol
I/O	Input/Output
IBM	International Business Machines
ID	Identifier
IS	Information Systems
ISR	Information Systems Research
IT	Information Technology
J2EE	Java 2 Enterprise Edition
JARM	Java Application Respones Time Measurement
JDBC	Java Database Connectivity
JNDI	Java Naming and Directory Provider
JTA	Java Transaction API
JTS	Java Transaction Service
JVM	Java Virtual Machine
KMU	Kleine und mittelständische Unternehmen
LCFS	Last Come, First Served
LCFS-PR	Last Come, First Served with Preempt and Resume
LOA	Large Object Area
LPAR	Logical Partition
LQN	Layered Queueing Networks
LQNS	Layered Queueing Network Solver
LQsim	Layered Queueing Network Simulator
LRU	Least Recently Used
LUW	Logical Unit of Work
MRP	Material requirements Planning
MRP II	Manufacturing Resource Planning
MRP III	Money resource Planning
MVA	Mean Value Analysis
NFS	Network File System

OS	Operating System
PARASOL	Parallel Simulation Object Library
PEER	Performance-Evaluations-Cockpit für ERP-Systeme
PCD	Portal Content Directory
PMAT	Pattern Modeling and Analysis Tool
PPR	Priority, Preemptive Resume
PS	Processor Sharing
QPN	Queueing Petri Nets
RISC	Reduced Instruction Set Computer
RR	Round Robin
SAAS	Software as a Service
SAP	Systeme, Anwendungen und Produkte in der Datenverarbeitung
SAP SID	SAP System Identifier
SAP UA	SAP University Alliances
SAP UCC	SAP University Competence Center
SAPS	SAP Application Performance Standard
SAT	Single Activity Trace
SDN	SAP Developer Network
SID	(siehe SAP SID)
SLA	Service Level Agreement
SPEC	Standard Performance Evaluation Corporation
SQL	Structured Query Language
SUN	Stanford University Network
TPC	Transaction Processing Performance Council
TPN	Time(d) Petri Nets
TPS	Transaction per Second
TUM	Technische Universität München
UA	(siehe SAP UA)
UCC	(siehe SAP UCC)
VIO	Virtual Input/Output
W3C	World Wide Web Consortium
XML	Extended Markup Language

1 Einleitung

Viele Bereiche der Natur-, Sozial- und Wirtschaftswissenschaften sind ohne den Einsatz von computergestützten Modellierungs- und Simulationstechniken nicht mehr denkbar. Physiker simulieren dynamische Eigenschaften von hochenergetischen Nuklearreaktionen (Blättel/Koch/Mosel 1993) oder die Entwicklung von Sternen und Galaxien (Kippenhahn/Weigert 1991), während ihre Kollegen aus den wirtschafts- und sozialwissenschaftlichen Disziplinen Kriegsausbrüche (Hermann/Hermann 1972), ökonomische Entwicklungen (Anderson/Arrow/Pines 1988) oder Entscheidungsfindungsprozesse in Organisationen (Simon 1970) nachvollziehen (vgl. Hartmann 1996).

Nach Niehans (1990, 313) hat die Ära der Modellentwicklung ihren Ursprung im späten 19. Jahrhundert. Zu dieser Zeit begannen Wissenschaftler vor allem im Bereich der Physik, ihre Aktivitäten auf die Bildung von Modellen zu konzentrieren. So schreibt William Thompson (1884) beispielsweise, dass er erst dann in der Lage sei, ein Phänomen zu verstehen, wenn er ein Modell des zu untersuchenden Systems erstellen könne. Ebenso formuliert Heinrich Hertz in seiner Einleitung über „Die Prinzipien der Mechanik“:

> „Ist es uns einmal geglückt, aus der angesammelten bisherigen Erfahrung Bilder von der verlangten Beschaffenheit abzuleiten, so können wir an ihnen, wie an Modellen, in kurzer Zeit die Folgen entwickeln, welche in der äußeren Welt erst in längerer Zeit oder als Folgen unseres eigenen Eingreifens auftreten werden; wir vermögen so den Thatsachen vorauszueilen und können nach der gewonnenen Einsicht unsere gegenwärtigen Entschlüsse richten.“ (Hertz 1894, 1f.)

Etwa 30 Jahre später erfreute sich die Modellbildung auch in den Wirtschaftswissenschaften großer Beliebtheit, wenngleich bereits im 18. Jahrhundert durch die Ökonomen Adam Smith und David Ricardo erste theoretische Modelle zur Beschreibung von ökonomischen Zusammenhängen entwickelt worden waren (Niehans 1990).

Werden in einem (dynamischen) Modell zeitliche Abläufe beschrieben, können Simulationstechniken zur Lösung der zugrundeliegenden Gleichungen eingesetzt werden. Eine Simulation imitiert somit die zeitliche Entwicklung eines weltlichen Prozesses oder Systems (Banks et al. 2004). Werden diese Abläufe von Computersystemen berechnet, wird von einer Computersimulation gesprochen.

Eben solche Computersysteme erfahren seit ihrem Beginn mit dem von Konrad Zuse im Jahr 1941 entwickelten, frei programmierbaren Rechenautomaten eine enorme Entwicklung in ihrer Leistungsfähigkeit (vgl. Moore's Law, Moore 1965). Heutzutage ist der Computer ein zentrales Instrument für verschiedenste Einsatzzwecke, sowohl im geschäftlichen als auch im privaten Bereich. Mit dem vielfältigen Einsatz geht ein breites Spektrum an verfügbaren Architekturen einher, das wiederum die Auswahl eines bestimmten Computersystems für eine bestimmte Anforderung erfordert. Sobald die Auswahl einer Architektur getroffen wurde, stellen sich Fragen zur Leistungsfähigkeit des gewählten Systems. Diese Fragestellungen fallen in den Bereich der Performance-Evaluation von Computersystemen. Ein möglicher Ansatz ist dabei der Einsatz von modellbasierter Performance-Simulation.

1.1 Problemstellung und Motivation der Arbeit

Speziell im Unternehmensumfeld ist die Frage zur Leistungsfähigkeit der eingesetzten Computersysteme von großer Bedeutung. Obwohl kein direkter Bezug zwischen der IT und ihrem erbrachten Wert hergestellt werden kann (Krcmar 2010, 520), führen unzureichende Systemressourcen zu hohen Antwortzeiten und somit zu einer erhöhten Prozessdurchlaufzeit, während im konträren Fall eine ungenaue und zu großzügige Einschätzung der benötigten Systemressourcen meist erhöhte IT-Investitionen nach sich zieht.

Die eingesetzten Geschäfts-Applikationen in einem Unternehmen basieren oft auf verteilten, mehrschichtigen Architekturen und beinhalten verschiedene Komponenten in einer heterogenen Systemumgebung. Die inhärente Komplexität dieser Umgebung erschwert es Systemarchitekten, die benötigte Größe und Kapazität richtig einzuschätzen, damit festgelegte Service-Level-Agreements (SLAs) eingehalten werden können (Kounev/Buchmann 2003). Aufgrund der oft unzureichenden Integration der verschiedenen Geschäfts-Applikationen, der redundanten Datenhaltung sowie der losen Kopplung von Einzelsystemen, bietet sich der Einsatz eines unternehmensweiten, integrierenden Systems an. Dies wird von sogenannten Enterprise-Ressource-Planning-Systemen (ERP-Systemen) übernommen (vgl. bspw. Krcmar 2010, 236; Nah/Zuckweiler/Lau 2003).

Darüber hinaus bieten Unternehmensportale eine einheitliche Oberfläche für Mitarbeiter, Kunden und Lieferanten sowohl zur Integration verschiedener Dienste und Anwendungen, als auch zur Personalisierung und flexiblen Zuweisung von Benutzerrollen. Mit anderen Worten stellen sie eine Integrationsplattform für die Optimierung der betrieblichen Informations- und Wissenslogistik dar und eröffnen neue organisatorische und informationstechnische Möglichkeiten. Daher begannen die Unternehmen zu Beginn dieses Jahrtausends stark in Unternehmensportale zu investieren. Beispielsweise verkündeten SAP und Siemens im Jahr 2002 (SAP 2002), die Portaltechnologie von SAP künftig konzernweit für alle rund 400.000 Siemens-Mitarbeiter einzusetzen (Amberg/Remus/Holzner 2003).

Die Integration der Dienste und Anwendungen auf einer einheitlichen Plattform werfen umso mehr kritische Fragen zur Leistungsfähigkeit des zentral verwendeten Systems auf:

- Welche maximalen Benutzerzahlen kann das System mit einer akzeptablen Geschwindigkeit bewältigen?
- Welche durchschnittliche Antwortzeit kann bei einer gegebenen Lastintensität erzielt werden?
- Welche Hardware- und Software-Ressourcen stellen Engpässe dar?
- Welche Systemressourcen werden benötigt, um gegebene SLAs einzuhalten?

Oft werden zur Beantwortung dieser Fragen Expertenmeinungen, Intuition und Daumenregeln herangezogen, da eine methodische Analyse entweder nicht verfügbar oder zu aufwändig in der Durchführung ist (Kounev/Buchmann 2003). Daher beschäftigt sich die vorliegende Arbeit mit der Frage, wie ein Modell zur technischen Performance-Evaluation und Simulation eines SAP-Netweaver-Portal-Systems charakterisiert und parametrisiert werden kann.

Online-Umfrage zur Performance-Evaluation von ERP-Systemen

In diesem Zusammenhang wurde 2010 eine Online-Umfrage zur Performance-Evaluation von ERP-Systemen durchgeführt (Mayer et al. 2011), die unter anderem darüber Aufschluss geben sollte, ob und auf welche Art und Weise Leistungsanalysen durchgeführt werden, zu welchem Zweck diese durchgeführt werden und welche Anforderungen an die eingesetzten Hilfsmittel gestellt werden.

Die Online-Umfrage wurde im Dezember 2010 auf zwei großen Internetplattformen initiiert, und zwar der Deutschsprachigen SAP-Anwendergruppe (DSAG) und dem SAP-Developer-Network (SDN). Die Befragung beschränkte sich auf den deutschen Raum, wobei die am meisten vertretenen Branchen IT-Dienstleistungen (25%), die öffentliche Verwaltung (19,44%) sowie Banken und Versicherungen (11,11%) waren.

97 Prozent der Befragten, die fast ausschließlich eine langjährige Erfahrung im Bereich von SAP-Systemen aufweisen und hauptsächlich Positionen als IT-Berater (33,33%) oder IT-Projektleiter (27,78%) bekleiden, treten dem Thema Performance-Evaluation sehr aufgeschlossen gegenüber. Zwei Drittel erachten diese als sehr wichtig, allerdings werden ausschließlich die in den ERP-Systemen integrierten Hilfsmittel zur Leistungsanalyse genutzt. Als Grund für den Verzicht auf weitere Hilfsmittel werden der notwendige Einarbeitungs- und Betreuungsaufwand sowie zusätzliche Kosten genannt.

Knapp 70 Prozent der Befragten nutzen Lasttests, jedoch nur ein Viertel verwendet analytische Modelle zur Performance-Evaluation und nur etwa ein Drittel der Umfrageteilnehmer setzt Simulation ein. Damit einhergehend wird nur selten versucht, Prognosen über die Eignung der genutzten ERP-Systeme für zukünftige Lastprofile zu erstellen. Als Ursache hierfür werden funktionale Schwächen der verwendeten Werkzeuge genannt. Zudem führt ein mangelndes Vertrauen in die Akkuratesse der prognostizierten Werte zu dem geringen Einsatz. Keiner der Befragten ist voll zufrieden mit der Reliabilität, 62,5 Prozent sind im Allgemeinen zufrieden und ein Viertel der Befragten ist nur teilweise zufrieden mit den Ergebnissen der eingesetzten Hilfsmittel.

Nichtsdestotrotz werden die Vorteile der Leistungsanalyse von ERP-Systemen gesehen, vor allem in Bezug auf Effizienz (85,29%), Risikominderung (76,47%), höhere Flexibilität (50%) sowie Kostenreduktion (44,12%). Ebenso wurde in den freien Anmerkungen mehrfach betont, dass ein reges Interesse an einer Entwicklung zusätzlicher Methoden und Hilfsmittel zur Leistungsanalyse und -prognose besteht, die die bisherigen, genannten Defizite mindern.

Es stellt sich somit heraus, dass der Bedarf an einem methodischen Vorgehen zur Performance-Evaluation von Unternehmenssoftware besteht, das den bestehenden Mängeln der Leistungsbewertung entgegenwirkt. Ebenso hat sich gezeigt, dass sich die Leistungsanalyse hauptsächlich auf die Verwendung von integrierten Hilfsmitteln beschränkt und modellbasierte, analytische Verfahren sowie Simulation nur teilweise zum Einsatz kommen. Diese Lücke soll mit der vorliegenden Arbeit reduziert werden, indem ein in der Literatur etablierter, modellbasierter Ansatz zur Leistungsanalyse eines SAP-Netweaver-Portal-Systems, welches in den Kontext von Unternehmenssoftware einzuordnen ist, angewendet wird.

1.2 Ziele

Zusammenfassend hat die vorliegende Arbeit das Ziel, aus den Erkenntnissen der Literatur und den analytischen Hilfsmitteln eines SAP-Netweaver-Portal-Systems, relevante Leistungsdaten zu erfassen und über die Modellierung und Simulation eines gegebenen Lastmusters (Workloads) sowie der Komponenten, die als Einflussgrößen für das Leistungsverhalten gelten, Leistungswerte bei einer steigenden Anzahl von Benutzern zu ermitteln.

Im ersten Schritt werden aus der Literatur die Grundlagen erarbeitet, die für die Performance-Evaluation eines SAP-Netweaver-Portal-Systems benötigt werden. Für die Modellierung und Simulation werden die auf der Warteschlangentheorie basierenden Layered-Queueing-Networks (LQNs) sowie ein an der Carleton University entwickelter LQN-Simulator (LQsim 2011) eingesetzt. LQNs sind eine Erweiterung der klassischen Warteschlangennetze, um komplexe Software-Systeme über logische Software-Ressourcen, die eine vertikal geschichtete Struktur bilden, zu beschreiben. Sie entstammen den Arbeiten über „Active Servers“ (Woodside et al. 1986), dem „Lazy Boss Model“ (Rolia 1988), den „Rendezvous Networks“ (Woodside et al. 1995) sowie der „Method of Layers“ (Rolia/Sevcik 1995). Neben der hierarchischen Struktur bietet die Unterstützung von replizierten Entitäten, geteilten Warteschlangen und synchronen Aufrufen, um nur einige zu nennen, eine sehr gute Grundlage zur Modellierung der komponentenbasierten, geschichteten Architektur eines SAP-Netweaver-Portal-Systems.

Neben der Aufarbeitung der theoretischen Grundlagen werden die Hilfsmittel zur Überwachung der Leistungsdaten (Monitoring-Hilfsmittel) eines SAP-Netweaver-Portal-Systems untersucht. Ziel ist es, die in der SAP-Netweaver-Dokumentation genannten Performance-Einflussgrößen und ermittelbare Leistungswerte zu identifizieren. Dazu ist es notwendig, die Architektur eines SAP-Netweaver-Portal-Systems zu analysieren und für die einzelnen Komponenten entsprechende Monitoring-Hilfsmittel zu ermitteln. Zudem werden Analyse-Werkzeuge auf Betriebssystemebene zur Erfassung von grundsätzlichen Leistungsdaten (CPU, Hauptspeicher, I/O) erörtert.

Aus den Erkenntnissen der theoretischen Grundlagen, der dokumentierten Architektur sowie der verfügbaren Monitoring-Hilfsmittel werden die zu modellierenden Komponenten abgeleitet, die als wesentliche Performance-Einflussgrößen eines SAP-Netweaver-Portal-Systems gelten. Dazu werden den einzelnen Architekturkomponenten zentrale Einflussfaktoren und Möglichkeiten zur Leistungsdatenerfassung zugeordnet. Beispielsweise können aus der Architekturkomponente Tabellenpuffer die speziellen Einflussfaktoren Verdrängungen und Invalidierungen identifiziert werden, deren Anzahl über den Tabellenpuffer-Trace erfasst und festgehalten wird.

Mit der Identifikation der zu modellierenden Komponenten bzw. Einflussgrößen ist es Ziel dieser Arbeit, ein entsprechendes LQN-Modell zur Performance-Simulation eines SAP-Netweaver-Portal-Systems zu erstellen. Da der verwendete Workload inhärenter Bestandteil eines LQN-Modells ist, wird die Modellierung und Parametrisierung einzelner Lastschritte ermöglicht, die wiederum, entsprechend ihrem tatsächlichen Verhalten, einzelne Systemkomponenten verwenden. Dieser Schritt der Modellierung ist stets abhängig vom verwendeten Workload, allerdings bleibt das grundsätzliche Vorgehen identisch. Für das Sperrmanagement und zur expliziten Parametrisierung der Aktivitätsintensität des Garbage-Collectors (GC)

werden in der vorliegenden Arbeit zusätzliche Komponenten modelliert, die in der LQN-Literatur bis dato nicht untersucht worden sind.

Für die anschließend durchgeführte Evaluation des LQN-Modells wird das Leistungsverhalten eines SAP-Netweaver-Portal-Systems unter einer ansteigenden Anzahl von Benutzern analysiert und mit den ermittelten Leistungsdaten der Simulation verglichen. Die dabei verwendete Performance-Metrik beschränkt sich auf die Antwortzeit als Messgröße, da diese eine hohe Aussagekraft für die besagte Fragestellung bietet, mit den vorhandenen Analyse-Hilfsmitteln gemessen werden kann und die Anforderungen an eine Metrik für die Performance-Analyse erfüllt (Lilja 2000). Ziel ist es, aus der Analyse und Interpretation der Evaluationsergebnisse Rückschlüsse auf die Aussagekraft der eingesetzten Performance-Simulation zu ziehen. Als Workload werden die Inhalte der SAP-Netweaver-Portal-Schulung aus dem „SAP University Alliances Programm“ (SAP UA 2012) eingesetzt, die unter anderem von dem „SAP University Competence Center“ der Technischen Universität München (SAP UCC TUM 2012) angeboten werden. Dabei handelt es sich um ein breites Spektrum grundlegender und erweiterter Portalfunktionen, die unter anderem zur Inhaltserstellung, Verwaltung, Personalisierung und Kommunikation mit anderen Benutzern verwendet werden.

Zusammenfassend lassen sich die Kernbeiträge der vorliegenden Arbeit wie folgt festhalten:

- Identifikation der wesentlichen Performance-Einflussgrößen aus den Erkenntnissen der Literatur und Dokumentation des SAP-Netweaver-Portal-Systems, in Abhängigkeit von den gegebenen Möglichkeiten zur Leistungsdatenerfassung.

- Erstellung eines LQN-Modells zur Performance-Simulation eines SAP-Netweaver-Portal-Systems, unter Berücksichtigung der identifizierten Einflussgrößen. Das LQN-Modell beinhaltet unter anderem einen Ansatz zur Abbildung von Lese- und Schreibsperren im Sperrmanagement sowie der Aktivitätsintensität des Garbage-Collectors, da diese in der LQN-Literatur bisher nicht behandelt wurden.

- Evaluation und Interpretation der Simulationsergebnisse anhand der Modellierung und Simulation einer am SAP UCC der Technischen Universität München angebotenen Portal-Schulung.

Nicht Bestandteil dieser Arbeit ist die Modellierung der Komponenten des zugrundeliegenden Datenbanksystems. Dieses wird als Black-Box betrachtet und mögliche Performance-Engpässe ausgeschlossen, damit der Fokus allein auf die Applikationsebene gesetzt werden kann. Ebenso wird davon ausgegangen, dass die Clients auf der Präsentationsebene über genügend Ressourcen verfügen. Da die Leistungsdaten direkt am Applikationsserver erfasst werden, beinhalten die ermittelten Antwortzeiten keine Netzwerkzeiten zwischen Client und Server. Nicht zuletzt beschränkt sich das in dieser Arbeit entwickelte LQN-Modell auf die Architektur eines SAP-Netweaver-Portal-Systems. Portalsysteme anderer Hersteller oder zukünftige Portal-Versionen können architekturale Unterschiede aufweisen und würden eine Anpassung des Modells erfordern. Wenngleich diese Anpassungen möglicherweise eine erneute Analyse der Systemarchitektur und vorhandener Monitoring-Hilfsmittel erfordern, können aus dieser Arbeit die Modellierungskonzepte der betrachteten Einflussgrößen übernommen werden.

1.3 Forschungsfragen

Aus der beschriebenen Problemstellung leiten sich drei forschungsleitende Fragen ab, die in dieser Arbeit beantwortet werden.

> Forschungsfrage 1:
>
> Wie kann im Zuge der Performance-Modellierung und Simulation die Architektur eines SAP-Netweaver-Portal-Systems charakterisiert werden und welche Systemkomponenten sowie Einflussgrößen auf das Leistungsverhalten lassen sich aus der Literatur und den gegebenen Hilfsmitteln zur Leistungsdatenerfassung ableiten?

Im Rahmen der ersten Forschungsfrage wird die Architektur eines SAP-Netweaver-Portal-Systems analysiert. Dabei werden die einzelnen Komponenten des Systems betrachtet, die der Literatur folgend als wesentliche Einflussgrößen auf das Leistungsverhalten des Systems gelten. Zudem werden vorhandene Hilfsmittel zur Erfassung und Aufzeichnung der Leistungsdaten ermittelt.

Die Beantwortung der ersten Forschungsfrage ergibt somit eine Gruppe von identifizierten Komponenten, die die Performance wesentlich beeinflussen und deren Leistungsdatenerfassung über die gegebenen Analysewerkzeuge ermöglicht wird. Somit bildet das Ergebnis der ersten Forschungsfrage die Grundlage für eine akkurate Performance-Modellierung eines SAP-Netweaver-Portal-Systems.

> Forschungsfrage 2:
>
> Wie können die ermittelten Komponenten und Einflussgrößen in einem LQN-Modell umgesetzt und parametrisiert werden, damit möglichst akkurate und zuverlässige Simulationsergebnisse erzielt werden?

Die zweite Forschungsfrage widmet sich der Umsetzung der in Forschungsfrage 1 ermittelten Komponenten und Einflussgrößen in einem LQN-Modell und damit der Erstellung des zentralen Artefakts dieser Arbeit. Aufgrund von modellierungsbezogenen Limitationen sowie der stochastischen Natur von LQNs entspricht die Abbildung der Komponenten einer Annäherung an das tatsächliche Verhalten. Daher wird bei dem vorgestellten Vorgehen zur Modellierung der einzelnen Komponenten auf bestehende Einschränkungen hingewiesen und für die Festsetzung einzelner Parameter Möglichkeiten zur Berechnung aufgezeigt.

Das Ergebnis von Forschungsfrage 2 ist, wie bereits erwähnt, ein LQN-Modell zur Performance-Evaluation eines SAP-Netweaver-Portal-Systems. Mögliche Performance-Einflüsse auf der Präsentations- sowie Datenbankebene werden ausgeschlossen, damit der Fokus auf die Applikationsebene gelegt werden kann. Da der betrachtete Workload inhärenter Bestandteil des LQN-Modells ist, sieht das vorgestellte Modell die Modellierung einzelner Lastschritte vor, die von den Benutzern aufgerufen werden. Diese müssen zwar bei einer Änderung des betrachteten Workloads angepasst werden, das grundsätzliche Vorgehen kann jedoch beibehalten werden.

Forschungsfrage 3:

Welche Resultate werden, unter Verwendung eines Workloads aus der Praxis, mittels Simulation des LQN-Modells im Vergleich zu den Messwerten erzielt und wie lassen sich auftretende Abweichungen erklären?

Die Simulation und Evaluation des in Forschungsfrage 2 erstellten LQN-Modells steht bei der Beantwortung von Forschungsfrage 3 im Fokus. Eine am SAP UCC der TU München eingesetzte Portal-Schulung (auch Fallstudie genannt) dient als Basis für den modellierten Workload.

Für die Lasterzeugung wird das in Jehle (2010, 131f.) entwickelte Performance-Evaluation-Cockpit für ERP-Systeme (PEER) eingesetzt. Dabei wird die Funktionalität zur Generierung des Workloads genutzt; die Ermittlung der Leistungsdaten erfolgt über die entsprechenden Log- und Trace-Dateien des SAP-Netweaver-Portal-Systems. Ein Lasttreiber für das Portalsystem von SAP ist bereits vorhanden, sodass dieser lediglich für die Ziele dieser Arbeit angepasst werden muss.

Im Zuge der Evaluation der Simulationsergebnisse werden zwei Szenarien entwickelt, die den Fokus auf unterschiedliche zu untersuchende Eigenschaften des Performance- und Simulationsverhaltens legen. Grundsätzlich wird die aggregierte Antwortzeit der einzelnen Interaktions- bzw. Lastschritte bei einer steigenden Anzahl von Benutzern betrachtet. Neben der Genauigkeit der Simulationswerte werden die Ursachen für das Leistungsverhalten in den entsprechenden Lastbereichen untersucht. Während im ersten Szenario die Leistung des Portalsystems bei genügend Hardware-Ressourcen unter die Lupe genommen wird, werden im zweiten Szenario die Bedingungen verschärft, indem bestimmte Hardware- bzw. Systemressourcen verknappt werden. Aus den Erkenntnissen dieser Ergebnisse werden im Anschluss der System-Overhead und das Java-Speichermanagement analysiert, da diese im Hochlastbereich einen starken Einfluss auf das Antwortzeitverhalten ausüben.

Der Vergleich der Mess- und Simulationswerte widmet sich neben der Darstellung der Akkuratesse ebenso der Ursachenanalyse für das Leistungsverhalten und etwaigen Abweichungen. Dabei wird geprüft, wie sich die Einflussgrößen auf das Antwortzeitverhalten auswirken und ob sie von dem Simulationsmodell entsprechend erfasst werden. Diese Analyse zeigt, ob die identifizierten Einflussgrößen aus Forschungsfrage 1 das Antwortzeitverhalten ausreichend genau beschreiben und untersucht die Gründe für potentielle Abweichungen. Dabei muss zwischen Einschränkungen bei der Messung und Parametrisierung sowie nicht explizit modellierten Einflussgrößen unterschieden werden.

Das Ergebnis von Forschungsfrage 3 ist somit die Evaluation des in Forschungsfrage 2 erstellten Artefakts mittels einer Fallstudie. Die Bewertung erfolgt zunächst faktisch über den Vergleich zwischen den gemessenen und simulierten Antwortzeiten sowie der Analyse des Antwortzeitverhaltens und potentiellen Abweichungen. Abschließend wird eine Interpretation der Ergebnisse durchgeführt, die die erreichten Resultate einordnet und erlangtes Wissen beschreibt.

1.4 Forschungsdesign

Ein wesentlicher Bestandteil der Wirtschaftsinformatik ist das Design und die Implementierung von Artefakten der Informationstechnologie, mit dem Ziel, die Performance von Unternehmen zu verbessern. Das Management der Unternehmen sieht die Performance aus einer ökonomischen Perspektive und stellt gerechtfertigterweise die Fragen, warum Investitionen in IT-Artefakte oftmals nicht die gewünschte Wertsteigerung erzielen und welche IT-Artefakte dies erreichen können. Während die erste Frage theoriebasiert und kausalbezogen ist, stellt die zweite Frage eine konstruktionsorientierte, problemlösende Fragestellung dar. Wissenschaftliche Arbeiten, die letztere Frage adressieren und dabei IT-Artefakte erstellen und evaluieren, fallen in den Bereich der gestaltungsorientierten Forschung (Design-Science) (vgl. March/Storey 2008).

Die gestaltungsorientierte Forschung hat ihre Wurzeln in den ingenieurswissenschaftlichen Disziplinen und untersucht im Gegensatz zu den Naturwissenschaften, die natürlich vorkommende Phänomene erforschen, vom Menschen erschaffene Gegebenheiten, um bestimmte Ziele zu erreichen (Simon 1996). In diesem Zusammenhang postuliert Simon (1996, 141f.) in seinem Buch „The Science of the Artificial“ zwei grundsätzliche Elemente: Zum einen soll der Problemraum definiert werden, der die gewünschte Situation, die derzeitige Situation und den Unterschied zwischen den Situationen darstellt. Zum anderen sollen über einen Suchprozess Aktionen definiert werden, die mit großer Wahrscheinlichkeit bestimmte Unterschiede zwischen der derzeitigen und der gewünschten Situation beseitigen. Es ist somit die Hauptaufgabe der gestaltungsorientierten Forschung, Designprobleme aufzuzeigen und Lösungen zu erstellen und zu evaluieren (March/Storey 2008).

Neben der verhaltensorientierten Forschung erfreut sich der gestaltungsorientierte Forschungsansatz in der Wirtschaftsinformatik, aber auch zunehmend im angloamerikanisch geprägten Information-Systems-Research (ISR), einer immer größer werdenden Beliebtheit (Hevner 2007; Iivari 2007). A. Hevner formuliert diesbezüglich in einem für die Zeitschrift „Wirtschaftsinformatik“ durchgeführten Interview:

> „Ich sehe die steigende Wahrnehmung der Design Research im Bereich Information Systems als eine ganz natürliche Evolution unseres Forschungsgebiets. […] Ein Großteil der Arbeit von IS-Fachkräften und Managern beschäftigt sich mit „Gestalten“ (Design) – dem zweckorientierten Einsatz von Ressourcen zur Erreichung eines Ziels. Aus diesem Grund ist es meines Erachtens für IS-Forscher unerlässlich, die gestaltungsorientierten Wissenschaften besser zu verstehen und die gewonnenen Erkenntnisse (bspw. Design-Theorien, Design-Prozesse) in die Erstellung und Evaluation neuer und nützlicher Artefakte einfließen zu lassen.“ (Winter 2009, 148)

Besagte Artefakte werden in Konstrukte, Modelle, Methoden und Instanziierungen unterteilt, um die Repräsentation, die Analyse, das Verständnis und die Entwicklung von bewährten Informationssystemen innerhalb von Organisationen zu ermöglichen (March/Smith 1995). Konstrukte bilden die Sprache, in der Probleme und Lösungen definiert und kommuniziert werden. Modelle verwenden definierte Konstrukte, um Probleme und den Lösungsraum darzustellen. Sie stellen somit eine zielgerichtete Abstraktion der Realität dar. Methoden definieren den Prozess, wie Probleme gelöst werden, das heißt sie können Algorithmen sowie informale Beschreibungen von Best-Practice-Lösungen enthalten, die zeigen, wie der Lösungs-

raum entwickelt oder durchsucht wird. Instanziierungen zeigen schließlich die Machbarkeit, das heißt wie die Konstrukte, Modelle und Methoden in der realen Welt umgesetzt werden können (vgl. bspw. Bichler 2006).

Das Ziel, eine Klasse von Problemen mittels eines innovativen Artefakts zu lösen und die Situation damit zu verbessern, ist nicht immer einfach zu erreichen. Der Grund hierfür liegt oft an dem Mangel an vorhandenen Theorien, die als Basis für den Lösungsansatz von neuartigen Problemen oder bekannten Problemen in neuen Domänen dienen können (Hevner et al. 2004, 76). Die kreierten Artefakte entspringen somit weniger zugrundeliegenden Theorien, viel mehr entwickeln sich theoretische Modelle aus der Anwendung und Evaluation der Artefakte, da diese für ein besseres Verständnis des Problems und der inhärenten Mechanismen sorgen (Nunamaker et al. 1991).

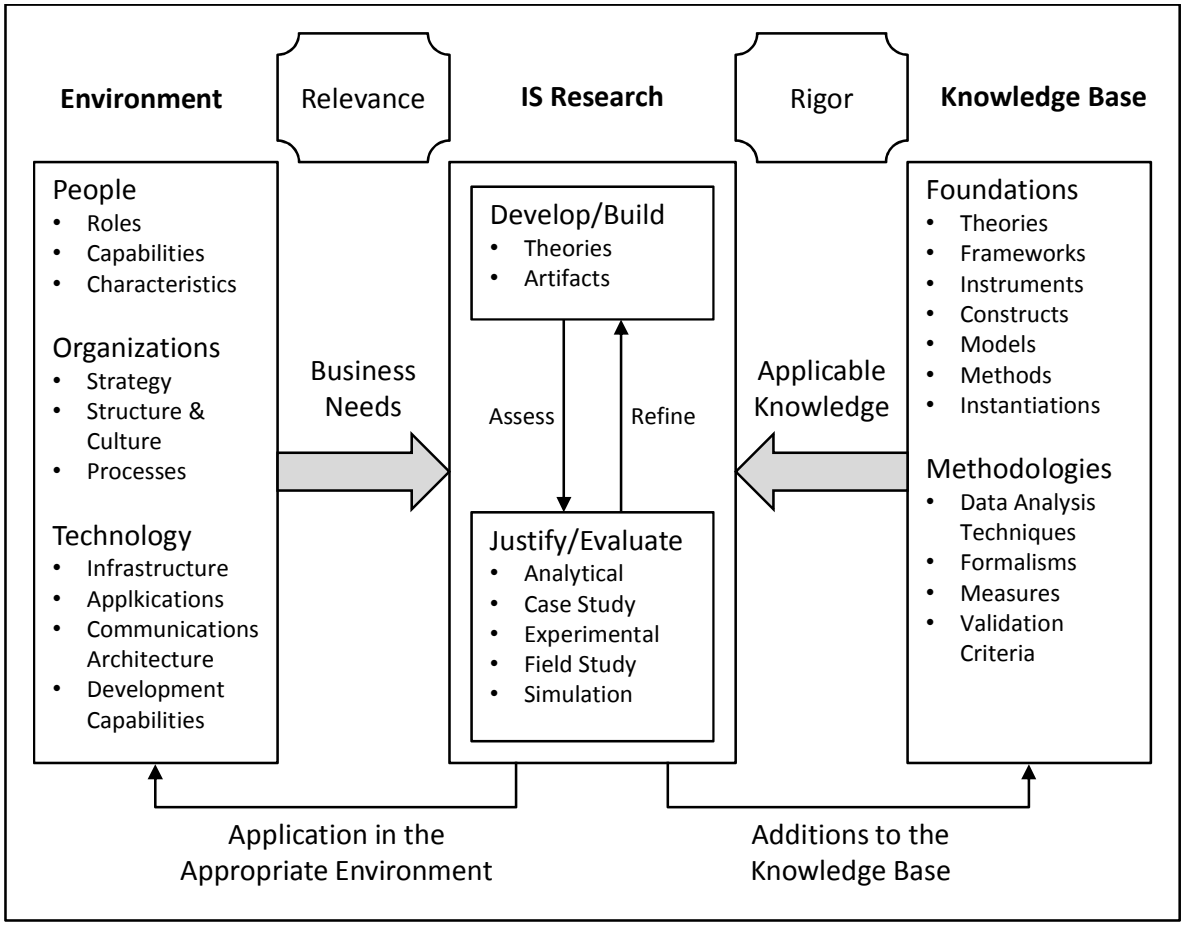


Abbildung 1-1: Design-Science-Framework
Quelle: Hevner et al. (2004, 80)

Konzeptionelle Rahmenbedingungen sowie Handlungsempfehlungen zum Designprozess können der Arbeit von Hevner et al. (2004) entnommen werden. Wie dem in Abbildung 1-1 dargestellten Rahmenwerk zu entnehmen ist, bezieht die der angewandten Forschung angehörende gestaltungsorientierte Forschung ihre Zweckdienlichkeit aus dem Umfeld („Environment“). Die Elemente des Umfelds werden in Menschen („People“), Organisationen („Organizations“) und Technologie („Technology“) untergliedert, die sich wiederum gegenseitig beeinflussen (vgl. bspw. Krcmar 2010, 28). Der gestaltungsorientierte Forschungsprozess erfolgt iterativ zwischen der Entwicklung („Develop/Build“) und der Bewertung („Jus-

tify/Evaluate“) des Artefakts (Hevner et al. 2004, 79) und ermöglicht somit eine wiederholte Überprüfung bzw. Verbesserung des Artefakts. Bei der Entwicklung des Artefakts wird auf etablierte Theorien, Methoden und Modelle zurückgegriffen, die unter dem Begriff Wissensbasis („Knowledge Base“) subsumiert sind (Hevner et al. 2004, 80). Dadurch wird die Stringenz („Rigor“) des Vorgehens sowie der entwickelten Lösungen sichergestellt und der Bezugsrahmen definiert. Aus den Erfahrungen über das entwickelte Artefakt und einem besseren Verständnis für den Problemraum kann wiederum neues Wissen an die Wissensbasis zurückgeführt werden.

Unter Berücksichtigung des Design-Science-Frameworks von Hevner et al. (2004) folgt diese Arbeit dem Paradigma der gestaltungsorientierten Forschung. Es wird eine aktuelle Herausforderung im Unternehmensumfeld („Business Need“) adressiert, deren Problemstellung und Motivation bereits in Kapitel 1.1 dargestellt wurde. Für den vorgestellten Lösungsansatz werden existierende Erkenntnisse aus der Wissensbasis verwendet („Applicable Knowledge“) und in Kapitel 2 und 3 diskutiert. Kapitel 4 stellt das entwickelte Artefakt vor. Eine Instanz des Artefakts wird in Kapitel 5 praktisch angewendet („Application in the Appropriate Environment“) und evaluiert. Die Erkenntnisse der Evaluation aus Kapitel 5 sowie die Interpretation der Ergebnisse in Kapitel 6 führen das erlangte Wissen an die Wissensbasis zurück („Additions to the Knowledge Base“).

Obwohl in der Arbeit von Hevner et al. (2004) neben dem vorgestellten Rahmenwerk diverse Handlungsempfehlungen gegeben werden, wird kein Forschungsprozess beschrieben, der eine ordnungsgemäße Durchführung der Designwissenschaft definiert. Daher wird in dieser Arbeit der von Pfeffers et al. (2006, 89ff.) vorgeschlagene Designprozess angewendet. Dieser wird in sechs Phasen unterteilt:

1. *Problemidentifikation und Motivation*: Ziel dieser Phase ist die Definition der Problemstellung und der Motivation der Arbeit. Neben dem Problemverständnis sollen bestehende Lösungen analysiert und das Verbesserungspotential aufgezeigt werden.

2. *Zieldefinition*: Aus dem Problemverständnis der ersten Phase werden die Ziele, quantitativer oder qualitativer Art, abgeleitet.

3. *Konzeption und Entwicklung*: In dieser Phase wird, unter Zuhilfenahme der Erkenntnisse aus der Wissensbasis, das eigentliche Artefakt entwickelt.

4. *Demonstration*: Die Zweckmäßigkeit des entwickelten Artefakts wird in der vierten Phase überprüft, indem ein geeignetes Anwendungsszenario gewählt und eine Instanz des Artefakts eingesetzt wird.

5. *Evaluation*: In der Evaluationsphase werden die erreichten Ergebnisse in Bezug auf die formulierten Ziele bewertet. Über die Auswahl einer Evaluationsmethode (vgl. Hevner et al. 2004, 83) wird die Effizienz und Effektivität des entwickelten Artefakts überprüft und führt gegebenenfalls zu einem neuen Entwicklungszyklus.

6. *Kommunikation*: Schließlich werden die gewonnenen Erkenntnisse (Beschreibung des Problemraumes, Signifikanz, Lösungsansatz, erwarteter und evtl. nachgewiesener Beitrag sowie Stringenz des Designprozesses) an die Wissensbasis zurückgeführt.

1.5 Forschungsbereich am Lehrstuhl für Wirtschaftsinformatik

Die vorliegende Arbeit ist Teil einer Reihe von Forschungsarbeiten im Bereich der Performance-Evaluation von SAP-ERP-Systemen, die am Lehrstuhl für Wirtschaftsinformatik von Prof. Dr. Helmut Krcmar an der Technischen Universität München (TUM) durchgeführt werden. Aufgrund der beiden Technologieplattformen eines SAP-ERP-Systems, ABAP (Advanced Business Application Programming) und J2EE (Java 2 Enterprise Edition), unterteilen sich die Arbeiten vertikal in die jeweilige technologische Basis. In horizontaler Hinsicht gehen aus den Erkenntnissen der durchgeführten Performance-Messungen die Untersuchungen des Leistungsverhaltens eines SAP-ERP-Systems mittels Performance-Modellierung und Simulation hervor (siehe Abbildung 1-2).

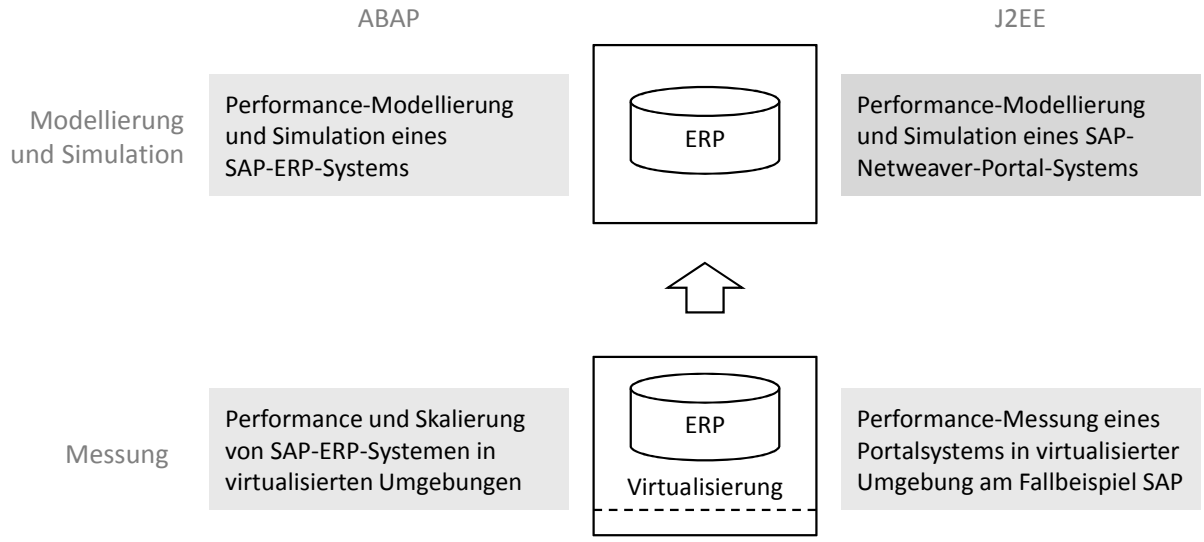


Abbildung 1-2: Arbeiten im Forschungsbereich „Performance-Evaluation von ERP-Systemen“
Quelle: eigene Darstellung

In beiden Arbeiten zur Performance-Messung wurden die Auswirkungen der Virtualisierung auf das Leistungsverhalten des jeweiligen SAP-Systems anhand eines kontrollierten Software-Experiments untersucht. Auf Seiten der Technologieplattform ABAP wurden durch Anwendung des synthetischen Benchmarks *Zachmanntest* (Boegelsack et al. 2011) Vergleichsmessungen durchgeführt (Boegelsack 2012). Darauf aufbauend wird derzeit ein Lastgenerator entwickelt, der über die Web-Service-Schnittstelle in der Lage ist, remotefähige Bausteine aufzurufen und Leistungsdaten aufzuzeichnen. Diese Daten werden für die Modellierung und Simulation eines SAP-ERP-Systems verwendet.

Im Java-Umfeld wurde für die Lasterzeugung und Aufzeichnung der Antwortzeiten das Performance-Evaluation-Cockpit für ERP Systeme (PEER) entwickelt (Jehle 2010, 131-148). Wenngleich die implementierten Funktionen zur Erfassung der Leistungsdaten in der vorliegenden Arbeit nicht ausreichen, da lediglich die Zeit aus Benutzersicht zwischen dem Absenden der Anfrage und dem Erhalt der Antwort festgehalten wird, kann das PEER-Rahmenwerk für die Lasterzeugung verwendet werden (vgl. „Applicable Knowledge“ in Abbildung 1-1).

1.6 Aufbau der Arbeit

Mit der Integration der Dienste und Anwendungen auf einer einheitlichen Plattform, dem Unternehmensportal, ist die Sicherstellung ausreichender Ressourcen von zentraler Bedeutung. Die Bewertung der Leistungsfähigkeit des Systems mittels Ex-Post-Analysen soll daher durch den Einsatz von Ex-Ante-Simulation ergänzt werden, um zukünftige Lastmuster und Lastintensitäten einschätzen und gegebenenfalls frühzeitig darauf reagieren zu können. Die vorliegende Arbeit hat das Ziel, mittels LQN ein Simulationsmodell für die Leistungsanalyse eines SAP-Netweaver-Portal-Systems zu erstellen, das heißt ein geeignetes Artefakt für die Unterstützung eines frühzeitigen Analyseprozesses bereitzustellen. Unter Anwendung des vorgestellten Design-Science-Prozesses von Pfeffers et al. (2006) ist diese Arbeit den einzelnen Phasen entsprechend strukturiert, um die in Kapitel 1.3 beschriebenen Forschungsfragen zu beantworten. Abbildung 1-3 stellt den Zusammenhang zwischen den sechs Kapiteln dieser Arbeit, die Bearbeitung der drei Forschungsfragen und den Phasen des Design-Science-Prozesses dar.

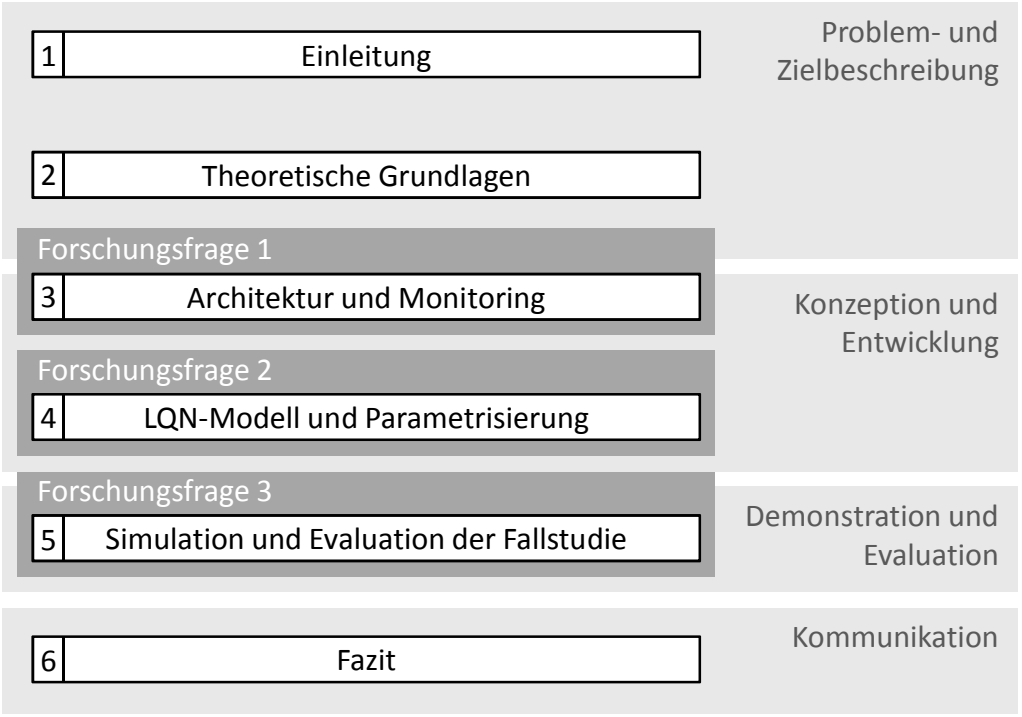


Abbildung 1-3: Aufbau der Arbeit in Bezug auf den Design-Science-Prozess nach Pfeffers et al. (2006)
Quelle: eigene Darstellung

In Kapitel 2 werden die theoretischen Grundlagen dieser Arbeit behandelt. Dabei soll ein Überblick über die Terminologie, die technische Infrastruktur des betrachteten Systems sowie diese Arbeit betreffende und benachbarte Themenfelder gegeben werden. Zu Beginn werden der Begriff Enterprise-Ressource-Planning (ERP) erläutert und über die geschichtliche Entwicklung Unternehmensportale entsprechend eingeordnet. Mit der Applikations- und Integrationsplattform Netweaver und dem inhärenten Bestandteil Netweaver-Portal wird anschließend auf die von SAP entwickelte Umsetzung für eine unternehmensweite Modellierung, Planung und Steuerung von Geschäftsprozessen eingegangen. Anschließend wird das Feld der Messtheorie vorgestellt, wobei vor allem auf die in den späteren Kapiteln verwendeten statistischen Kennzahlen Bezug genommen wird. In Abschnitt 2.3 wird das zentrale Forschungsfeld dieser Arbeit, die Performance-Evaluation von Rechnersystemen, aufgearbeitet. Die

gleichwertige Notwendigkeit der Festlegung der verwendeten Leistungskenngröße (auch Metrik genannt), dem Workload und der Methode der Leistungsbewertung bei der Analyse eines bestimmten Systems spiegelt sich in den entsprechenden Unterkapiteln wider. Neben der in dieser Arbeit eingesetzten Simulation von LQNs, einer speziellen Form von Warteschlangennetzen, wird ebenso auf alternative Methoden der Performance-Evaluation kurz eingegangen. Daraufhin wird dargestellt, welche Vergleichskriterien in Form von Benchmarks für die Leistungsbewertung von Rechnersystemen eingesetzt werden können. Den Abschluss des Grundlagenkapitels bildet ein kurzer Überblick über die Kapazitätsplanung, die sich mit der Frage beschäftigt, wann die Systemauslastung gesättigt ist und welche kostengünstigen Methoden den Sättigungspunkt am längsten hinauszögern.

Kapitel 3 widmet sich der Architektur und den Möglichkeiten zur Überwachung der Leistungsdaten eines SAP-Netweaver-Portal-Systems. Bei der Beschreibung der Architektur wird zu Beginn die Abarbeitung von Anfragen dargestellt und die Konfiguration beteiligter Elemente erläutert. Daraufhin wird die Funktionsweise der Systemkomponenten aufgezeigt, die in der Literatur und der Systemdokumentation als wesentliche Einflussgrößen auf die Antwortzeit identifiziert wurden. Ein Hauptaugenmerk wird dabei auf das Sperrmanagement von Datenbankobjekten, die Pufferung von Datenbanktabellen und das Java-Speichermanagement gelegt. Im zweiten Teil dieses Kapitels werden entsprechende Monitoring-Funktionen dargestellt, die die Erfassung und Aufzeichnung der Leistungsdaten ermöglichen und die Voraussetzung für eine akkurate Parametrisierung des angestrebten Performance-Modells bilden. Anschließend wird die Leistungsdatenerfassung auf Betriebssystemebene vorgestellt, die für die Analyse der Systemressourcen benötigt wird. Kapitel 3 beantwortet somit die in Forschungsfrage 1 gestellte Frage, wie ein SAP-Netweaver-Portal-System für die Performance-Modellierung und Simulation charakterisiert werden kann und welche Analyseinstrumente die benötigten Informationen bereitstellen.

In Kapitel 4 wird das zentrale Artefakt dieser Arbeit, das LQN-Modell und die entsprechende Parametrisierung, vorgestellt. Bevor jedoch die einzelnen Modellkomponenten detailliert erläutert werden, werden einige Grundannahmen getroffen, die sich direkt aus der Warteschlangentheorie ableiten. Anschließend werden die in der Literatur und der Systemdokumentation identifizierten Einflussfaktoren auf die Antwortzeit, sowie deren Möglichkeit zur Leistungsdatenerfassung, den zu modellierenden Komponenten zugeordnet. Mit dieser Zuordnung wird die Grundlage für das zu erstellende Modell geschaffen, welches das Leistungsverhalten eines SAP-Netweaver-Portal-Systems beschreiben soll. Im anschließenden Abschnitt über die Modellierung und Parametrisierung der Systemkomponenten werden die tatsächliche Umsetzung der Modellkomponenten beschrieben und Möglichkeiten zur akkuraten Parametrisierung, aber auch Limitationen der Berechnung unbekannter Quantitäten aufgezeigt. Kapitel 4 beantwortet somit die zweite Forschungsfrage, in der die Erstellung und Beschreibung des Performance-Modells gefordert ist.

In Kapitel 5 wird das erstellte Artefakt über eine Fallstudie demonstriert und evaluiert. Der Einsatz der Simulation des LQN-Modells sowie der Vergleich zwischen den Mess- und Simulationsergebnissen ist zugleich Gegenstand von Forschungsfrage 3. Dazu werden zwei Szenarien entwickelt, die eine unterschiedliche Systemkonfiguration beinhalten und in der Folge einen differenten Fokus auf den Untersuchungsgegenstand richten. Im ersten Szenario werden die Systemressourcen in ausreichendem Maße zur Verfügung gestellt, sodass die Analyse auf

die Einflussfaktoren und Architektureigenschaften des Portalsystems sowie dem daraus resultierenden Antwortzeitverhalten gerichtet werden kann. Das zweite Szenario dient der Prüfung der Antizipationsfähigkeit des Simulationsmodells bezüglich der Auswirkungen knapper Systemressourcen auf das Leistungsverhalten des Systems. Da im Hochlastbereich externe Einflüsse das Antwortzeitverhalten maßgeblich mitbestimmen, werden anschließend der System-Overhead und das Java-Speichermanagement untersucht und somit die Ursachen für die Divergenz zwischen simulierten und gemessenen Antwortzeiten bei zunehmender Systemlast untersucht.

Im sechsten und letzten Kapitel dieser Arbeit werden die Ergebnisse der Performance-Modellierung und Simulation eines SAP-Netweaver-Portal-Systems zusammengefasst und interpretiert. Die identifizierten Einflussgrößen auf die Antwortzeit werden den Resultaten der Leistungsbewertung gegenübergestellt und eine Einschätzung über die Akkuratesse der Performance-Evaluation mittels Modellierung und Simulation abgegeben. Dabei werden sowohl die Interessensgruppen aus der Praxis, als auch aus der Wissenschaft adressiert. Abschließend werden die Limitationen dieser Arbeit diskutiert und ein Ausblick auf weitergehende Forschungsmöglichkeiten gegeben.

2 Theoretische Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen, die in dieser Arbeit zum Einsatz kommen, vorgestellt. Die Beschreibung der einzelnen Themen und Terminologien beschränkt sich dabei nicht nur auf die in dieser Arbeit eingesetzten Methoden, sondern soll einen kurzen Überblick über die relevanten Themengebiete geben.

2.1 Enterprise-Resource-Planning am Fallbeispiel SAP

Ein Enterprise-Resource-Planning-System (ERP-System) stellt ein „unternehmensweites und Unternehmensfunktionen integrierendes System“ (Krcmar 2010, 236) dar, das „auf Basis standardisierter Module alle oder wesentliche Teile der Geschäftsprozesse eines Unternehmens aus betriebswirtschaftlicher Sicht informationstechnisch unterstützt“ (Görtz/Hesseler 2008, 5). Die standardisierten Module umfassen mengenorientierte Funktionen (zum Beispiel Vertrieb, Materialwirtschaft, Produktionsplanung, Fertigung, etc.) sowie wertorientierte Systeme (beispielsweise Rechnungs- und Finanzwesen) gleichermaßen (Gadatsch 2002, 212) und können auf die speziellen Anforderungen und Bedürfnisse des Unternehmens angepasst werden.

Aufgrund der Vielfältigkeit von ERP-Systemen unterscheiden sich auch die Definitionsvarianten des Begriffs. Dies ist auf inhaltliche Schwerpunkte sowie die historische Entwicklung von ERP-Software zurückzuführen. In den meisten Fällen wird ein ERP-System als Standardsoftware charakterisiert, das für eine große Anzahl von Kunden entwickelt wird und an die speziellen Anforderungen und Bedürfnisse eines Unternehmens individuell anpassbar ist. Somit lassen sich ERP-Systeme von Individualsoftware abgrenzen, die für ein bestimmtes Unternehmen zu einem bestimmten Anwendungszweck entwickelt wird (Schwarze 1997, 329).

Ein weiteres zentrales Merkmal ist die Integration. Galten früher ERP-Systeme als ein vorwiegend innerbetriebliches, integriertes Informationsverarbeitungssystem, berücksichtigen jüngere Definitionen ebenso Aspekte der zwischen- oder überbetrieblichen Integration (Wallace/Kremzar 2001, 10f.). Damit dieser neuen Generation von ERP-Systemen, die auch überbetriebliche Kollaboration ermöglichen soll, Rechnung getragen wird, werden sie in der Literatur oft auch als ERP-II-Systeme bezeichnet (Bond et al. 2000).

Ein Blick auf die historische Entwicklung von ERP-Systemen soll den identifizierten inhaltlichen Wandel weiter illustrieren. Als erste Generation betrieblicher Anwendungen und Vorgänger von ERP-Software gelten die in den sechziger Jahren entwickelten Material-Requirements-Planning-Systeme, kurz MRP-Systeme (Wannenwetsch/Nicolai 2004, 72). Sie ermöglichten eine bedarfsorientierte Materialdisposition und wurden im Laufe der Jahre um diverse Funktionen wie Beschaffung, Fertigung, Lager, Kapazitäts- und Terminplanung ergänzt. Dies führte zu den sogenannten MRP-II-Systemen (dem „Manufacturing Resource Planning“) (Mertens 2001, 141). Vereinzelt wird in der Literatur auch von MRP-III-Systemen (dem „Money Resource Planning“) gesprochen, wenn neben den mengenorientierten Größen auch wertorientierte Indikatoren berücksichtigt wurden (Mertens 2001, 142).

Parallel zu den Produktionsanwendungen wurde seit den sechziger Jahren das betriebliche Rechnungswesen in die elektronische Datenverarbeitung integriert. Mitte der achtziger Jahre trafen beide Entwicklungszweige zusammen und ergänzten das MRP-II-Konzept um Rechnungswesen, Einkauf und Personalwesen. Zu Beginn der neunziger Jahre wurde zur Beschreibung solcher Systeme der Begriff Enterprise-Resource-Planning geprägt und innerhalb kürzester Zeit wurde er zum Synonym für vollintegrierte, betriebliche Anwendungssysteme (Wannenwetsch/Nicolai 2004, 73). In Abbildung 2-1 ist der grundsätzliche Aufbau von ERP-Systemen dargestellt.

<table>
<tr><td>Produktion</td><td>Materialwirtschaft</td><td>Logistik</td></tr>
<tr><td>Personalwirtschaft</td><td rowspan="2">Datenbasis</td><td>Finanz-/Rechnungswesen</td></tr>
<tr><td>Vertrieb</td><td>Controlling</td></tr>
<tr><td colspan="3">Querschnittsanwendungen, z. B. Workflow, Archivierung, Reporting</td></tr>
</table>

Abbildung 2-1: Prinzipieller Aufbau eines ERP-Systems
Quelle: Schwarzer/Krcmar (2004)

Die Entwicklung von ERP-Systemen in ihrer heutigen Form war in den neunziger Jahren stark von der Erfolgsgeschichte von SAP[1] geprägt. Die Abkürzung SAP steht für „Systeme, Anwendungen, Produkte in der Datenverarbeitung“. SAP wurde 1976 von den ehemaligen IBM-Mitarbeitern Dietmar Hopp, Hans-Werner Hector, Hasso Plattner, Klaus Tschira und Claus Wellenreuther gegründet und ist weltweiter Marktführer für betriebswirtschaftliche Standardsoftware (Meissner 1999). So listet eine in 2011 durchgeführte ERP-Studie über die Marktanteile von ERP-Software in deutschen Industriebetrieben ab 50 Mitarbeiter (Konradin 2011, 23) SAP mit 48,1%, Microsoft Dynamics NAV+AX mit 21,5%, Infor ERP mit 9,0%, Oracle/Enterprise One/World mit 6,1% und noch einige kleinere Anbieter.

Das von SAP entwickelte System SAP R/2 fand vorwiegend in Großunternehmen eine hohe Verbreitung. Zu Beginn der neunziger Jahre, als sich die Client-Server-Architektur durchsetzte, löste SAP R/3 seinen großrechnerbasierten Vorgänger ab. Der durch die Abkehr von Mainframe-Systemen vollzogene Paradigmenwechsel ermöglichte eine signifikante Reduktion der Kosten, sodass ERP-Systeme auch für den gehobenen Mittelstand interessant wurden. Technische Notwendigkeiten, wie zum Beispiel das Jahr-2000-Problem oder die europäische Währungsunion, förderten zudem den Umstieg von Individuallösungen zu Standardsoftware (Gadatsch 2002, 210).

Ende der neunziger Jahre sowie Anfang dieses Jahrhunderts wurden nicht zuletzt durch technische Fortschritte und steigende Leistungsfähigkeit von Computersystemen ERP-Systeme auch für den mittleren sowie unteren Bereich des Mittelstandes wirtschaftlich (Gadatsch 2002, 210f.). Das Werben um die Gunst des Mittelstandes dauert bis heute an, da sich im Gegensatz zu Großunternehmen und dem oberen Segment der Mittelstandsunternehmen, bei

[1] SAP-Webseite: http://www.sap.com/germany/index.epx

denen SAP die Marktführerrolle seit Jahren übernommen hat, kein eindeutiger Marktführer herausbilden konnte. Jüngste Bestrebungen von SAP, kleine und mittlere Unternehmen (KMUs) zu erreichen, stellt sich in Form der Softwarelösung Business-by-Design (ByD) dar, die als Software-as-a-Service (SaaS) angeboten wird. Dabei wird die Software von SAP als Application-Service-Provider (ASP) zur Verfügung gestellt und von den Kunden angemietet. Der Unterschied zu dem klassischen ASP-Konzept liegt dabei vor allem bei der Entkopplung zwischen Servicekonsument und Applikation: Die Software wird nicht mehr in einer Eins-zu-Eins-Beziehung zwischen Anbieter und Kunde, sondern einer Vielzahl von Servicekonsumenten bereitgestellt (Krcmar 2010, 698).

Die technische Entwicklung wurde in den letzten Jahren vor allem durch das Internet und die damit verbundenen Standards und Technologien geprägt. So ermöglichen Unternehmensportale einen ortsungebundenen Zugang zu relevanten Informationen für Mitarbeiter, Kunden und Lieferanten. Aber auch Standards wie XML („eXtended Markup Language“) und Webservices treiben die Interoperabilität heterogener Systeme sowohl im inner- als auch im zwischenbetrieblichen Umfeld heran.

Abbildung 2-2 skizziert die Entwicklung von ERP-Systemen und ordnet Unternehmens-Portale entsprechend ein.

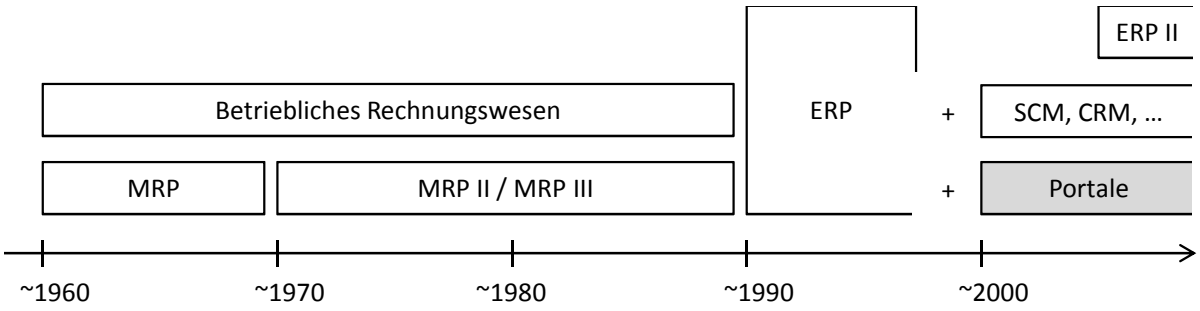


Abbildung 2-2: Entwicklung von ERP-Systemen sowie Einordnung der Unternehmensportale
Quelle: eigene Darstellung

2.1.1 SAP-Netweaver

Die genannten technischen Entwicklungen sowie die Berücksichtigung entstehender Geschäftsmodelle im Internet forderten eine unternehmensweite, service-orientierte Integrationsplattform, die bei SAP mit dem Namen *Netweaver* realisiert wurde. Ziel dieser Anwendungs- und Integrationsplattform ist es, eine unternehmensweite Modellierung, Planung und Steuerung von Geschäftsprozessen anzubieten und die Integration von bestehender Software in diese Prozesse zu verwirklichen. Durch das sogenannte Shared-Collaboration-Knowledge soll zudem die Wiederverwendbarkeit von Informationen und Services erhöht werden, indem diese an zentraler Stelle gespeichert und somit Entwicklungs- und Änderungsaufwand reduziert werden (Heilig/Karch 2007, 68). Die Architektur der SAP-Netweaver-Plattform ist in sechs Module unterteilt und in Abbildung 2-3 dargestellt.

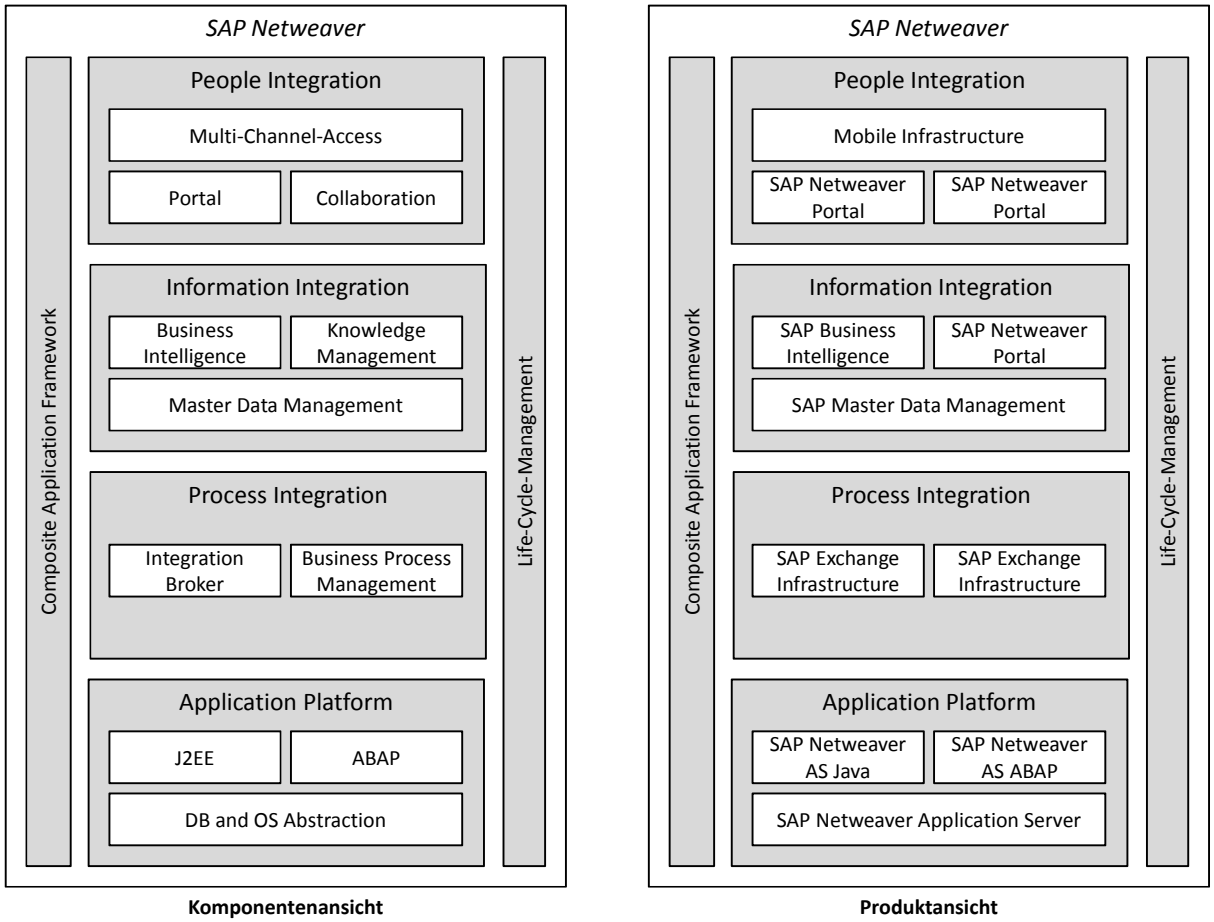


Abbildung 2-3: SAP-Netweaver-Komponentenansicht (links) und -Produktansicht (rechts)
Quelle: Bönnen/Herger (2007, 27)

Die „Application Platform" beinhaltet die Kernanwendungen, auf die die aufbauenden Module zurückgreifen können, wie beispielsweise Basisfunktionalitäten für den Datenaustausch, Monitoring, Benutzermanagement, etc. Die Netweaver-Plattform ist auf zwei technologischen Säulen (Stacks genannt) aufgebaut, und zwar dem ABAP-Stack und dem J2EE-Stack (SAP 2011b). Der ABAP-Stack rührt noch von der betrieblichen Standardsoftware SAP R/3 her und wird ständig weiterentwickelt. Der Java-Stack wurde im Zuge der bereits geschilderten Integration von gängigen Technologien eingeführt und wird im Kapitel über das SAP-Netweaver-Portal (siehe Kapitel 2.1.2) näher erläutert. Die Architektur der Anwendungsplattform, im Speziellen die 3-Schicht-Architektur, wird in Kapitel 2.1.1.1 dargestellt.

Die Komponente der „Process Integration" wird in das „Business Process Management" und den „Integration Broker" unterteilt. Das „Business Process Management" unterstützt die Modellierung und Automatisierung von Geschäfts- und Integrationsprozessen über Systemgrenzen hinweg. Der „Integration Broker" ist für die Konvertierung und Zustellung der verschiedenen Nachrichtenformate zuständig und ist somit mit einem Service-Bus zu vergleichen (SAP 2011j).

Der Bereich der „Information Integration" umfasst die Bereiche „Master Data Management", „Knowledge Management" und „Business Intelligence". Während das „Master Data Management" eine zentrale Stammdatenverwaltung darstellt, bietet das Modul „Business Intelligence" Funktionen zur Analyse von strukturierten Datenbeständen. Unstrukturierte Daten

werden im Modul „Knowledge Management“ organisiert und vom SAP-Netweaver-Portal übernommen (SAP 2011i).

Die „People Integration“ bietet über den „Multi Channel Access“ die Möglichkeit zur Integration von mobilen Endgeräten. Der Begriff „Collaboration“ steht für die Zusammenfassung von Funktionen wie Terminkalender oder Email (SAP 2011l). Das bedeutendste Modul in diesem Bereich ist allerdings das SAP-Netweaver-Portal. Mittels eines webbasierten Front-Ends werden Daten und Funktionen dem Benutzer zur Verfügung gestellt und der Zugriff darauf gesteuert. Das SAP-Netweaver-Portal ist somit eines der zentralen Komponenten im Netweaver-Stack, mit denen der Benutzer interagiert (Bönnen/Herger 2007, 46) und wird in Kapitel 2.1.2 näher erläutert.

Mit dem „Life Cycle Management“ werden Funktionen zur Verfügung gestellt, die den Lebenszyklus einer Anwendung begleiten. Die wichtigste Komponente in diesem Bereich stellt der SAP-Solution-Manager dar (SAP 2011n).

Die sechste und letzte Komponente des Netweaver-Stacks, das „Composite Application Framework“, dient zur Unterstützung der Entwickler und stellt eine Reihe von Hilfsmitteln zur Verfügung, wie zum Beispiel zur Modellierung von Prozessen oder zur Gestaltung von grafischen Benutzeroberflächen (SAP 2011d).

2.1.1.1 Architektur

Seit der Einführung des R/3-Systems setzt SAP auf eine 3-Schicht-Architektur, die die Datenbank von der Applikations- und der Präsentationsschicht trennt. Diese Trennung ermöglicht nicht nur eine individuelle Spezialisierung der einzelnen Komponenten, sondern bietet auch die Grundlage für eine höhere Skalierbarkeit (Jehle 2010, 13f.). Die drei Schichten unterteilen sich in:

1. *Präsentationsschicht*:

   Die Präsentationsschicht bildet die oberste Ebene und dient der Kommunikation mit dem Benutzer. Dazu gehören die benutzerfreundliche Aufbereitung und Darstellung der Informationen und die Entgegennahme und Weiterleitung von Benutzeraktionen an die Anwendungsschicht. Für die Präsentation kommen vorwiegend zwei Typen von Clients zum Einsatz, zum einen das „SAP Graphical User Interface“ (SAP-GUI) und zum anderen der Web-Browser auf dem Client. Darüber hinaus bietet die „SAP Mobile Infrastructure“ auch für mobile Endgeräte Zugriff auf die Applikationsschicht.

   Bei dem SAP-GUI kommt das proprietäre DIAG-Protokoll („Dynamic Information and Action Gateway“) zum Einsatz, das für jeden Dialogschritt nur wenige Kilobytes an Daten an die Applikationsschicht überträgt. Aufgrund des nicht offengelegten Kommunikationsprotokolls ist es nicht möglich, über einen alternativen Client, der zum Beispiel spezielle Funktionen für die Leistungsanalyse von SAP-Systemen bereitstellt, in derselben Art und Weise mit der Applikationsschicht zu kommunizieren. Es werden zwar Schnittstellen angeboten, über die die Benutzereingaben automatisiert gesteuert werden können, allerdings spiegeln diese nicht exakt die Abarbeitung der Aufträge durch das konventionelle SAP-GUI wider. Nach Jehle (2010, 18) führen die-

se alternativen Schnittstellen zu einem erhöhten Ressourcenverbrauch, der in der Folge Messergebnisse verfälschen kann.

Im Gegensatz zu dem SAP-GUI besteht die Präsentationsschicht des J2EE-Stacks aus einem nicht modifizierten Web-Browser. Es wird das im Internet standardmäßig verwendete HTTP-Protokoll ("Hyper Text Transport Protocol", Fielding et al. 1999) verwendet.

2. *Applikationsschicht*:

   Die Applikationsschicht wird in einen betriebssystemspezifischen und in einen unabhängigen Teil untergliedert. Der betriebssystemspezifische Teil wird als Operating-System-Abstraktion (OS-Abstraktion) bezeichnet und enthält zum Beispiel den Kernel der SAP-Anwendungsplattform. Der betriebssystemunabhängige Teil der Applikationsschicht stellt die Umgebung dar, in der alle Anwendungen, abgeschottet von den Besonderheiten des zugrundeliegenden Betriebssystems, verwaltet und ausgeführt werden.

   Die Anwendungen können sowohl ABAP als auch J2EE als technologische Grundlage verwenden. Der Java-Bereich der Applikationsschicht bietet eine J2EE-zertifizierte Laufzeitumgebung, in der auch der Zugriff auf ABAP-basierte Anwendungsobjekte möglich ist. Zudem wird eine Reihe von Kommunikationsstandards unterstützt, wie beispielsweise Web-Services, die die Entwicklung von Geschäftsprozessen, die mehrere Geschäftspartner mit unterschiedlichen Anwendungssystemen integrieren, ermöglichen.

   Für den Zugriff auf die Daten, die in der Datenbankschicht vorgehalten werden, ist eine ständige Verbindung zwischen der Applikationsschicht und der Datenbankschicht notwendig. Somit ist das Leistungsverhalten der Applikationsschicht auch von der darunter liegenden Datenbankschicht abhängig. Damit diese Abhängigkeit verringert werden kann, werden die zu verarbeitenden Daten in der Applikationsschicht (in der Regel im Hauptspeicher) zwischengespeichert.

3. *Datenbankschicht*:

   Die Datenhaltung erfolgt in der Datenbankschicht, auch Persistenzschicht genannt. Beim SAP-Netweaver-Application-Server-Java (kurz Netweaver-AS-Java) kann dies eine völlig autonome Datenbankanbindung, aber auch ein eigenes Schema in der Datenbank sein. Die dort gespeicherten Daten werden der Applikationsschicht zur Verfügung gestellt.

   Das Ansprechen der (relationalen) Datenbank erfolgt über die „Structured Query Language“ (SQL, Chamberlin/Boyce 1974). Java-Anwendungen verwenden dabei üblicherweise die JDBC-API („Java Database Connectivity – Application Programming Interface“) als Kommunikationsschnittstelle. Aufbauend auf die JDBC-API hat SAP eine Open-SQL-Schicht implementiert, wie sie in ähnlicher Weise auch beim SAP-Netweaver-AS-ABAP verwendet wird. In Bezug auf die Leistungsverbesserung sowie -analyse werden Funktionen wie Tabellenpufferung, Wiederverwendbarkeit von häu-

fig benutzten SQL-Anweisungen („Statement Pooling“) und datenbankunabhängige Monitor- und SQL-Trace-Unterstützung geboten.

Damit die zugrundeliegende Datenbank ohne Probleme ausgetauscht werden kann, verwendet ein SAP-System nur Basisfunktionen der unterstützten relationalen Datenbanksysteme. Beispielsweise werden keine Bedingungen zur referenziellen Integrität eingerichtet. Somit wird die Datenbank hauptsächlich als ein leistungsfähiger Hintergrundspeicher verwendet. Die Ressourcennutzung ist infolgedessen stark I/O-lastig. Der Einfluss der Datenbankschicht sollte daher getrennt von der Leistungsfähigkeit eines SAP-Systems betrachtet werden (Jehle 2010, 14).

Neben den relationalen Datenbanken wird inzwischen vermehrt mit In-Memory-Technologien experimentiert, wo die Daten im Hauptspeicher vorgehalten werden und somit einen wesentlich schnelleren Zugriff erlauben. Die von SAP entwickelte In-Memory-Datenbank SAP-HANA ("High Performance Analytic Appliance", SAP 2012) wurde im Frühjahr 2010 vorgestellt und soll Echtzeitberechnungen auch im Bereich von ERP-Systemen ermöglichen. Die Leistungsanalyse von In-Memory-Datenbanken ist nicht Teil dieser Arbeit, der modulare bzw. geschichtete Aufbau des Warteschlangenmodells ermöglicht es jedoch, lediglich die Datenbankschicht gegen ein entsprechendes Untermodell auszutauschen.

2.1.1.2 Installationsvarianten

Größere SAP-Systeme, die bis zu mehrere tausend Benutzer bedienen, werden meist als dreistufige Systeme konfiguriert. Dabei werden die Datenbank-, Applikations- und Präsentationsschicht jeweils auf unterschiedlichen Rechnern installiert. Die 3-Schicht-Installation bietet die größte Flexibilität, die Server-Systeme dynamisch an veränderte Nutzungsbedingungen anzupassen, da der Anwendungsschicht ohne größeren Aufwand weitere Server hinzugefügt werden können und das System daher sehr gut skaliert. Zusätzlich ist es möglich, durch Datenbank-Cluster die Datenbank auf mehrere Server zu verteilen.

Zweistufige Systeme, bei denen zwar die Präsentationsschicht entkoppelt ist, die Applikations- sowie Datenbankschicht aber auf einem Server liegen, findet man häufig bei kleineren SAP-Systemen, die oft als Entwicklungs-, Schulungs- oder Testsysteme eingesetzt werden. Einstufige Systeme werden in der Praxis fast ausschließlich zu Demonstrationszwecken eingesetzt. Bei bestimmten Testfällen kann es zudem nützlich sein, keine zusätzlichen Übertragungsmedien beachten zu müssen.

Abbildung 2-4 zeigt die verschiedenen Installationsvarianten in Bezug auf die 3-Schicht-Architektur von SAP-Systemen.

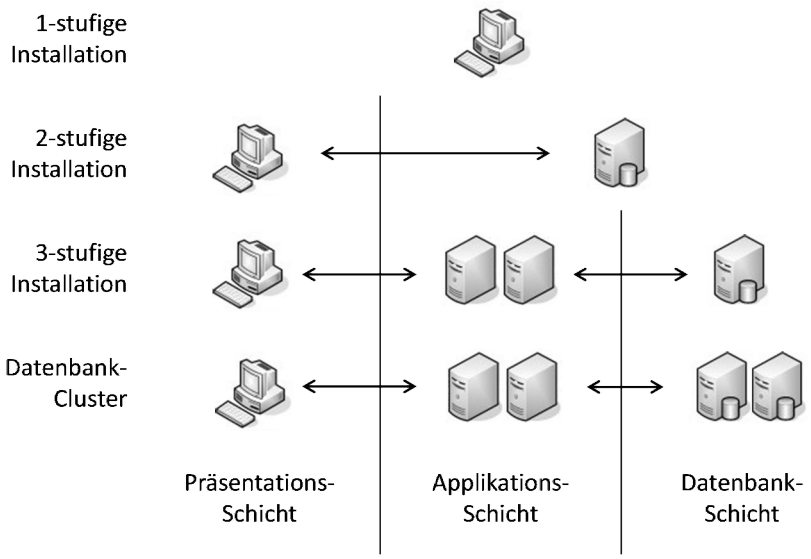


Abbildung 2-4: 3-Schicht-Architektur und Installationsvarianten
Quelle: eigene Darstellung

2.1.1.3 Ressourcennutzung

Eine von Jehle (2010, 15f.) durchgeführte Erhebung der Ressourcennutzung lässt bereits eine grundsätzliche Einordnung der einzelnen Schichten bzw. Technologien eines SAP-ERP-Systems hinsichtlich der CPU-, Hauptspeicher- und I/O-Nutzung zu. Die Einordnung dient lediglich als grobe Übersicht, welche Ressource von welchen Komponenten besonders stark beansprucht wird.

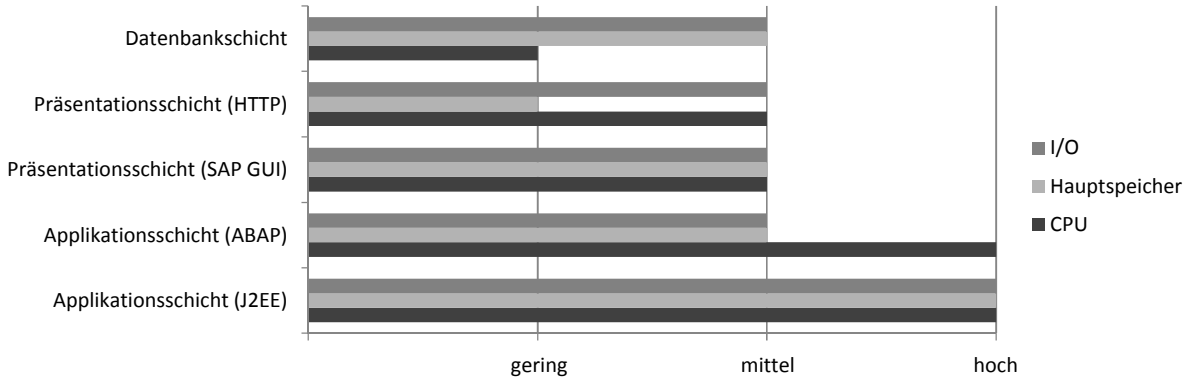


Abbildung 2-5: Ressourcenbedarf nach Schichten
Quelle: eigene Darstellung in Anlehnung an Jehle (2010)

Wie man in Abbildung 2-5 erkennen kann, ist der Ressourcenbedarf bei der Applikationsschicht im Java-Umfeld besonders hoch. Die I/O-Ressourcen werden allerdings nur dann stark beansprucht, wenn die entsprechende Portal-Applikation viele Zugriffe auf die Datenbasis aufweist. In vielen Fällen aber, beispielsweise bei den in dieser Arbeit modellierten portalinternen Funktionen, kann von einem hauptspeicher- und CPU-intensiven Lastmuster ausgegangen werden.

2.1.2 SAP-Netweaver-Portal

Das in dieser Arbeit betrachtete SAP-Netweaver-Portal fällt in die Kategorie der Unternehmensportale, einer speziellen Form der Portale. Portale im Allgemeinen werden nach Nicolescu/Klappert/Krcmar (2007, 18) als Anwendungssysteme bezeichnet, die „[…] durch die Integration von Anwendungen, Prozessen und Diensten sowie durch die Bereitstellung von Funktionen zur Personalisierung, Sicherheit, Suche und Präsentation von Informationen gekennzeichnet [sind]“.

Großmann/Koschek (2005, 32) definieren Unternehmensportale als „[…] ein geschlossenes Portal, das den Anwendern einen individuellen, personalisierbaren Zugang zu allen relevanten Inhalten bietet, um alle Aufgaben bequem und schnell erledigen zu können. Dieser Zugang muss jederzeit und überall auf sicherem Weg erreichbar sein. Zu den Anwendern eines Unternehmensportals zählen die Mitarbeiter des Unternehmens, aber auch die Kunden, Lieferanten oder Partner.“

Das SAP-Netweaver-Portal war das erste J2EE-basierte Produkt bei SAP und kann als zentrale Applikation zur Integration heterogener Anwendungen genutzt werden. Die Integration der Daten unterschiedlicher Quellapplikationen geschieht typischerweise über sogenannte Portlets, die in der SAP-Terminologie *iViews* genannt werden (Popp 2002, 23). Diese dynamische Integration wird mittels Drag-and-Relate-Technologie (Jay 2008, 6) realisiert, die logische Verknüpfungen zwischen Objekten von unterschiedlichen Applikationen erstellt. So kann beispielsweise die Bestellung im ERP-System zu dem entsprechenden Kundensatz im Customer-Relationship-Management-System (CRM-System) gezogen werden.

Die grundsätzliche Architektur des SAP-Netweaver-Portals ist in Abbildung 2-6 dargestellt und basiert auf den folgenden Kernkomponenten:

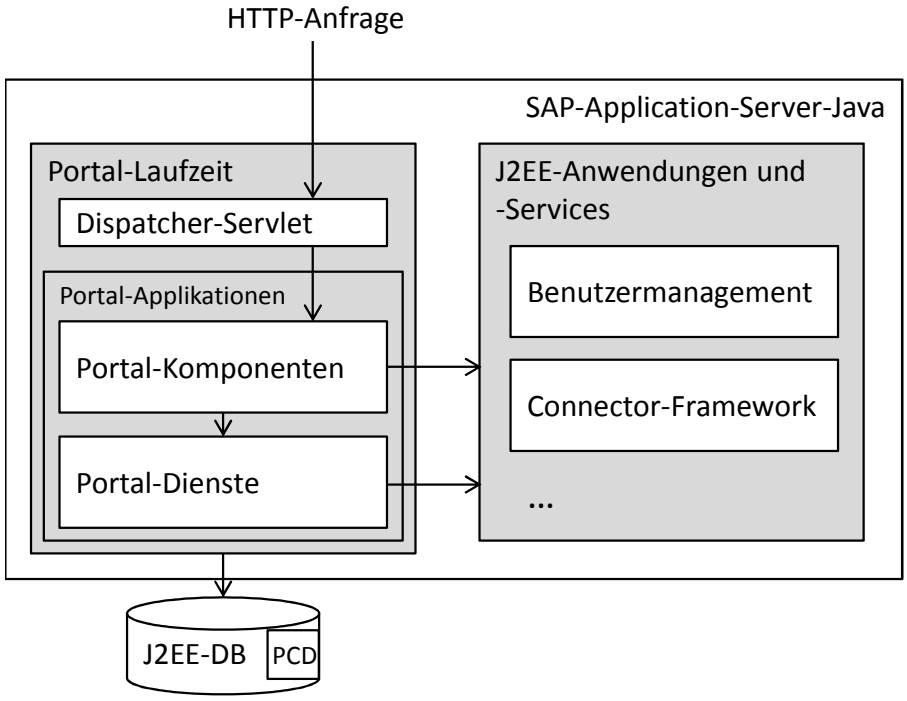


Abbildung 2-6: Portal-Architektur
Quelle: eigene Darstellung in Anlehnung an SAP (2011m)

- Die Portal-Laufzeit stellt die Laufzeit für die Portal-Komponenten und Portal-Dienste zur Verfügung. Die eingehenden HTTP-Anfragen werden über ein Dispatcher-Servlet an die Portal-Komponenten weitergereicht.

- Portal-Applikationen bestehen aus einer Sammlung aus bereits zur Verfügung gestellten Kernapplikationen (zum Beispiel zur Darstellung von Navigationsleisten oder i-Views) sowie optionalen Eigenentwicklungen. Dabei wird zwischen den bereits genannten Portal-Komponenten und Portal-Diensten unterschieden. Letztere werden den Portal-Components zur Verfügung gestellt (beispielsweise dem Java-Naming-and-Directory-Provider (JNDI-Provider) für den Zugriff auf das Portal-Content-Directory).

- Das Portal-Content-Directory (PCD) beinhaltet semantische Objekte wie zum Beispiel iViews.

- Die J2EE-Laufzeit-Umgebung stellt weitere Applikationen bereit, wie zum Beispiel das Benutzermanagement oder das Connector-Framework.

Auf die Architektur des Portalsystems bzw. des SAP-Netweaver-AS-Java sowie der Mittel zur Leistungsanalyse wird in Kapitel 3 detailliert eingegangen.

Wie eingangs bereits detailliert dargestellt, setzt sich diese Arbeit für die Performance-Evaluation von SAP-Netweaver-Portal-Systemen zum Ziel, einen Ansatz zur Modellierung und Simulation eines SAP-Netweaver-Portals vorzustellen. Aufgrund der Nutzung des HTTP-Protokolls ist eine automatisierte Lasterzeugung, die Aufträge an die Applikationsschicht schickt, möglich und bildet somit die Grundlage für die Messung der Leistungsdaten. Das Profil der Ressourcennutzung aus Kapitel 2.1.1.3 zeigt, dass der Ressourcenverbrauch hauptsächlich im Bereich des Hauptspeichers und der CPU liegt, sodass der Verzicht auf eine feingranulare Modellierung der Datenbank keinen starken Einfluss auf die Genauigkeit der Antwortzeitprognose nimmt.

2.2 Messtheoretische Grundlagen

Die Messtheorie ist kein zentraler Gegenstand dieser Arbeit, allerdings stellt sie die Grundlage der durchgeführten Messungen am Portalsystem dar. Die Messungen erfolgen dabei aus zwei Gründen: Zum einen werden verschiedene Messreihen zur Parametrisierung des Warteschlangenmodells verwendet, zum anderen dienen die Messungen am realen Testsystem zum Validieren bzw. Vergleichen der Simulationswerte. Die Messung im spezifischen Kontext der Performance-Evaluation von Rechnersystemen wird in Kapitel 2.3.3.1 erläutert.

Nach Finkelstein (1984) ist Messung als der Prozess definiert, der Zahlen oder Symbole den Attributen von Entitäten in der realen Welt unter Einhaltung klar definierter Regeln zuordnet. Eine Entität kann ein Objekt, wie beispielsweise eine Person oder eine Software-Spezifikation, aber auch ein Ereignis, wie zum Beispiel die Testphase eines Softwareprojekts, sein. Ein Attribut ist eine Eigenschaft der Entität, wie zum Beispiel die Größe (Höhe) einer Person oder die Dauer der Testphase.

Bortz/Döring (1995, 65) definieren Messen als „[…] eine Zuordnung von Zahlen zu Objekten oder Ereignissen, sofern diese Zuordnung eine homomorphe Abbildung eines empirischen Relativ in ein numerisches Relativ ist“. Dafür sind folgende Aspekte von Bedeutung:

- Es erfolgt eine Zuordnung von Zahlen zu Merkmalen von Objekten (den Merkmalsträgern).
- Die Relationen der Merkmalsträger bezüglich der Merkmalsausprägung sollen durch die ihnen zugeordneten Zahlen wiedergegeben werden.
- Der Messvorgang stellt die Konstruktion eines numerischen Relativs zu einem gegebenen empirischen Relativ dar.

Ein empirisches Relativ besteht aus einer Menge von n Objekten, zwischen denen eine oder mehrere Beziehungen bestehen (Relationssystem). Die Zuordnung des numerischen zum empirischen Relativ ist homomorph, das heißt die Beziehung der empirischen Objekte ist in der numerischen Struktur so abgebildet, dass die Eigenschaften erhalten bleiben (Strukturgleichheit). Dieser Zusammenhang ist in Abbildung 2-7 zusammenfassend dargestellt.

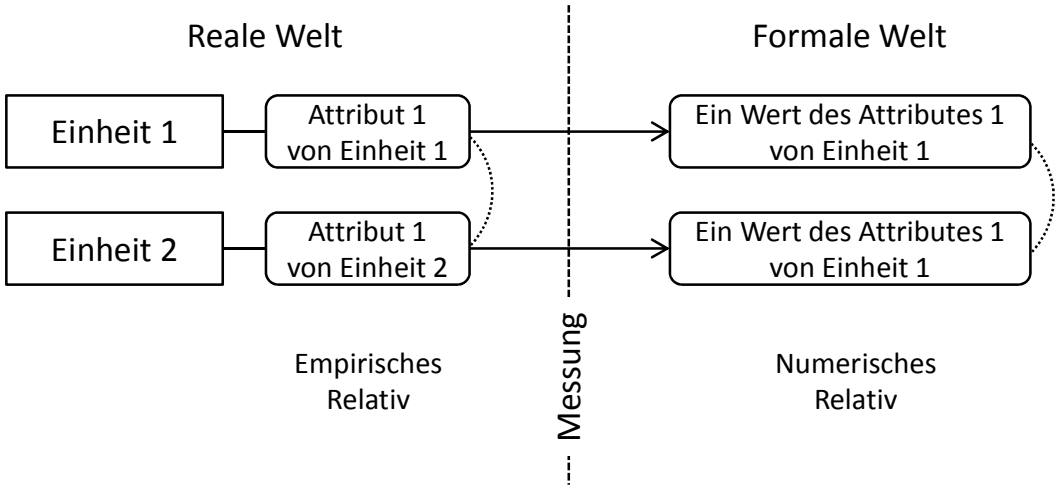


Abbildung 2-7: Relationssystem in der Messtheorie
Quelle: eigene Darstellung

Die Regeln, mit denen die Transformation der verschiedenen Attribute in numerische Werte durchgeführt wird, werden in einer Skala festgelegt. Bortz/Döring (1995, 65) definieren eine Skala als „[…] ein empirisches Relativ, ein numerisches Relativ und eine die beiden Relative verknüpfende, homomorphe Abbildungsfunktion. Die Messbarkeit eines Merkmals bzw. die Konstruierbarkeit einer Skale ist an Bedingungen geknüpft.“

Es gibt verschiedene Skalentypen, die durch die jeweils auf ihnen zulässigen Transformationen definiert werden. Abhängig von den zulässigen Transformationen sind nur bestimmte statistische Berechnungsmöglichkeiten anwendbar. Die in Tabelle 2-1 aufgelisteten Skalenniveaus bauen aufeinander auf, das heißt, nachfolgend genannte Skalen beinhalten implizit auch alle Eigenschaften der vorher genannten.

Skalenniveau		Eigenschaft	Erlaubte Operationen	
			mathematisch	stochastisch
qualitativ	Nominalskala	Reine Kategorisierung von Werten	$=, \neq$	Modus, Frequenz
	Ordinalskala	Skalenwerte geordnet und vergleichbar	$=, \neq, <, >$	Median, Perzentil
quantitativ	Intervallskala	Werte geordnet, Distanzen bestimmbar.	$=, \neq, <, >, +, -$	Arithmetisches Mittel, Standard-Abweichung
	Verhältnisskala	Werte geordnet, Skala hat einen absoluten Nullpunkt	$=, \neq, <, >,$ $+, -, *, \div, \%$	Geometrisches Mittel
	Absolutskala	Skalenwerte sind absolute Größen	$=, \neq, <, >,$ $+, -, *, \div, \%$	

Tabelle 2-1: Skalentypen
Quelle: eigene Darstellung in Anlehnung an Lilienthal (2008)

Mittels Maßvalidierung wird sichergestellt, dass die numerische Charakterisierung des betrachteten Merkmals gültig ist, das heißt, dass zuverlässige Messwerte geliefert werden können. Sie umfasst zwei Schritte, eine interne und eine externe Validierung. Bei der internen Validierung wird zuerst überprüft, ob die Grundsätze der Messtheorie eingehalten werden. Über die externe Validierung wird festgestellt, ob die zugrunde liegende Fragestellung durch die erhobenen Messwerte und ihre Interpretationen beantwortet wird (Zuse 1998, 71f.).

2.2.1 Statistische Kennzahlen

Nachstehend werden die für diese Arbeit relevanten, grundlegenden statistischen Kennzahlen zusammenfassend aufgeführt. Vorab seien folgende Grundbegriffe beschrieben:

- *Stichproben* sind Teilerhebungen aus der Grundgesamtheit, da es oft nicht möglich ist, die Grundgesamtheit, also die Menge aller potentiellen Untersuchungsobjekte einer Fragestellung, zu untersuchen. Die Stichprobe soll repräsentativ für die Grundgesamtheit sein.

- Ein *Parameter* stellt eine Eigenschaft der Grundgesamtheit dar, die der Beschreibung der Verteilung der Einheiten dient. Beispielsweise ist der Mittelwert ein Parameter.

2.2.1.1 Maße der zentralen Tendenz

Mittelwerte können als Maße der zentralen Tendenz bezeichnet werden, da sie den typischen, zentralen oder durchschnittlichen Wert der Beobachtungswerte (Stichprobe) beschreiben (Schulze 2007, 35). Für die quantitative Erfassung existieren verschiedene Maße:

- *Arithmetisches Mittel*: Das arithmetische Mittel $\bar{x}$ bildet den Durchschnitt der Beobachtungswerte $x_1 \dots x_n$:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Neben dem arithmetischen Mittel sind das geometrische und harmonische Mittel weitere Durchschnittsparameter.

- *Median*: Der Median, auch als Zentralwert bezeichnet, stellt die mittlere Zahl in einer sortierten Liste der erhobenen Werte dar. Ist die Anzahl der Werte gerade, so wird aus den beiden zentralen Elementen der Durchschnitt gebildet.

- *Modus* oder *Modalwert*: Der Modus oder Modalwert ist der Wert, der am häufigsten vorkommt.

2.2.1.2 Streuungsmaße

Maße der zentralen Tendenz vermögen zwar Informationen über das Zentrum zu geben, enthalten aber keine Informationen darüber, wie weit die Beobachtungswerte von diesem Zentrum entfernt sind (Schulze 2007, 64).

Spannweite

Die Spannweite R ist der einfachste Streuungsparameter und ist definiert als die Differenz zwischen dem größten und dem kleinsten vorkommenden Beobachtungswert:

$$R = x_{max} - x_{min}$$

Der größte Nachteil der Spannweite und somit eine Einschränkung der Aussagekraft ist, dass sie nur aus zwei Werten gebildet wird und im Falle von Ausreißern, also untypischen Minimal- bzw. Maximalwerten, nicht zu repräsentativen Ergebnis führt.

Quartile und Quantile

Unterteilt man die Beobachtungswerte in vier gleich große Abschnitte, dann sind die Werte an den drei Schnittstellen die Quartile. Das zweite Quartil (50%) ist folglich der Median. Die Differenz zwischen dem ersten und dritten Quartil wird Interquartilsabstand genannt und stellt somit ein Streuungsmaß dar. Da die Unterteilungen oft variieren (beispielsweise zehn Dezile oder hundert Perzentile) werden die Abschnitte im Allgemeinen Quantile genannt.

Standardabweichung und Varianz

Zu den bekanntesten Streuungsmaßen zählen die Standardabweichung und die Varianz. Die Varianz beschreibt den durchschnittlichen quadratischen Abstand der Beobachtungswerte zum arithmetischen Mittel:

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Die Standardabweichung wird entsprechend aus der Quadratwurzel der Varianz gebildet. Da bei Stichproben, die relativ klein sind im Verhältnis zur Grundgesamtheit, die Division durch die Gesamtzahl der Beobachtungen tendenziell eine zu niedrige Schätzung der Standardabweichung ergibt, zeigt sich, dass der Faktor $N - 1$ diese Unterschätzung korrigiert (Berman 2008):

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N-1}}$$

Variationskoeffizient

Der Variationskoeffizient c_X einer Messreihe $x_1, \ldots, x_n$ ist definiert als das Verhältnis zwischen der Standardabweichung (σ) und dem arithmetischen Mittel ($\bar{x}$):

$$c_X = \frac{\sigma}{\bar{x}} \qquad \text{für } \bar{x} \neq 0$$

Im Gegensatz zur Varianz (σ^2) ist der Variationskoeffizient ein relatives Streuungsmaß, das heißt er hängt nicht von der Maßeinheit der Variablen ab. Er ist mit anderen Worten eine Normierung der Varianz. Falls die Standardabweichung größer als der Mittelwert ist, weist der Variationskoeffizient einen Wert > 1 auf.

2.2.1.3 Parameterschätzung

Will man bei einer statistischen Untersuchung die Parameter einer Grundgesamtheit genau berechnen, muss jede Einheit der Grundgesamtheit bei der Berechnung erfasst werden. Da dies jedoch oft unmöglich ist, werden auf Basis von Stichproben Näherungswerte der entsprechenden Parameter, sogenannten Punktschätzer, berechnet. Dabei weiß man jedoch nicht, ob diese von dem tatsächlichen Wert abweichen und falls ja, wie groß der Fehlbetrag ist.

Anstelle eines Punktschätzers kann nun ein Intervall um den Messwert einer Stichprobe berechnet werden, für das gilt, dass der tatsächliche Parameterwert zu einem bestimmten Vertrauensgrad in diesem Bereich liegt. Dieses Verfahren bezeichnet man als Intervallschätzung.

Konfidenzintervall normalverteilter Stichproben

Liegt die Stichprobengröße bei 30 oder mehr, kann aufgrund des Zentralen Grenzwertsatzes von einer Normalverteilung ausgegangen werden (vgl. bspw. Zwerenz 2009, 343) und mit Hilfe des z-Werts das Konfidenzintervall berechnet werden. Der z-Wert ergibt sich aus dem geschätzten Mittelwert $\bar{x}$ und dem (unbekannten) Mittelwert der Grundgesamtheit (μ) und wird durch den Standardfehler geteilt (Berman 2008):

$$z = \frac{\bar{x} - \mu}{\frac{\sigma}{\sqrt{N}}}$$

Damit ein Konfidenzintervall für den Mittelwert μ aufgestellt werden kann, muss der gewünschte Vertrauensgrad angegeben werden, damit der dem z-Wert entsprechende Anteil ermittelt werden kann. Die Berechnung des Konfidenzintervalls folgt schließlich aus einer Umformung der Gleichung für den z-Wert:

$$Konfidenzintervall(\mu) = \bar{x} \pm (|z - Wert| \times Standardfehler)$$

Der Wert $(|z - Wert| \times Standardfehler)$ wird auch als Fehlertoleranz der Stichprobe bezeichnet.

2.2.2 Messfehler

Bei jeder Messung muss damit gerechnet werden, dass der Messwert aufgrund von Störgrößen vom eigentlichen Wert abweicht. Daher ist eine Fehlertheorie von Nöten, die sich „[…] mit der Schätzung des wahren Wertes aus einer Reihe von fehlerbehafteten Messwerten, mit dem Fehler dieser Schätzung und der Fehlerfortpflanzung bei Verwendung in arithmetischen Ausdrücken [beschäftigt]“ (Jaenecke 1982).

Grundsätzlich wird zwischen zufälligen und systematischen Fehlern unterschieden. Der Unterschied zwischen den beiden Fehlerarten liegt darin, dass systematische Fehler das zuvor aufgestellte Axiom falsifizieren. Der Schwellwert, ab welcher Fehlerrate ein systematischer Fehler vorliegt, muss versuchsbezogen festgelegt werden. Die Fehlertheorie bestimmt somit die Kriterien, in welchem Rahmen ein Axiom empirisch zu bestätigen und ab wann es zu verwerfen ist (Orth 1974, 91).

Nach Jehle (2010, 37) genügt bei der Leistungsanalyse von Portalsystemen primär die Betrachtung, ob Operationen vollständig abgeschlossen wurden. Für die Fehlerdefinition werden, abhängig von den ausgewählten Attributen im Sinne der Messtheorie, Erwartungswerte definiert. So kann beispielsweise das numerische Relativ Antwortzeit innerhalb eines gewissen Rahmens, aber auch außerhalb liegen und somit als Fehler definiert werden. Die fehlertheoretische Analyse beschäftigt sich anschließend mit der Frage, wie viele Verfehlungen der Sollwerte akzeptabel sind.

2.2.2.1 Ausreißer

Ausreißer kommen häufig durch unerwartete externe Einflüsse (wie z.B. Übertragungsfehler im Netzwerk) zustande. Ein möglicher Ansatz zur Definition des tolerierten Rahmens stellt die Schwellwertangabe des Konfidenzintervalls dar. Werte, die außerhalb dieses Rahmens liegen, werden als Ausreißer klassifiziert und aus der Messreihe eliminiert (Jehle 2010, 165).

Wie im Verlauf dieser Arbeit noch gezeigt wird, können jedoch auch architekturspezifische Eigenschaften zu vermeintlichen Messfehlern führen (siehe bspw. Einflüsse des Garbage-Collectors auf die Antwortzeit in Kapitel 4.2.6 und 5.8). Die grundsätzliche Elimination dieser Werte würde zu einer zu optimistischen Einschätzung des Antwortzeitverhaltens führen. Daher ist eine individuelle Plausibilitätskontrolle einer automatisierten Elimination vorzuziehen.

2.2.2.2 Konsistenz- und Glaubwürdigkeitsprüfung

Die Plausibilitätskontrolle der Messwerte verfolgt das Ziel, unerwartete Werte zu untersuchen und mögliche Fehler zu identifizieren. Dazu wird eine Konsistenzprüfung sowie eine Glaubwürdigkeitsprüfung der Daten durchgeführt (Prechelt 2001, 176f.). Die Konsistenzprüfung prüft die logische Korrektheit der erhobenen Messwerte. Im Falle der Antwortzeiten könnte ein Kriterium eine Antwortzeit > 0 fordern. Die Glaubwürdigkeitsprüfung stellt die Frage zur Plausibilität der erhobenen Messwerte. Bei auffallenden Werten führt dies zu einer Untersuchung der möglichen Gründe. Beispielsweise kann ein auffallend hoher Wert durch die Aktivität des Garbage-Collectors als plausibel eingestuft werden, eine Netzwerkstörung jedoch zu einem nicht plausiblen Wert führen.

2.3 Performance-Evaluation von Rechnersystemen

Die Komplexität moderner Computersysteme steigt aufgrund immer größer werdender Funktionalität. Doch nicht nur die Komplexität der Systeme an sich, sondern auch die zugrundeliegenden Architekturen erfahren eine ansteigende Vielfalt (Obaidat/Boudriga 2010). Eine Notwendigkeit, die dabei entsteht, ist die Auswahl eines bestimmten Systems für eine bestimmte Anforderung. Die richtige Auswahl ist dabei abhängig von der Art der Anforderung. Eine Systemarchitektur, die sich für eine bestimmte Klasse von Anforderungen eignet, muss nicht zwangsläufig für alle anderen Problemklassen geeignet sein (Hu/Gorton 1997).

Mit der Auswahl einer bestimmten Architektur stellen sich zugleich Fragen über die Leistungsfähigkeit des Systems: Wie wird sich die Leistungskurve des Systems bei zunehmender Last entwickeln? Welche Kriterien sollen für die Leistungsmessung herangezogen werden? Welche Techniken und technischen Hilfsmittel können für die Leistungsbeurteilung verwendet werden? All diese Fragen fallen in das Gebiet der Performance-Evaluation von Computersystemen.

Laut Ferrari (1986) datiert die erste wissenschaftliche Erwähnung von Performance-Evaluation von Computersystemen auf eine Arbeit von Alan Scherr (1965) zurück. Seitdem hat sich diese Fragestellung zu einer eigenen Disziplin entwickelt, wobei neben einer Vielzahl von wissenschaftlichen Arbeiten auch diverse Bücher erschienen sind, wie zum Beispiel Jain (1991) oder Sauer/Chandy (1981).

In der Literatur finden sich verschiedene Definitionen zu dem Begriff *Performance* (zum Beispiel Doherty (1970), Graham (1975)). Allgemein kann die Performance eines Systems mit der Fähigkeit, eine bestimmte Operation in einer festen Zeit in bestimmter Qualität und Häufigkeit durchzuführen, definiert werden (John/Eeckhout 2006). Nach Hu/Gorton (1997) sind folgende Aspekte in Bezug auf Performance von Computersystemen von Belang:

- Funktionalität
- Zuverlässigkeit
- Geschwindigkeit
- Ökonomische Effizienz

Funktionalität und Zuverlässigkeit werden in den meisten Fällen bereits von den Systementwicklern berücksichtigt. Demnach liegt das Hauptaugenmerk der Performance-Analysten hauptsächlich auf der Geschwindigkeit des Systems sowie der ökonomischen Effizienz. Die Geschwindigkeit eines Systems wird durch Leistungskenngrößen wie zum Beispiel Durchsatz oder Antwortzeit ausgedrückt. Ökonomische Effizienz bewertet die Kosten beim Design und der Implementierung eines Systems.

Für die Leistungsbewertung eines Systems müssen im ersten Schritt Leistungskenngrößen (Metriken) definiert werden. Unterschiedliche Metriken können gänzlich unterschiedliche Performance-Werte ergeben (Jain 1991). Daher ist die Auswahl passender Metriken ein entscheidender Faktor für eine erfolgreiche Leistungsbewertung. Ebenso wichtig ist die Auswahl einer entsprechenden Belastung des Systems (Workload). Nachdem passende Metriken und

der Workload definiert wurden, muss eine passende Methode der Performance-Evaluation gewählt werden. Diese werden grundsätzlich in Messung, analytische Verfahren und Simulation unterteilt. Eine genauere Betrachtung dieser Verfahren erfolgt in Kapitel 2.3.3. Abbildung 2-8 veranschaulicht die Gleichwertigkeit von System, Performance-Metriken und Workload.

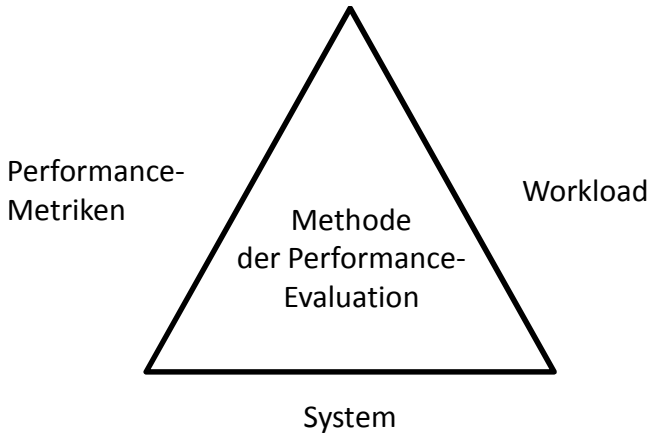


Abbildung 2-8: Aspekte der Leistungsbewertung
Quelle: eigene Darstellung in Anlehnung an Rechenberg/Pomberger/Pirklbauer (2006, 454)

2.3.1 Performance-Metriken

Für die Bewertung der Leistungsfähigkeit eines Systems müssen Performance-Metriken definiert werden. Die Auswahl passender Metriken ist dabei stark problemorientiert. So ist beispielsweise für Service-Anfragen eines Benutzers an ein System die Antwortzeit von großer Bedeutung, wogegen bei der Analyse von Computernetzwerken dem Durchsatz eine größere Bedeutung zugeschrieben wird. Es existieren eine Vielzahl von verschiedenen Metriken, aber keine offiziellen Standards (Risse 2006). Metriken von Standard-Benchmarks, beispielsweise von dem „Transaction Processing Council“[2], betreffen nur bestimmte Applikationsszenarios und können nicht generell eingesetzt werden (siehe auch Kapitel 2.3.3).

Nach Ray Jain (1991) sind folgende Eigenschaften für Performance-Metriken gefordert:

- Geringe Variabilität
- Redundanzfreiheit
- Vollständigkeit

Neben einer geringen Variabilität, die für eine höhere statistische Konfidenz sorgt, fordert die Redundanzfreiheit, dass nicht mehrere Metriken für abhängige Variablen definiert werden. Als Beispiel für Redundanzfreiheit kann das von Robertazzi (2000) dargestellte Routing-System angeführt werden, welches lediglich Pakete empfängt und sendet. Die durchschnittliche Anzahl $\bar{n}$ von Paketen im System kann mittels *Little's Law* (Little 1961) errechnet werden:

[2] Transaction Processing Council, http://www.tpc.org

$$\bar{n} = \lambda \cdot \bar{\tau}$$

λ ist hierbei die durchschnittliche Ankunftsrate, $\bar{\tau}$ stellt die durchschnittliche Wartezeit dar. Wie man unschwer erkennen kann, reicht es aus, entweder $\bar{n}$ oder $\bar{\tau}$ als Metrik zu wählen, da sich der jeweils zweite Wert aus den anderen Werten errechnen lässt.

Die dritte Eigenschaft, Vollständigkeit, bedeutet, dass mit den gewählten Metriken alle gewünschten Performance-Indikatoren abgedeckt werden sollen.

Leistungskenngrößen können zudem nach Zeit, Durchsatz und Auslastung kategorisiert werden (Rechenberg/Pomberger/Pirklbauer 2006, 454):

- *Kenngrößen zur Zeit*: Die Kenngröße Zeit kann in messbare Größen unterteilt werden. Die Bedienzeit (auch Service-Zeit genannt) ist die Bearbeitungszeit in einem System, also die Zeit, die zur Abarbeitung eines Programms benötigt wird. Die Wartezeit beschreibt die aufsummierte Zeit, die ein Programm auf seine Ausführung wartet. Die Verweil- oder Antwortzeit setzt sich aus Bedien- und Wartezeit zusammen. Sie beschreibt die gesamte Zeit vom Eintreffen bis zum Abgang eines Auftrags. In diesem Zusammenhang treten auch oft die Begriffe Latenzzeit und Verzögerungszeit auf. Bei der Bedien-, Verweil- und Wartezeit ist häufig nur der Erwartungswert wichtig, im Allgemeinen sind sie jedoch stochastische Größen (Melzer 2010, 180; Rechenberg/Pomberger/Pirklbauer 2006, 455).

- *Kenngrößen zum Durchsatz*: Die Kenngröße Durchsatz beschreibt die Anzahl der bearbeiteten Aufträge pro Zeiteinheit. Unter dem Begriff Grenzdurchsatz versteht man den maximal erreichbaren Durchsatz. Man spricht auch von der Bandbreite, wenn in einem Netzwerk der Durchsatz einen theoretischen Höchstwert erreicht. Dadurch wird die Antwortzeit beeinflusst, da eine übertragene Antwort nie schneller ist, als es die Bandbreite des Übertragungsmediums zulässt. Daher ist meist der maximal erreichbare Durchsatz an Aufträgen interessanter, der erreicht wird, ohne dass die Verweilzeit entweder die vorgegebene obere oder untere Schranke über- bzw. unterschreitet. Die Verlustrate beschreibt die Anzahl an abgewiesenen oder verlorenen Aufträgen pro Zeiteinheit (Melzer 2010, 180; Rechenberg/Pomberger/Pirklbauer 2006, 456).

- *Kenngrößen zur Auslastung*: Die Kenngröße Auslastung oder auch Effizienz eines Systems bezeichnet das Verhältnis von dem tatsächlich erreichten Durchsatz zum Grenzdurchsatz. Der maßgebliche Indikator, der angibt, wie performant ein System ist, ist der Engpass (auch Flaschenhals genannt). Unter einem Engpass versteht man die Komponente mit der höchsten Auslastung, somit ist er die schwächste Komponente in einem System. Um die Leistung des Gesamtsystems zu steigern, muss die Leistung der Engpasskomponente verbessert werden. Weitere Kenngrößen der Auslastung sind die Nutzlast, die Systemauslastung, der Verwaltungsaufwand sowie die Skalierbarkeit (Melzer 2010, 181; Rechenberg/Pomberger/Pirklbauer 2006, 456).

In dieser Arbeit wird die Leistung von Portalsystemen unter einer ansteigenden Anzahl von simultanen Benutzeranfragen evaluiert. Die dabei verwendete Performance-Metrik beschränkt sich auf die Antwortzeit als Messgröße, da diese

- die Anforderungen an eine Metrik für die Performance-Analyse erfüllt (Lilja 2000),

- mittels der Analysewerkzeuge des Portalsystems für die einzelnen Komponenten gemessen werden kann, sowie

- eine hohe Aussagekraft für die Fragestellung, wie sich eine steigende Anzahl an Benutzern auf das (für einen Benutzer gefühlte) Leistungsverhalten des Portalsystems auswirkt, bietet.

2.3.2 Workload

Systeme sind dahingehend konzipiert, in einer bestimmten Umgebung unter bestimmten Lastmustern zu arbeiten. Dementsprechend ist bei der Performance-Analyse darauf zu achten, eine ähnliche Umgebung mit ähnlichen Lastmustern zu schaffen. Nach Oed/Mertens (1981) ist der Workload definiert als die Gesamtheit aller Prozesse, Aufgaben, Transaktionen und Daten, die innerhalb eines bestimmten Zeitraums abgearbeitet werden.

Aufgrund der Komplexität und Inhomogenität realer Workloads muss ein Test-Workload entwickelt werden, der den realen Workload repräsentiert. Dieser sollte den statischen und dynamischen Eigenschaften des realen Workloads bestmöglich entsprechen. Zudem sollte er einfach reproduzierbar sein, damit er für verschiedene Systemanalysen verwendet werden kann (Jain 1991).

Die Lasterzeugung wird in der Regel mit einem eigenständigen Programm durchgeführt, welches über eine definierte Schnittstelle mit dem System kommuniziert. Durch die automatisierte Lasterzeugung wird eine Reproduzierbarkeit gewährleistet, die vergleichende Untersuchungen ermöglicht (Wilhelm 2001).

2.3.2.1 Workload-Charakterisierung

Bei der Workload-Charakterisierung wird ein Workload-Modell erstellt, welches ein vereinfachtes Abbild des realen Lastmusters darstellt. Dabei wird aus der Vielzahl an gesammelten Daten währen der Systemlaufzeit eine Teilmenge an relevanten Daten extrahiert.

Der Repräsentationsgrad eines Workload-Modells kann mittels einer Äquivalenzrelation ausgedrückt werden. Gegeben seien ein System *S*, sowie ein Set von Performance-Indizes *L*, welche von einem bestimmten Workload *W* gewonnen werden. Es kann nun eine Funktion f_S für *S* definiert werden, sodass $L = f_S(W)$. Sei nun W_r der reale Workload und L_r die entsprechenden Performance-Indizes, dann ergibt sich $L_r = f_S(W_r)$. Ebenso gilt $L_m = f_S(W_m)$, wobei W_m das Workload-Modell und L_m die entsprechenden Performance-Indizes darstellen. Das Workload-Modell W_m ist nun äquivalent zu dem realen Workload W_r, wenn L_m innerhalb bestimmter Grenzen von L_r liegt (vgl. Ferrari 1978).

Zwei Begriffe werden in der Literatur im Zusammenhang mit Workload-Charakterisierung häufig verwendet (vgl. Jain 1991):

1. Eine *Workload-Komponente* beschreibt Einheiten in dem zu analysierenden System. Dies können Applikationen, Programme, Kommandos, Maschineninstruktionen, aber auch Benutzer sein.

2. *Workload-Parameter* stellen die Messgrößen, Ressourcenanforderungen und Service-Anfragen dar, welche verwendet werden, um den Workload zu charakterisieren (beispielsweise Paketgrößen, Transaktionstypen, etc.).

Nach Ferrari/Serazzi/Zeigner (1983) erfolgt die Modellbildung in drei Phasen:

1. *Formulierungsphase*: Während der Formulierungsphase werden die zu betrachtenden Parameter definiert. Dies ist abhängig von dem Ziel der Performance-Analyse sowie den zur Verfügung stehenden Daten. Neben den klassischen Parametern auf dem physikalischen Level (CPU-Zeit, Hauptspeicher- und IO-Operationen), können Parameter auf funktionalem Level in Betracht gezogen werden, wie beispielsweise Ausführungshäufigkeit bestimmter Programme oder die Anzahl der Aufrufe eines Web-Services.

2. *Konstruktionsphase*: Bei der Konstruktionsphase werden die gewählten Parameter mit den entsprechenden Werten der gesammelten Daten in Verbindung gebracht und die Werte den einzelnen Komponenten des Workload-Modells zugewiesen.

3. *Validierungsphase*: Während der Validierungsphase werden Evaluationskriterien definiert und die Validität des Workload-Modells überprüft.

Die Schwierigkeit liegt vor allem bei der Auswahl der Parameter. Der zu analysierende Workload kann als *n*-dimensionaler Raum (wobei *n* die Anzahl an Workload-Parametern darstellt) verstanden werden. Jedes Set von Parameter-Werten repräsentiert einen Punkt in diesem Raum. Die Herausforderung liegt nun darin, die Anzahl an Punkten in dem Raum zu reduzieren und Parameter mit ähnlichen Charakteristiken zu gruppieren.

2.3.3 Methoden der Performance-Evaluation

In der Literatur, wie zum Beispiel bei Jain (1991), werden zumeist drei Methoden der Performance-Evaluation unterschieden:

1. Messung

2. analytische Verfahren

3. Simulation

Ferrari/Serazzi/Zeigner (1983) fassen analytische Verfahren und Simulation zu einer Gruppe zusammen, da beide ein Performance-Modell als Grundlage benötigen. Unabhängig von der Klassifizierung, hängt die richtige Wahl der Methode vom Entwicklungsstand des Systems ab (Risse 2006). Nach Jain (1991) und Trivedi et al. (1994) werden analytische Verfahren hauptsächlich in sehr frühen Entwicklungsstadien verwendet, da sie einen geringeren Aufwand erfordern, allerdings auch eine geringere Genauigkeit bieten. Eine höhere Genauigkeit, allerdings verbunden mit einem größeren Aufwand, kann mittels Simulation erreicht werden. Messungen wiederum bieten die höchste Genauigkeit, können allerdings erst dann durchgeführt werden, sobald das System zur Verfügung steht. Ihr Ziel ist es, leistungsschwache Programmteile aufzudecken und die Leistung der gesamten Rechenanlage oder auch nur von bestimmten Komponenten zu erfassen (Rechenberg/Pomberger/Pirklbauer 2006, 460; Versick 2010, 29).

2.3.3.1 Messung

Messungen werden aus drei verschiedenen Gründen durchgeführt. Zum einen sollen sie Daten zur Leistungsfähigkeit eines Systems sammeln, um Informationen zur Steigerung der Performance zu erhalten. Der zweite Grund für die Durchführung von Messungen ist die Datensammlung für die Workload-Modellierung. Der dritte Anwendungsfall entsteht bei der Validierung von Performance-Modellen.

Damit die Aktivitäten des Systems beobachtet werden können, wird ein sogenanntes Monitoring-Tool verwendet. In der Informatik ist ein Monitoring-Tool ein Werkzeug, welches zur Beobachtung der Aktivitäten auf einem System genutzt wird. Im Allgemeinen überwachen Monitore die Performance von Systemen, sammeln Performance-Statistiken, analysieren die Daten und zeigen die Ergebnisse an (Jain 1991, 93). Snodgrass (1988) definiert Monitoring als eine Extraktion von dynamischen Informationen eines Rechenvorgangs, während dieser ausgeführt wird. Ziel ist es, die Ursachen für die gemessene Leistung darzulegen. Die dadurch gewonnenen Informationen dienen vor allem dazu, Programm- oder Datenstrukturen zu verbessern (Alpar et al. 2008, 55).

Die unterschiedlichen Messgrößen, die mit einem Monitor erfasst und ausgewertet werden können, müssen definiert sein. Mit den erhobenen Daten kann man anschließend Aussagen über den Betriebszustand des überwachten Systems tätigen, das Leistungsverhalten der Funktionseinheiten analysieren und Leistungsengpässe aufdecken. Die verrichtete Überwachungstätigkeit wird, wie bereits in Kapitel 1.2 erwähnt, Monitoring genannt (Winkler 2010, 27; Rechenberg/Pomberger/Pirklbauer 2006, 460).

Für die Leistungsmessung können unterschiedliche Monitoring-Methoden verwendet werden. Ausgehend von der Implementierungsebene unterteilt man Monitore in Hardware-, Software- und Hybrid-Monitore:

- *Hardware-Monitore*: Ein Hardwaremonitor ist ein eigenständiges, komplexes, elektronisches Messinstrument, welches mit Messfühlern die Leistung der Hardwarekomponenten erfassen kann. Durch die Messung elektrischer Signale in der Hardware eines Systems können dessen Zustände ermittelt werden, wie zum Beispiel dessen CPU. Die Datenerfassung untergliedert sich in Aufnahme, Verknüpfung und Verarbeitung der mit Hilfe der Messfühler erfassten Signale (Alpar et al. 2008, 55; Haas/Zorn 1995, 209). Ein großer Vorteil von Hardwaremonitoren ist, dass sie sehr kurze Vorgänge korrekt erfassen können und keinen Messaufwand im System erzeugen, was für Echtzeitsysteme wesentlich ist. Nachteile liegen vor allem bei der Herstellung des Bezuges zwischen den Messdaten in der Hardware und den verursachten Programmen im System (Heinrich/Lehner 2005, 535). Daher ist der Anwendungsbereich von Hardwaremonitoren meist auf hardwarenahe Analysen beschränkt und kaum für die Parametrisierung von Modellen geeignet. Des Weiteren sind die Anschaffungskosten und das benötigte Knowhow meist sehr hoch (Jain 1991, 100).

- *Software-Monitore*: Nach Teuffel/Vaupel (2010, 231) ist ein Software-Monitor „[…] ein Programm, das neben anderen Programmen auf dem Rechner läuft und Daten sammelt, die entweder vom Betriebssystem oder von einzelnen Anwendungen zur Verfügung gestellt werden“. Dabei wird das Verhalten des zu untersuchenden Anwen-

dungssystems durch die in regelmäßigen Abständen ausgeführten Programmkomponenten, dem Softwaremonitor, protokolliert (Winkler 2010, 27). Software-Monitore können auf den unterschiedlichsten Ebenen Daten sammeln. Zu den typischen Daten, die gemessen und analysiert werden, zählen CPU-Zeiten, Speicherbelegung, I/O-Zugriffe und Cache-Aktivitäten. Der Vorteil von Software-Monitoren liegt vor allem darin, dass Rechner- und Programmzustände detailliert erfasst und dem verursachenden Programm zugeordnet werden können. Im Gegensatz zu Hardwaremonitoren haben Softwaremonitore eine höhere Eingabebreite, eine höhere Aufnahmekapazität, wenn nötig kann man sie leichter modifizieren, sie sind einfacher zu entwickeln und die Anschaffungskosten sind geringer. Jedoch haben sie generell eine niedrigere Eingabegeschwindigkeit, eine niedrigere Auflösung und einen höheren Overhead, da die Softwaremonitore mit anderen Prozessen des Objektsystems konkurrieren (Jain 1991, 95). Dieser Ressourcenverbrauch führt oft zu einer Verfälschung der Laufzeit und somit des dynamischen Verhaltens des zu untersuchenden Objektes. Software-Monitore werden somit meist dann bevorzugt, wenn sichergestellt ist, dass das Zeitgefüge des zu messenden Systems nur unbedeutend gestört wird (Zitterbart 1990).

- *Hybrid-Monitore*: Hybrid-Monitore sind eine Kombination aus einem Hardware- und einem Software-Monitor. Ziel ist es, die positiven Charaktereigenschaften beider Monitortypen in einem Typ zu vereinen. Somit besitzt ein Hybridmonitor sowohl eine Software-, also auch eine unterstützende Hardware-Komponente. Die gute Datenaufbereitungsmöglichkeit des Softwaremonitors wird mit der guten Auflösung und dem geringen Overhead des Hardwaremonitors kombiniert. Die Softwarekomponente hilft, die gewünschten Messdaten im Objektsystem zu ermitteln. Die Hardware dient mit einer geeigneten Zeitbasis und einem Speicher zur Speicherung der Messdaten, diese chronologisch zu erfassen und anschließend zu speichern (Zieher 1989, 24).

Eine weitere mögliche Klassifizierung ist nach dem Auslösemechanismus in zeitbasierte und ereignisbasierte Monitore möglich (Jain 1991, 94).

- *Zeitbasiertes Monitoring*: Das zeitbasierte bzw. passive Monitoring muss die gewünschten Informationen selbst abfragen, da das Objektsystem von sich aus keine Daten zur Verfügung stellt. Über eine periodische Abfrage kann die gesamte Statusinformation des zu messenden Systems ermittelt werden. Da man keine Informationen über Zustandsänderungen hat, wird ein zeitbasierter Monitor in festen Zeitabständen aktiviert. Diese Aktivierung nennt man Ereignis. Ein Ereignis ist eine atomare momentane Aktion. Passive Monitore sind ideal für die Beobachtung häufiger Ereignisse und für die Überprüfung von lang laufenden Prozessen in verhältnismäßig großen Intervallen. Zudem muss die Anwendung nicht angepasst oder verändert werden. Allerdings ist aufgrund der hohen Datenübertragung der Zeitaufwand für die periodischen Abfragen sehr aufwendig (Nissen/Petsch/Schorcht 2008, 208).

- *Ereignisbasiertes Monitoring*: Ereignisbasiertes Monitoring stellt das dynamische Verhalten eines Programms durch eine Abfolge von Ereignissen dar. Im Gegensatz zum zeitbasierten Monitoring ist ereignisbasiertes Monitoring für effiziente Programmanalysen geeignet, da es darauf abzielt, Einblick in das dynamische Verhalten eines parallelen Programms zu bekommen (Hofmann et al. 1994). Damit die Informationen bereitgestellt werden, werden entsprechende Anweisungen im Anwendungs-

code eingefügt. Durch diese Änderung im Code kann sich zum einen das Verhalten des Objektsystems ändern und zum anderen die Anwendungsperformance beeinträchtigt werden (Nissen/Petsch/Schorcht 2008, 208).

2.3.3.2 Analytische Verfahren

Wie bereits zu Beginn des Kapitels 2.3.3 erwähnt, benötigen sowohl analytische Verfahren als auch Simulationsansätze ein zugrundeliegendes Modell (Ferrari/Serazzi/Zeigner 1983). Damit ein einheitliches Verständnis für die verwendeten Begriffe System, Modell und analytisches Verfahren bzw. Simulation gegeben ist, werden diese Begriffe in Abbildung 2-9 in den entsprechenden Kontext gesetzt.

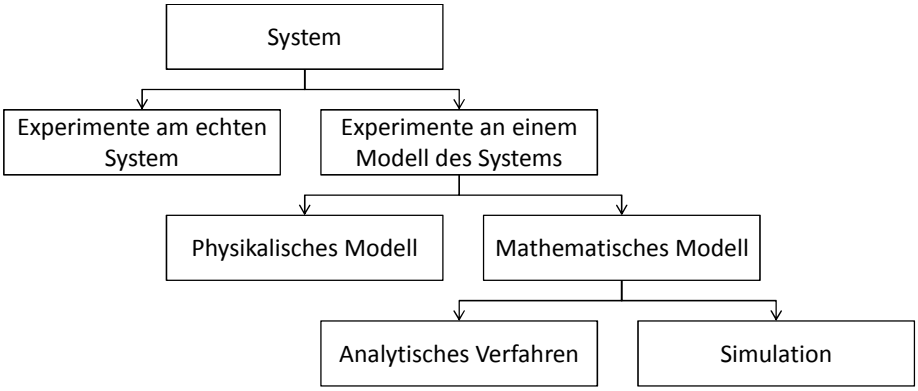


Abbildung 2-9: Terminologie der Systemmodellierung
Quelle: eigene Darstellung

Ein System ist in sich strukturiert als Kollektion interdependenter Komponenten, wobei einzelne Komponenten (auch Teilsysteme genannt) wiederum aus Komponenten bestehen können. Die Durchführung der Experimente, beispielsweise eine Performance-Analyse, kann nun am System selbst oder mit einem Modell des Systems erfolgen. Ein Modell ist ein Ersatzsystem, welches unter Vereinfachung und Idealisierung mit der wesentlichen Zielsetzung der Reduktion der Komplexität des zu untersuchenden Systems gebildet wird. Es kann entweder physikalisch erstellt oder mathematisch ausgedrückt werden. Die mathematischen Ausdrücke wiederum beinhalten Formeln oder auch eine graphische Repräsentation, die mathematisch beschrieben werden kann. Zum Lösen des Modells können diverse Simulationsansätze oder analytische Verfahren herangezogen werden.

Analytische Modelle basieren auf einer Reihe von Formeln und numerischen Algorithmen, die zur Generierung von Performance-Metriken von Modellparametern verwendet werden. Performance-Modellierungs-Formalismen können grundsätzlich in zwei Klassen eingeteilt werden: deterministisch und probabilistisch. Bei deterministischen Modellen sind die Messgrößen festgelegt, wobei bei probabilistischen Modellen ein gewisses Maß an Ungewissheit möglich ist (vgl. Jonkers 1994). Zu den bekanntesten probabilistischen, analytischen Modellen zählen Markov-Ketten und Petri-Netze, die in diesem Kapitel vorgestellt werden, sowie Warteschlangennetze, die in Kapitel 2.4 näher beschreiben werden.

Zur Darstellung der gesammelten Leistungsdaten werden bei der analytischen Modellierung einfache mathematische Ausdrücke verwendet, die in kurzer Zeit rechnerisch gelöst werden können. Die Vereinfachung der Charakteristiken führt allerdings oft auch dazu, dass das Mo-

dell das Systemverhalten nicht mehr ausreichend korrekt darstellt und somit zu inakkuraten Ergebnissen führt. Dennoch führen sehr viele Modelle in der Praxis zu annehmbaren Ergebnissen, die typischerweise eine Fehlerquote von zehn bis dreißig Prozent aufweisen. Diese Fehlerquote ist für viele Anwendungen ausreichend (Lazowska et al. 1984).

Markov-Ketten

Ein stochastischer Prozess ist definiert als eine Familie von Zufallsvariablen $\{X_t : t \in T\}$, wobei jede Zufallsvariable X_t einen Index $t \in T$ aufweist, der als Parameter der Zeit verstanden wird, wenn gilt $T \subseteq \mathbb{R}_+ = [0, \infty]$. Die Menge aller möglichen Werte von X_t (für jedes $t \in T$) wird als der Zustandsraum S bezeichnet (Bolch et al. 2006, 52). Für jeden beliebigen Zeitpunkt $t \in T$ liefert der stochastische Prozess also eine diskrete oder kontinuierliche Zufallsvariable. Dadurch entstehen von Zeitpunkt zu Zeitpunkt zufällige, stochastische Übergänge.

Markov-Prozesse sind eine besondere, vielleicht die wichtigste Unterklasse von stochastischen Prozessen. Sie bieten ein sehr flexibles, leistungsfähiges und effizientes Mittel zur Beschreibung und Analyse von dynamischen Systemeigenschaften. Zudem stellen Markov-Prozesse die fundamental Theorie des zugrundeliegenden Konzepts der Warteschlangen-Systeme dar (Bolch et al. 2006, 51f.). Sie erfüllen die Forderung, dass Zustände „[…] nur von dem jeweiligen vorherigen Zustand stochastisch abhängig und von allen weiter zurückliegenden Zuständen stochastisch unabhängig sind“ (Lunze 2006, 347). Mit anderen Worten wird ein Prozess Markov-Prozess genannt, wenn die zukünftigen Zustände eines Prozesses nicht von der Vergangenheit, sondern nur von der Gegenwart abhängen. Diese Markov-Eigenschaft macht es leichter, einen Prozess zu analysieren, da man sich nicht an den kompletten, vergangenen Bewegungsablauf erinnern muss (Jain 1991, 516).

Auch der Zustandsraum eines stochastischen Prozesses kann kontinuierlich oder diskret sein. Bei Markov-Prozessen ist der Zustandsraum im Allgemeinen stetig. Bei diskretem Zustandsraum wird von Markov-Ketten gesprochen. Die folgende Vier-Felder-Tafel zeigt eine Klassifizierung auf zwei Ebenen auf.

	Diskreter Parameterraum	**Kontinuierlicher Parameterraum**
Diskreter Zustandsraum	Diskrete Markov-Kette	Kontinuierliche Markov-Kette
Kontinuierlicher Zustandsraum	Diskreter Markov-Prozess	Kontinuierlicher Markov-Prozess

Tabelle 2-2: 4-Felder-Tafel
Quelle: eigene Darstellung in Anlehnung an Ferschl (1970)

Eine zeitdiskrete Markov-Kette kann aufgrund ihrer Zustandsmenge Z entweder endlich oder unendlich sein (Lunze 2006, 338). Zeitkontinuierliche Markov-Ketten sind eine wichtige Klasse von stochastischen Prozessen, die weithin in der Praxis verwendet werden, um Systemperformance- und Zuverlässigkeitsmerkmale zu bestimmen. Ihre Analyse betrifft am häufigsten die Berechnung von Dauerzustands- und vorübergehenden Zustandswahrscheinlichkeiten (Baier et al. 2003). Beide Arten von Markov Ketten bieten verschiedene, dennoch miteinander verwandte Modellierungstechniken an, jeder von ihnen mit einer anderen Anwendungsdomäne.

Die wichtigsten strukturellen Eigenschaften von Markov-Ketten sind wie folgt:

- *Homogen*: Eine homogene Markov-Kette besitzt zu jedem Zeitpunkt dieselben Wahrscheinlichkeiten, somit sind die Übergangswahrscheinlichkeiten zeitunabhängig (Domschke/Drexl 2007, 218).

- *Gedächtnislos*: Die Gedächtnislosigkeit nimmt Einfluss auf die Markov-Eigenschaft, da für künftige Systembewertungen lediglich der aktuelle Zustand wichtig ist und keine vorhergehenden Zustände im Gedächtnis bleiben (Lunze 2006, 347).

- *Absorbierend*: Wenn eine Markov Kette mindestens einen absorbierenden Zustand besitzt, heißt sie absorbierend. Ein Zustand $i \in S$ wird genau dann ein absorbierender Zustand genannt, wenn kein anderer Zustand der zeitdiskreten Markov-Kette von ihm erreicht werden kann (Bolch et al. 2006, 62).

- *Irreduzibel*: Eine Markov-Kette wird irreduzibel genannt, wenn alle Zustände in der Kette paarweise untereinander erreichbar sind. Erreichbarkeit ist gegeben, wenn es möglich ist, von Zustand i zu Zustand j in einer endlichen Anzahl von Schritten zu gelangen (Bolch et al. 2006, 62).

- *Reduzibel*: Bei einer reduzierbaren Markov-Kette kann die Zustandsmenge Z in disjunkte Teilmengen Z_i von untereinander stark zusammenhängenden Zuständen zerlegt werden. Zwei Zustände i und j sind stark zusammenhängend, wenn es einen Pfad sowohl von i nach j als auch umgekehrt gibt (Lunze 2006, 357).

Für die Analyse von Markov-Prozessen existieren verschiedene Methoden, wie zum Beispiel die Berechnung der Wahrscheinlichkeit, einen bestimmten Zustand zu erreichen. Allerdings kann die Anzahl an Zuständen bei komplexer werdenden Systemen sehr stark ansteigen („State Space Explosion“), sodass eine automatische Generierung des Zustandsraumes und effiziente Lösungswege der zugrundeliegenden Gleichungen notwendig werden (Bolch et al. 2006). Daraus haben sich spezielle Notationen (basierend auf Petri Netzen und Warteschlangenmodellen, welche in den folgenden Kapiteln vorgestellt werden) sowie Anwendungen zur automatischen Generierung des Zustandsraumes entwickelt.

Petri-Netze

Anfang der sechziger Jahre beschrieb Carl Adam Petri in seiner Dissertation „Kommunikation mit Automaten“ (Petri 1962) eine Theorie, die in den darauffolgenden Jahren unter der Bezeichnung Petri-Netze weltweit bekannt wurde (Peterson 1977). Petri-Netze können als grafischer Formalismus zur Beschreibung von Prozessstrukturen verstanden werden. Ein großer Vorteil der Systemmodellierung mit Petri-Netzen ist, dass durch die grafische Notation ein hoher Grad an Anschaulichkeit erreicht wird.

Petri-Netze werden in vielen Anwendungsgebieten zur Systemmodellierung verwendet, wie zum Beispiel bei der Modellierung von Geschäftsprozessen oder Workflow-Systemen. Bei der Modellierung von Rechensystemen eignen sich Petri-Netze vor allem für die Untersuchung funktionaler Aspekte („globale“ Funktionalität, Korrektheit). Für die quantitative Analyse, wie sie bei der Leistungsbewertung eines Systems benötigt wird, eignen sich klassische Petri-Netze nur bedingt. Diesen Mangel versuchte man durch zusätzliche Funktionen (zeitbe-

haftete Petri-Netze, stochastische Petri-Netze) aufzuheben, die im weiteren Verlauf dieses Kapitel vorgestellt werden.

Kurz gesagt ist ein Petri-Netz ein Graph, der mit graphentheoretischen Verfahren analysiert werden kann. In der Literatur findet sich eine Vielzahl von Petri-Netz-Typen. Sie unterscheiden sich beispielsweise dadurch, dass sie eine andere Beschriftung nutzen, unterschiedliche Marken verwenden, etc. Dennoch liegt bei der Definition immer ein Petri-Netz-Graph zugrunde. Dieser kann je nach Petri-Netz-Typ um weitere Funktionen und Abbildungen erweitert werden.

Die einfachste Form eines Petri-Netzes wird durch Stellen, Transitionen und Kanten beschrieben und mittels eines 5-Tupels dargestellt (Bause/Beilner 1989):

$$PN = (S, T, W^-, W^+, M_0)$$

Folgende Aussagen gelten:

- S ist eine endliche, nichtleere Menge von Stellen
- T ist eine endliche, nichtleere Menge von Transitionen.
- $S \cap T = \emptyset$
- W^-, W^+ sind Funktionen aus $[S \times T \rightarrow \mathbb{N}_0]$. W^- heißt Rückwärtsinzidenzfunktion, W^+ Vorwärtsinzidenzfunktion.
- M_0 ist eine Funktion aus $[S \rightarrow \mathbb{N}_0]$ und stellt die Anfangsmarkierung dar.

Petri-Netz-Modelle bestehen aus zwei wesentlichen Teilen: Netzstruktur und Markierung. Die Netzstruktur ist ein eingetragener, bipartiter, gerichteter Graph, der den statischen Teil des Systems darstellt. Dazu gehören Stellen, Transitionen und Kanten, die die statischen Verhältnisse eines Systems modellieren. Ein Knoten in einem Petri-Netz-Modell besteht aus Stellen und Transitionen.

Stellen sind passive Elemente und stellen Zustandszustände dar. Graphisch wird eine Stelle durch einen Kreis dargestellt. Transitionen sind aktive Elemente, die lokale Zustandsübergänge modellieren können. Eine Transition wird graphisch durch ein Rechteck, manchmal auch durch ein Quadrat, dargestellt.

Kanten verbinden Stellen und Transitionen. Sie stellen somit eine Beziehung zwischen den beiden Elementen her. Dabei können Stellen nicht mit anderen Stellen und Transitionen nicht mit anderen Transitionen verbunden werden. Die Kanten werden graphisch mit einem Pfeil, der die Richtung der Relation angibt, dargestellt.

Markierungen sind das vierte Element von Petri-Netzen. Diese modellieren die dynamischen Verhältnisse eines Systems, indem sie angeben, in welchem Zustand sich das Gesamtsystem im Augenblick befindet. Sie befinden sich auf einer Stelle und werden als schwarze Kreise in den Stellen graphisch dargestellt. Abbildung 2-10 zeigt die vier genannten Elemente und deren entsprechende graphische Notation.

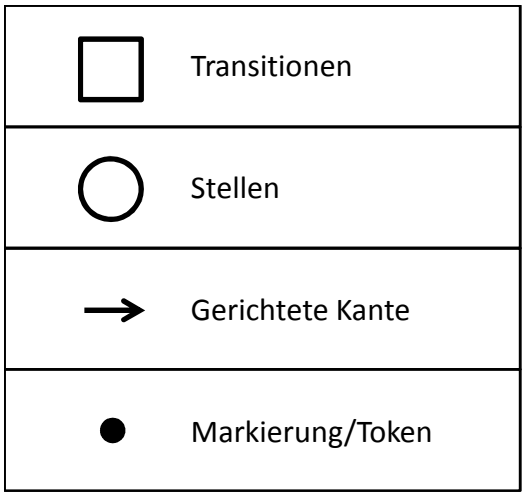


Abbildung 2-10: Elemente von Petri-Netzen
Quelle: eigene Darstellung in Anlehnung an Müller (2005, 87)

Eine Möglichkeit, die vielen verschiedenen Petri-Netz-Typen zu klassifizieren, ist die Unterteilung in Low-Level- und High-Level-Petri-Netze. Low-Level nennt man alle Petri-Netz-Typen, in denen es nur eine Markierungsart gibt. Bei High-Level-Petri-Netzen werden individuelle Markierungen verwendet. Mit diesen individuellen Markierungen, auch Prädikate genannt, ist es möglich, komplexe Zustände abzubilden. Die Schaltregeln werden erweitert und den Transitionen werden Schaltbedingungen mit Variablen zugeordnet.

Zur Leistungsbewertung (quantitativen Analyse) eines Systems eignen sich Petri-Netze nicht, da sie den Begriff der Zeit nicht beinhalten. Daher entstanden Anfang der siebziger Jahre sogenannte zeitbehaftete Petri-Netze (Time(d)-Petri-Nets, TPN) (Merlin/Farber 1976). Durch die Möglichkeit, Zeitpunkte und Intervalle abbilden zu können, kann die Reihenfolge des Markierungsverbrauches und des Schaltens gesteuert werden. Ein weiterer gängiger Vertreter von High-Level-Petri-Netzen sind beispielsweise gefärbte Petri-Netze (Coloured-Petri-Nets, CPN) (Buchholz 1992).

Stochastische Petri-Netze sind ein spezieller Zweig der zeitbehafteten Petri-Netze, wobei jeder Transition eine exponentiell verteilte Schaltrate zugewiesen wird. Für die analytische Modellierung ist die Exponentialverteilung die wichtigste und auch die am leichtesten handhabbare Verteilung, da sie als einzige kontinuierliche Verteilung die Markov-Eigenschaft der Gedächtnislosigkeit besitzt. Entsprechend lassen sich somit viele Kennwerte wie bei der Warteschlangentheorie (siehe Kapitel 2.4) berechnen. (Fink/Schneidereit/Voß 2005, 129; Vogel-Heuser 2003, 91). Ein stochastisches Petri-Netz besitzt neben den bereits eingeführten Elementen $PN = (S, T, W^-, W^+, M_0)$ eine Menge der Schaltraten $R = \{r_1, \dots, r_m\}$, wobei r_i den Mittelwert der zur Transition t_i gehörenden, exponentiell verteilten Schaltrate darstellt.

Obwohl stochastische Petri-Netze viele Vorteile für die analytische Modellierung besitzen, sind auch einige Nachteile bekannt. Beispielsweise zeigt Bause (1993) Probleme bei der Beschreibung von Scheduling-Strategien mit Petri-Netz-Elementen auf und stellt dabei einen Formalismus, der Petri-Netze mit der Warteschlangenmodellierung kombiniert (Queueing-Petri-Nets, QPN), vor. Allerdings finden sich in der Literatur nur wenige praktische Beispiele für den Einsatz von QPN. Kounev/Buchmann (2003) nutzen sie zum Beispiel für die Performance-Evaluation von verteilten E-Business-Anwendungen. Der Grund für die wenigen An-

wendungsbeispiele könnte darin liegen, dass die Analyse der QPN immer noch auf Markov-Ketten mit all ihren Vor- und Nachteilen beruht.

2.3.3.3 Simulation

Arnold/Isermann/Kuhn (2004) beschreiben Simulation als „[…] das Nachbilden eines Systems mit seinen dynamischen Prozessen in einem experimentierbaren Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind. Insbesondere werden die Prozesse über die Zeit entwickelt. Im weiteren Sinne wird unter Simulation das Vorbereiten, Durchführen und Auswerten gezielter Experimente mit einem Simulationsmodell verstanden."

In der Literatur variieren die Meinungen über die Klassifizierung der Simulationsansätze (Domschke/Drexl 2007). Abbildung 2-11 stellt die in dieser Arbeit vertretene Klassifizierung dar.

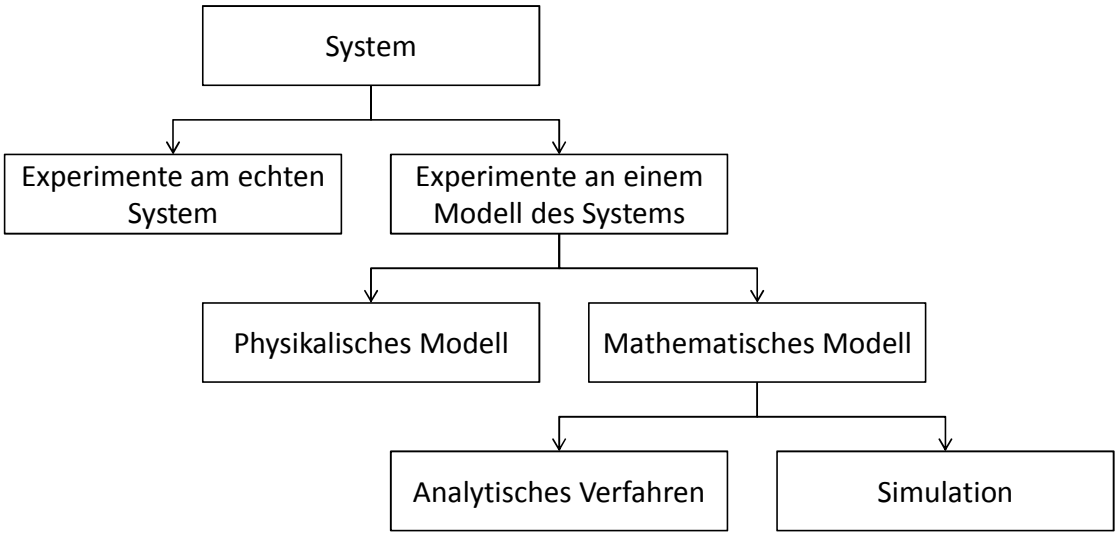


Abbildung 2-11: Klassifizierung der Simulationsansätze
Quelle: eigene Darstellung

Bei einem deterministischen Simulationsmodell sind keine Zufallsvariablen, sondern nur deterministische Komponenten enthalten, das heißt, es werden alle mathematischen und logischen Beziehungen zwischen den Elementen im Voraus festgelegt (Rubinstein/Kroese 2008, 84).

Ausschlaggebend für Fragestellungen in der stochastischen Simulation sind zufallsabhängige Größen, deren Einfluss man nicht vollständig erfassen kann. Es müssen Rechenalgorithmen gefunden werden, die eine hinreichend genaue Nachbildung des zufallsbasierten Geschehens auf einem Rechner schaffen können. Da ein stochastisches Modell mindestens eine zufällige Eingabevariable hat, führt dies zu zufälligen Ausgabevariablen. Den kompletten Vorgang einer stochastischen Simulation kann man grob in folgende Teilschritte untergliedern: Zunächst wird ein Modell des realen Systems durch mathematische Mittel erstellt. Anschließend stellt man die zufallsabhängigen/stochastischen Input-Daten bereit. Nachdem die Simulationsexperimente durchgeführt worden sind, werden die Ergebnisse erfasst, ausgewertet, dargestellt und interpretiert. Wenn nötig, wird im letzten Schritt das Modell angepasst. Die meisten

Warteschlangensysteme und Markov-Prozesse werden stochastisch modelliert (Kolonko 2008, 1f.; Rubinstein/Kroese 2008, 84).

In einer statischen Simulation spielt der Faktor Zeit keine Rolle, da sie nur einen Zeitpunkt betrachten und sozusagen eine Momentaufnahme darstellen (Rubinstein/Kroese 2008, 84). Bei dem Einsatz der Monte-Carlo-Simulation können die vorliegenden Objekte zum einen stochastischer Natur sein, zum anderen kann man mit ihr auch deterministische Probleme lösen. Sie stützt sich dabei auf wiederholten Stichproben und die statistische Analyse, um die Ergebnisse zu berechnen. Die Methode der Simulation bezieht sich stark auf Zufallsexperimente, somit ist ein spezifisches Ergebnis nicht im Voraus bekannt. In diesem Kontext kann die Monte-Carlo-Simulation als eine sogenannte Was-wäre-wenn-Analyse angesehen werden (Raychaudhuri 2008).

Im Gegensatz zu den statischen Simulationsmodellen stellen dynamische Simulationsmodelle Systeme dar, die sich im Laufe der Zeit weiterentwickeln. Mit Hilfe von Zustandsvariablen beschreibt man den Zustand zu einem Simulationszeitpunkt. Die zeitlichen Änderungen können kontinuierlich oder diskret stattfinden.

Bei einer kontinuierlichen Simulation ändert sich der Wert der Zustandsvariablen kontinuierlich über die Simulationszeit. Solche Modelle bilden mittels einer oder mehrerer Differentialgleichungen den Zusammenhang zwischen Zeitfortschritt und Änderungen der Zustandsvariablen ab. Dabei entziehen sich die Differentialgleichungen aufgrund ihrer Komplexität einer analytischen Behandlung. Durch den Einsatz von numerischen Lösungsverfahren können kontinuierliche Modelle angenähert gelöst werden. Daher wird der Wert einer kontinuierlichen Variablen meist iterativ ermittelt. Somit ist in kontinuierlichen Modellen ebenfalls nur eine endliche Anzahl von unterschiedlichen Variablenwerten abgreifbar (Lantzsch/Schneider/Schwarz 2000; Domschke/Drexl 2007, 227).

Bei einer diskreten Simulation verändert sich der Wert der Zustandsvariablen nur an endlich vielen Zeitpunkten. Bei einer ereignisdiskreten Simulation, so wie sie auch von dem in dieser Arbeit verwendeten Simulator *LQsim* (siehe Kapitel 2.5.4) durchgeführt wird, ändert sich der Zustand eines Modells nur an einer diskreten Menge an simulierten Zeitpunkten, auch Ereigniszeit genannt. Im Zeitraum zwischen zwei Ereignissen bleibt der Wert einer Variablen konstant. Somit besteht eine ereignisdiskrete Simulation aus folgenden Phasen:

1. Die Simulationsuhr wird initialisiert und die Zeiten zukünftiger Ereignisse werden festgelegt.

2. Die Simulationsuhr wird auf den Zeitpunkt des nächsten Ereignisses gestellt.

3. Der Systemstatus wird aktualisiert.

4. Die Zeiten zukünftiger Ereignisse werden aktualisiert und Schritt 1 wird wiederholt.

Bei zeitdiskreten Simulationen schreitet die Simulationsuhr hingegen in festen Zeiteinheiten Δt fort. Nach jedem Schritt werden die Variablen für das Intervall $[t, t + \Delta t]$ aktualisiert. Der Unterschied zwischen ereignisdiskreten und zeitdiskreten Simulationen ist in Abbildung 2-12 visualisiert.

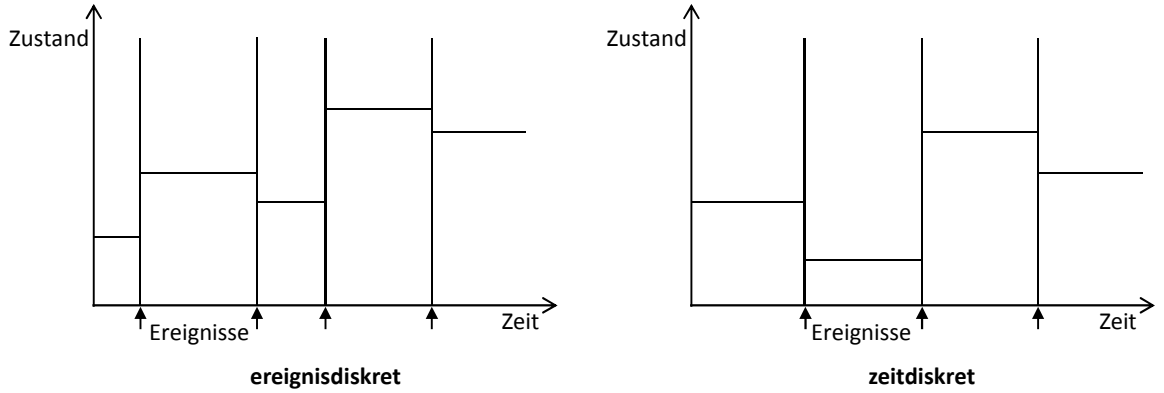


Abbildung 2-12: Ereignisdiskrete vs. zeitdiskrete Simulation
Quelle: eigene Darstellung in Anlehnung an Banks et al. (2004, 11)

Werden kontinuierliches und diskretes Simulationsverhalten kombiniert, spricht man von einer hybriden Simulation (Kourjanski/Varaiya 1995).

2.4 Warteschlangennetze

Im täglichen Leben begegnen uns sehr viele Situationen, bei denen wir warten; ob bei einem Zahnarzttermin, vor dem Kaffeeautomaten oder der Supermarktkasse. Bei Rechnersystemen sind ebenfalls viele Warteschlangensituationen vorzufinden, beispielsweise bei der Prozessverwaltung auf der Betriebssystemebene, bei der Nutzung geteilter Ressourcen (zum Beispiel Drucker) oder bei Kommunikationsprotokollen von Rechnernetzen. Die Warteschlangentheorie ist daher ein wertvolles Werkzeug für die Analyse von Rechnersystemen.

Als Vater der Warteschlangentheorie kann der Däne Agner K. Erlang angesehen werden, der sich als erster an die mathematische Behandlung von Telefongesprächen wagte. Den Startschuss setzte er mit seiner 1909 veröffentlichten Arbeit „The Theory of Probabilities and Telephone Conversations“ (Erlang 1909). In den dreißiger Jahren wurde der nächste Meilenstein im Zusammenhang mit der Warteschlangentheorie gelegt, die Pollaczek-Khintchine-Formel (Pollaczek 1930; Khintchine 1932). Diese von Felix Pollaczek entwickelte Formel ermöglichte erstmals eine stark vereinfachte Berechnung der Kunden in einem Wartesystem mit allgemeinen Annahmen. Im Jahr 1951 veröffentlichte David G. Kendall seine Arbeit über das Konzept der eingebetteten Markov-Ketten (Kendall 1951). Von ihm stammt auch die noch heute gültige Notation für Warteschlangensysteme. Fast zur gleichen Zeit entwickelte Dennis V. Lindley eine Gleichung zur Berechnung der mittleren Wartezeit in einem System mit sehr allgemeinen Annahmen (Lindley 1952). Bis zum Jahr 1957 lag das Hauptaugenmerk auf einzelnen Bedienstationen. James R. Jackson beschäftigte sich als erster eingehend mit Warteschlangennetzen und fand heraus, dass man die Lösung eines offenen Warteschlangennetzes unter bestimmten Bedingungen aus den Lösungen der einzelnen Warteschlangenknoten zusammensetzen kann (Jackson 1964). Gordon/Newell (1967) zeigten, dass auch geschlossene Netzwerke unter denselben Annahmen eine solche Produktformlösung besitzen. Mit dem Auftreten von Computern und Computernetzwerken erkannte man die Warteschlangentheorie als leistungsstarkes Design- und Analyseinstrument.

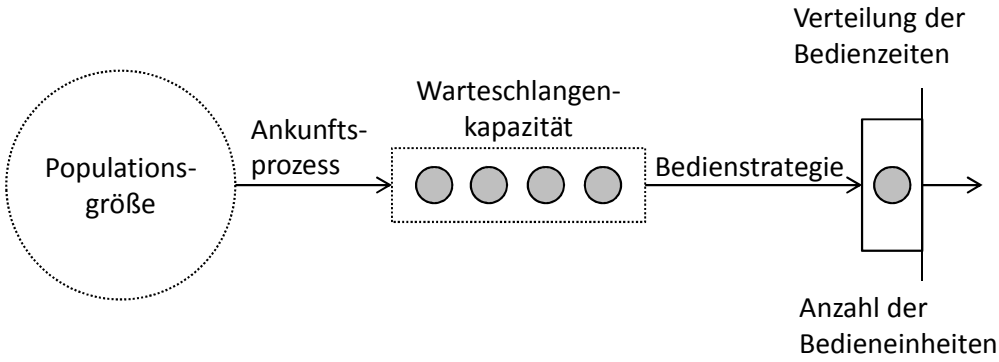


Abbildung 2-13: Schematische Darstellung eines Warteschlangensystems
Quelle: eigene Darstellung

Die Elemente eines Warteschlangensystems werden in Abbildung 2-13 dargestellt:

1. Seien $t_1, t_2, \ldots, t_n$ Ankunftszeiten, dann sind $\tau_j = t_j - t_{j-1}$ die Ankunftszeitenabstände. Der gängigste Ankunftsprozess ist der Poisson-Prozess, bei dem die Ankunftszeitenabstände stochastisch unabhängige, identisch verteilte Zufallsvariablen sowie exponentialverteilt sind.

2. Die Bedienzeit, auch Service-Zeit genannt, ist jene Zeit, die ein Element in der Bedieneinheit verbringt. Genauso wie bei den Ankunftszeitenabständen sind auch die Bedienzeiten eine Folge von stochastisch unabhängigen, identisch verteilten Zufallsvariablen. Auch hier ist die gängigste Verteilung die Exponentialverteilung.

3. Die Anzahl der Bedieneinheiten legt fest, wie viele Bedieneinheiten für jeden angebotenen Dienst zur Verfügung stehen. Jeder angebotene Dienst ist wiederum für sich ein eigenes Warteschlangensystem.

4. Die Warteschlangenkapazität gibt an, wie viele Elemente in einer Warteposition festgehalten werden können. Auch hier kann im Modell eine unbegrenzte Kapazität angenommen werden.

5. Die Populationsgröße der eingehenden Elemente kann entweder begrenzt (geschlossenes System) oder unbegrenzt sein (offenes System).

6. Die Bedienstrategie, auch Warteschlangenstrategie genannt, legt fest, in welche Reihenfolge die anstehenden Elemente abgearbeitet werden. Die wohl gängigste Bedienstrategie ist First-Come-First-Served (FCFS). Hierbei wird das Element zuerst abgearbeitet, das die kürzeste Verweildauer in der Warteschlange aufweist. Andere Strategien sind Last-Come-First-Served (LCFS), Service-in-Random-Order (SIRO), Last-Come-First-Served-with-Preempt-and-Resume (LCFS-PR) oder Round-Robin-Verfahren (RR). Bei letzterem kann jedes Element nur für eine bestimmte Zeit die Bedieneinheit in Anspruch nehmen. Falls mehr Zeit für die Abfertigung benötigt wird, muss sich das Element wiederholt in die Warteschlange einreihen.

Für die Definition eines Warteschlangensystems müssen alle sechs Parameter definiert sein. Die Kendall-Notation (Kendall 1951) beschreibt diese in der Reihenfolge

$$A/S/m/B/K/SD,$$

wobei A der Ankunftsprozess, S der Bedienprozess, m die Anzahl der Bedieneinheiten, B die Warteraumkapazität, K die Populationsgröße und SD die Bedienstrategie sind.

Die gängigsten Verteilungen des Ankunfts- sowie des Bedienprozesses werden durch Buchstaben abgekürzt und sind in folgender Tabelle aufgelistet:

$\boldsymbol{M}$	Exponentialverteilung (M für Markovian)
$\boldsymbol{E_k}$	Erlang-Verteilung mit k Phasen
$\boldsymbol{H_k}$	Hyper-Exponential-Verteilung mit Parameter k
$\boldsymbol{D}$	Deterministische Verteilung
$\boldsymbol{G}$	Allgemeine Verteilung, unspezifiziert
$\boldsymbol{G_I}$	Allgemeine Verteilung, mit unabhängigen Zufallsvariablen

Tabelle 2-3: Gängige Verteilungen bei Warteschlangensystemen
Quelle: eigene Darstellung

Bei der verkürzten Kendall-Notation werden nur die ersten drei Parameter angegeben und die letzten drei Parameter mit $\infty/\infty/1$ angenommen. So beschreibt zum Beispiel die Notation $M/M/1$ das Warteschlangensystem $M/M/1/\infty/\infty/1$.

Eine Reihe von Zufallszahlen und Parametern werden für die Analyse von Warteschlangensystemen verwendet und sind in Abbildung 2-14 anschaulich dargestellt.

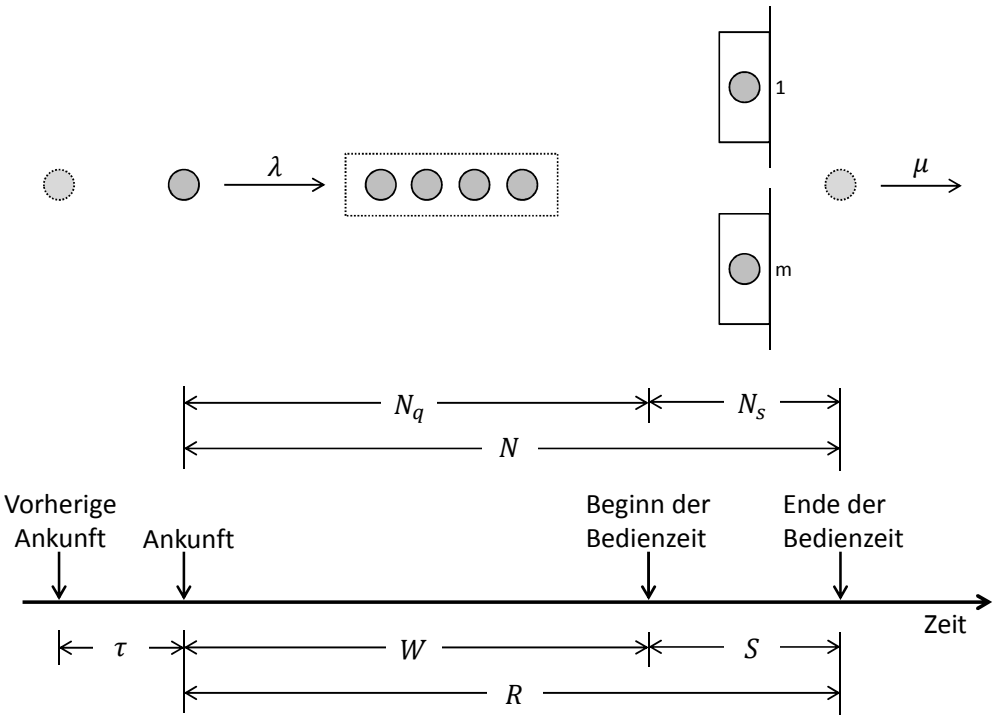


Abbildung 2-14: Zufallszahlen und Parameter in einem Warteschlangensystem
Quelle: eigene Darstellung

τ beschreibt hierbei den Ankunftszeitenabstand, $\lambda = 1/E(\tau)$ (E für Erwartungswert) die durchschnittliche Ankunftsrate, S die Bedienzeit, $\mu = 1/E(S)$ die durchschnittliche Bedienrate für eine Bedieneinheit, m die Anzahl der Bedienstationen (Server), N die Anzahl an Ele-

menten im System, N_q die Anzahl an Elementen in der Warteschlange, N_s die Anzahl an bedienten Elementen, W die Wartezeit und $R = W + S$ die Verweilzeit.

Eines der wichtigsten Gesetze der Warteschlangentheorie ist das Gesetz von Little (1961). Dieses besagt, dass die durchschnittliche Anzahl an Elementen (Aufträgen) im System gleich der Ankunftsrate mal durchschnittliche Verweilzeit ist:

$$N = \lambda \cdot R$$

In Warteschlangensystemen mit begrenzter Warteschlangenkapazität gilt das Gesetz von Little dann, wenn die tatsächliche Ankunftsrate verwendet wird.

Ein weiterer wichtiger Aspekt bei Warteschlangensystemen ist die Auslastung ρ. Sie ist der Anteil der Zeit, in der die Bedienstationen (Server) beschäftigt sind:

$$\rho = \frac{\lambda}{m\mu}$$

Bei einem stabilen Warteschlangensystem mit unbegrenzter Warteschlangenkapazität darf somit die Ankunftsrate nicht größer als die Bedienrate sein. Ansonsten wird von einem instabilen Warteschlangensystem gesprochen.

2.5 Layered-Queueing-Networks

Die in Kapitel 2.4 eingeführten Warteschlangennetze sind eine verbreitete Methode, um die Performance von Rechnersystemen zu analysieren und vorherzusagen. Obwohl sie sehr viele Erfolge im Bereich von traditionellen Rechnern mit Zeitteilverfahren feiern konnten, stoßen sie bei komplexen Interaktionen zwischen Software- und Hardwarekomponenten in Client-Server-Architekturen sehr oft an ihre Grenzen. Das Konzept der stochastischen Rendezvous-Netzwerke (Woodside 1989; Woodside et al. 1995) sowie der „Method of Layers“ (Rolia/Servcik 1995) führten zu einer Erweiterung der Warteschlangennetze, den LQNs, um genau solche komplexen Interaktionen beschreiben zu können.

Ein LQN-Modell wird durch einen azyklischen Graph repräsentiert, wobei Knoten Software-Komponenten und Hardware-Geräte darstellen und Pfeile Service-Anfragen symbolisieren (siehe Abbildung 2-15).

Die Tasks in LQN-Modellen werden in drei Kategorien unterteilt:

- *Reine Clients*: Reine Clients (auch als Referenz-Tasks bezeichnet, da sie das System antreiben) können nur Nachrichten bzw. Anfragen senden.
- *Aktive Server*: Aktive Server können Anfragen sowohl senden als auch empfangen.
- *Reine Server*: Reine Server können lediglich Anfragen empfangen.

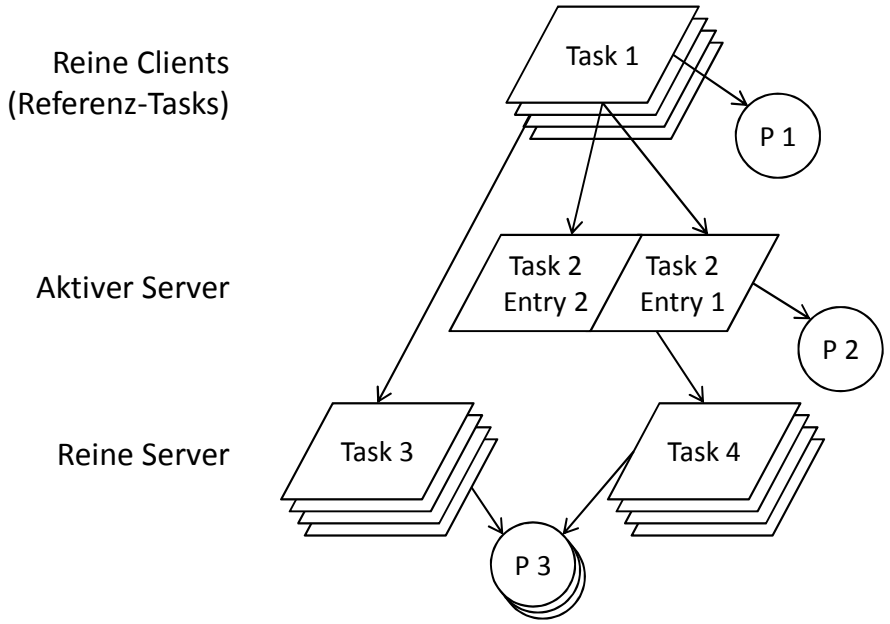


Abbildung 2-15: Beispielhaftes LQN-Modell
Quelle: eigene Darstellung

Die Kategorie der aktiven Server zeigt bereits einen der Hauptunterschiede zwischen Warteschlangenmodellen und LQN-Modellen, da sie Anfragen sowohl empfangen als auch senden können und somit zu Clients weiterer Server werden können. An dieser Stelle sei zudem erwähnt, dass der Begriff „geschichtet“ bei LQN keine strikte Schichtung impliziert, sondern Tasks durchaus andere Tasks aus derselben Schicht bzw. Ebene aufrufen oder Schichten überspringen können (Petriu/Shousha/Jalnapurkar 2000).

2.5.1 Komponenten von LQN-Modellen

In diesem Kapitel werden die einzelnen Komponenten von LQN-Modellen, wie sie beispielsweise in Franks et al. (2012) beschrieben sind, näher erläutert.

Server

Server-Knoten können entweder Tasks oder Prozessoren darstellen. Obwohl es in der LQN-Notation nicht explizit dargestellt wird, beinhaltet jeder Server implizit eine unendliche Warteschlange, um ankommende Anfragen in einer Warteposition aufzunehmen, bis sie bedient werden können. Die standardmäßige Bedienstrategie ist First-Come-First-Served. Weitere Bedienstrategien werden jedoch ebenfalls unterstützt.

Ein Software- oder Hardware-Knoten kann mit einer Multiplizität versehen werden (Einzel- oder Multi-Server). Ein Multi-Server besteht aus mehreren identischen Klonen, die parallel arbeiten, jedoch dieselbe Warteschlange teilen. Die Multiplizität kann ebenso unendlich sein (infiniter Server).

Tasks werden als Parallelogramm dargestellt, Prozessoren als Kreise (siehe Abbildung 2-16).

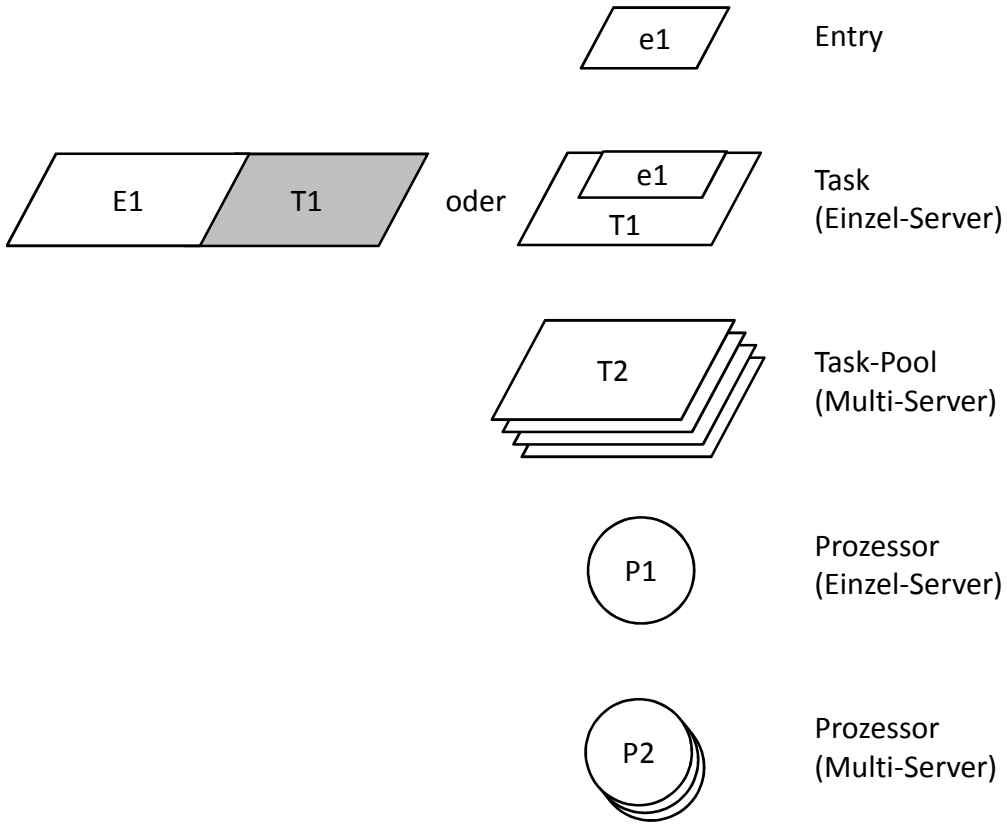


Abbildung 2-16: Tasks und Entries im LQN-Modell
Quelle: eigene Darstellung

Entry

Für einen LQN-Task besteht die Möglichkeit, verschiedene Dienste anzubieten. Diese werden mit sogenannten Entries modelliert (siehe Abbildung 2-16). Eine Entry ist mit einer Adresse oder einem Port eines bestimmten Tasks vergleichbar, bei dem ein bestimmter Dienst angeboten wird. Bei jeder Entry sind die Bedienzeit sowie optionale Aufrufe anderer Dienste definiert. Server mit mehreren Entries haben jedoch nur eine Warteschlange, sodass Anfragen an verschiedenen Diensten dieselbe Warteschlange teilen.

Für Interaktionen, die Gabelungen und Vereinigungen beinhalten, ist eine detaillierte Beschreibung notwendig. Dies wird mit sogenannten Aktivitäten erreicht, die im nächsten Kapitel erläutert werden.

Aktivität

Eine Aktivität ist eine detailliertere Beschreibung einer Entry und befindet sich auf der untersten Modellierungsebene eines LQN-Modells. Sie werden mittels Pfeilen zu einem gerichteten Graphen verbunden, der ein oder mehrere Ausführungsmöglichkeiten darstellt. Die Elemente der Aktivitäten-Notation sind in Abbildung 2-17 aufgezeigt.

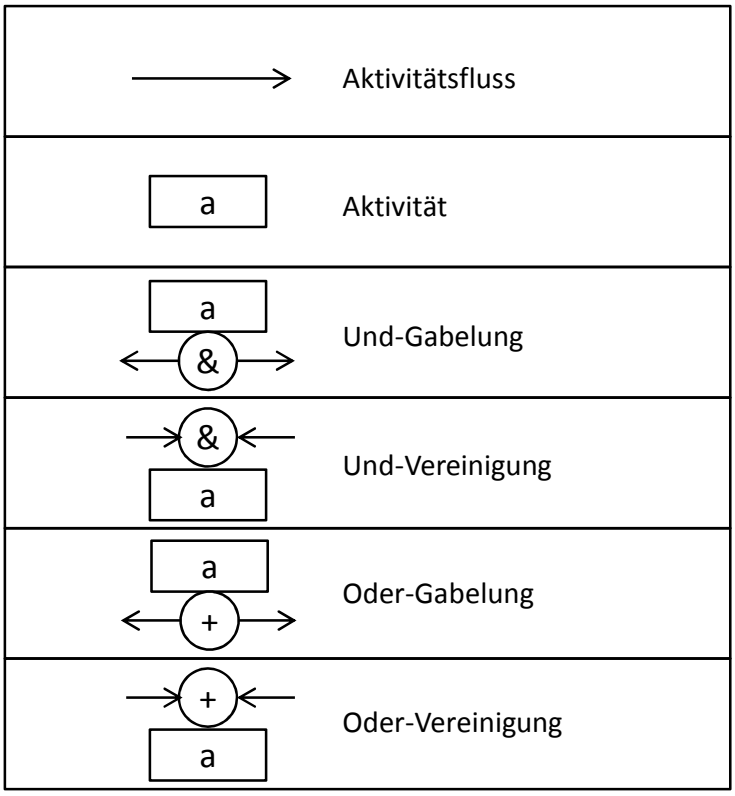


Abbildung 2-17: Notation von Aktivitäten in LQN-Modellen
Quelle: eigene Darstellung

Anfrage

Anfragen (auch Aufrufe genannt) werden in LQN-Modellen als Pfeile dargestellt und verbinden die Entries untereinander. Die Beschriftung der Pfeile gibt dabei die durchschnittliche Anzahl an Anfragen pro Aufruf an. Drei verschiedene Arten von Anfragen können zwischen den Entries ausgetauscht werden (siehe Abbildung 2-18):

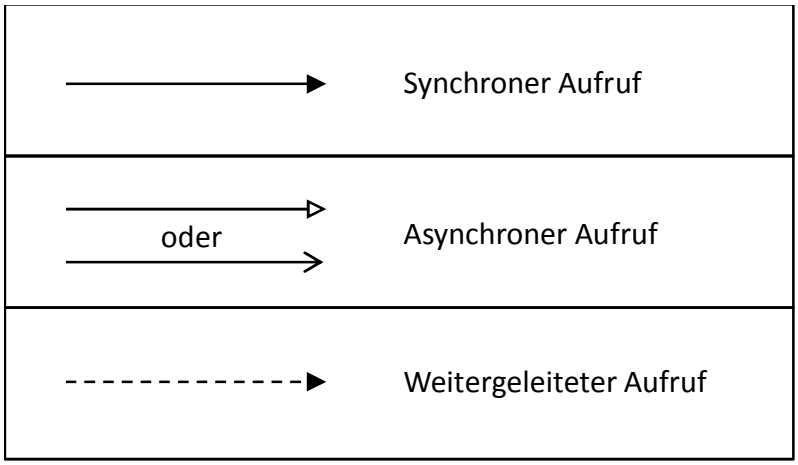


Abbildung 2-18: Notation der Aufrufe in LQN-Modellen
Quelle: eigene Darstellung

1. *Synchroner Aufruf*: Ein synchroner Aufruf stellt eine Anfrage des Clients an einen Server dar, bei dem der Client blockiert bleibt, bis er von dem Anbieter des Dienstes eine Antwort erhält. Sollte der Server gerade beschäftigt sein, wird die Anfrage in die

Warteschlange des Servers gestellt. Sobald die Anfrage bearbeitet wird, werden ein oder mehrere Phasen bei der Server-Entry gestartet. Phase 1 stellt dabei die Phase dar, in der der Client blockiert ist. Am Ende von Phase 1 wird die Antwort an den Client geschickt und nachfolgende Phasen werden gestartet. Diese sind dazu da, eventuelle Nacharbeiten und deren Ressourcenverbrauch abzubilden. Sobald die letzte Phase beendet ist, wird die nächste Anfrage, sofern vorhanden, in der Warteschlange des Servers bedient. Während sämtlicher Phasen können Anfragen an weitere Server gesendet werden, sodass der Server auch als Client fungiert (aktiver Server). Abbildung 2-19 stellt den Verlauf eines synchron bearbeiteten Aufrufs dar.

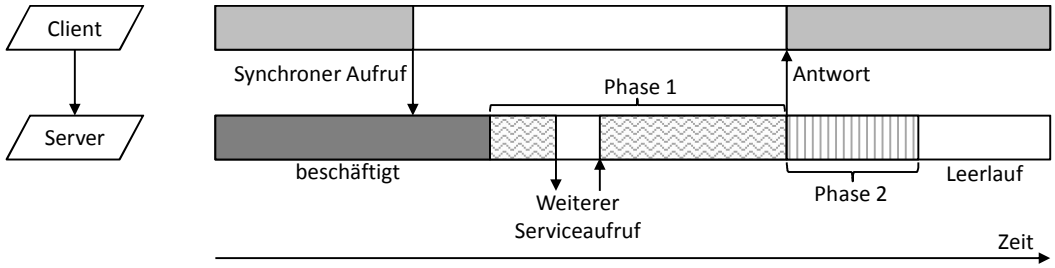


Abbildung 2-19: Ablauf eines synchronen Aufrufs
Quelle: eigene Darstellung

2. *Asynchroner Aufruf*: Bei einem asynchronen Aufruf ist der Client nach dem Senden der Anfrage nicht blockiert und der Server sendet nach Abschluss von Phase 1 keine Antwort an den Client. Diese Aufrufart ist vergleichbar mit dem parallelen Anstoßen von mehreren Java-Threads. Abbildung 2-20 zeigt den Verlauf einer asynchron bearbeiteten Anfrage.

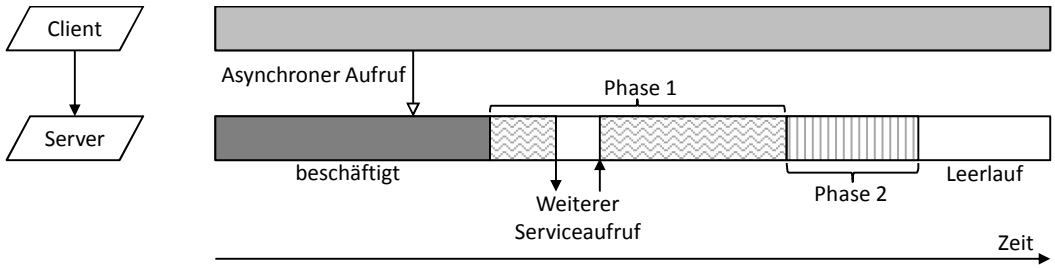


Abbildung 2-20: Ablauf eines asynchronen Aufrufs
Quelle: eigene Darstellung

3. *Weitergeleiteter Aufruf*: Ein weitergeleiteter Aufruf wird mittels eines gestrichelten Pfeils symbolisiert und stellt einen synchronen Aufruf dar, der durch eine Kette von Servern bedient wird. Der Client sendet dabei einen synchronen Aufruf an Server 1, der in Phase 1 beginnt, die Anfrage zu bearbeiten. Am Ende von Phase 1 wird die Anfrage an den nächsten Server weitergeleitet und Server 1 beginnt die restlichen Phasen abzuarbeiten. Der Client hingegen bleibt blockiert, bis die Server-Kette terminiert und der letzte Server die Antwort an den Client schickt. Der Ablauf einer weitergeleiteten Nachricht ist beispielhaft in Abbildung 2-21 dargestellt.

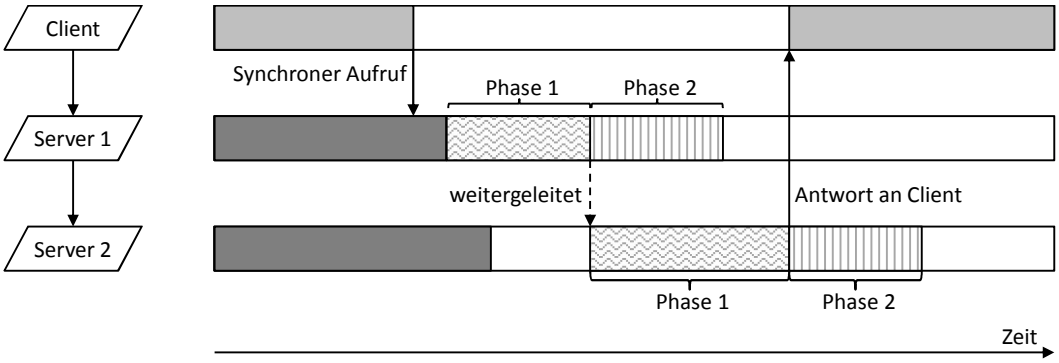


Abbildung 2-21: Ablauf eines weitergeleiteten Aufrufs
Quelle: eigene Darstellung

Eine Phase kann zudem deterministisch oder stochastisch sein und folgt folgenden Annahmen (Petriu et al. 2000):

- Der CPU-Ressourcen-Bedarf einer Phase (welcher als Parameter angegeben wird) ist in exponentialverteilte Teile für jede Anfrage an Servern niedriger Ebenen aufgeteilt. Die durchschnittliche Ausführungszeit ist bei allen Teilen identisch.

- Anfragen an Servern niedriger Ebenen sind geometrisch verteilt. Bei stochastischen Phasen wird dabei der Durchschnitt als Parameter angegeben, bei deterministischen Phasen eine bestimmte Anzahl von Aufrufen.

2.5.2 Parameter in LQN-Modellen

Folgende Parameter sind in einem LQN-Modell angegeben:

- Arten von Clients sowie deren Populationsgröße und Ankunftsrate.

- Die Anzahl an Prozessorknoten sowie Task-Zuweisungen.

- Die Multiplizität der einzelnen Task- oder Prozessorknoten.

- Bedienstrategien der einzelnen Software- und Hardware-Server.

- Die durchschnittliche Bedienzeit für jede Aktivität sowie Phasen der Entries.

- Die durchschnittliche Anzahl an synchronen, asynchronen und weitergeleiteten Aufrufen, welche von den verschiedenen Phasen einer Entry oder Aktivität gesendet werden.

2.5.3 Transformationsregeln

In vielen Software-Systemen existieren Zyklen, beispielsweise bei einem asynchronen Aufruf einer Anfrage, die der Server nach deren Abarbeitung beantwortet. LQN-Modelle fordern aber einen azyklischen Graphen, sodass eine Transformation notwendig wird, welche durch Semaphore erreicht wird (Shousha et al. 1998). Die Semaphore ermöglichen einen exklusiven Lese- oder Schreibzugriff, indem ein zusätzlicher Task mit einfacher Multiplizität und einer Bedienzeitdauer von Null modelliert wird, der den asynchronen Aufruf als synchronen Aufruf

an die entsprechende Entry weitergibt. Dadurch bleibt der Semaphor für die Dauer der Abarbeitung „beschäftigt“ und bearbeitet keine weiteren Anfragen. Damit diese Transformation nicht mehr durchgeführt werden muss, wurde 2011 ein eigener Semaphor-Task in die LQN-Modellierung eingeführt (Franks 2011).

Des Weiteren können offene Warteschlangensysteme nicht nativ abgebildet werden. Shousha et al. (1998) schlagen deshalb eine Transformation des offenen Systems in ein geschlossenes Modell vor, indem die Ankunftsrate λ anhand von reinen Clients mit sehr geringer Bedienzeit (damit das Performance-Verhalten nicht verfälscht wird) abgebildet wird. Zudem müssen asynchrone Aufrufe in weitergeleitete Aufrufe transformiert werden, damit Anfragen nur dann akzeptiert werden, wenn die notwendige Kapazität vorhanden ist.

2.5.4 Lösen und Simulieren von LQN-Modellen

Für das analytische Lösen und Simulieren von LQN-Modellen wurden von der „Real Time and Distributed Systems Group“[3] zwei Werkzeuge entwickelt, und zwar der LQNS (LQN-Solver) und der LQsim (LQN-Simulator).

2.5.4.1 LQN-Solver

Der LQNS, der von Greg Franks entwickelt wurde (Franks 1999), löst LQN-Modelle nach einem analytischen Verfahren. Die LQN-Schichten werden dabei in separate Warteschlangennetze (Teilmodelle) aufgegliedert und mittels „Mean Value Analysis“ (Petriu 1994), kurz MVA, gelöst. Das MVA-Ergebnis eines Teilmodells wird dann zur Feinabstimmung der verbundenen Teilmodelle verwendet. Am Ende der Kalibrierung wird die MVA erneut durchgeführt, solange, bis eine maximale Anzahl an Iterationen erreicht ist oder die Ergebnisse gegen einen bestimmten, vom Benutzer angegebenen Wert konvergieren.

Der LQNS kann verschiedene Methoden zur Schichtung der Teilmodelle verwenden. Standardmäßig werden die einzelnen Schichten aus so vielen Servern wie möglich gebildet. Beim „Loose Layering“, einer weiteren Schichtungsmethode, besteht jede Schicht nur aus einem einzelnen Server, während „Strict Layering“, die dritte Schichtungsmethode, auf der „Method of Layers“ (Rolia/Servcik 1995) basiert.

Die wohl wichtigste Einschränkung beim LQN-Solver ist die Forderung, dass Modelle hierarchisch zerlegt werden können. Somit können Gabelungen und Vereinigungen nur dann gelöst werden, wenn sie in ein und demselben Task auftreten.

Das LQNS-Meta-Modell ist in Abbildung 2-22 abgebildet. Neben den bereits eingeführten Elementen Prozessor, Task, Aktivität, Entry und Aufruf beinhaltet das Meta-Modell folgende Komponenten:

- Eine *Gruppe* (von Tasks) kann für ein spezielles Scheduling-Verfahren (Li/Franks 2009) definiert werden.
- Eine *Präzedenz* verbindet Aktivitäten.

[3] Carleton University, Ottawa, siehe http://www.sce.carleton.ca/rads/rads.html

– Ein (zählender) *Semaphor* ist ein spezieller Task-Typ zur Modellierung von Sperrmechanismen sowie Puffern.

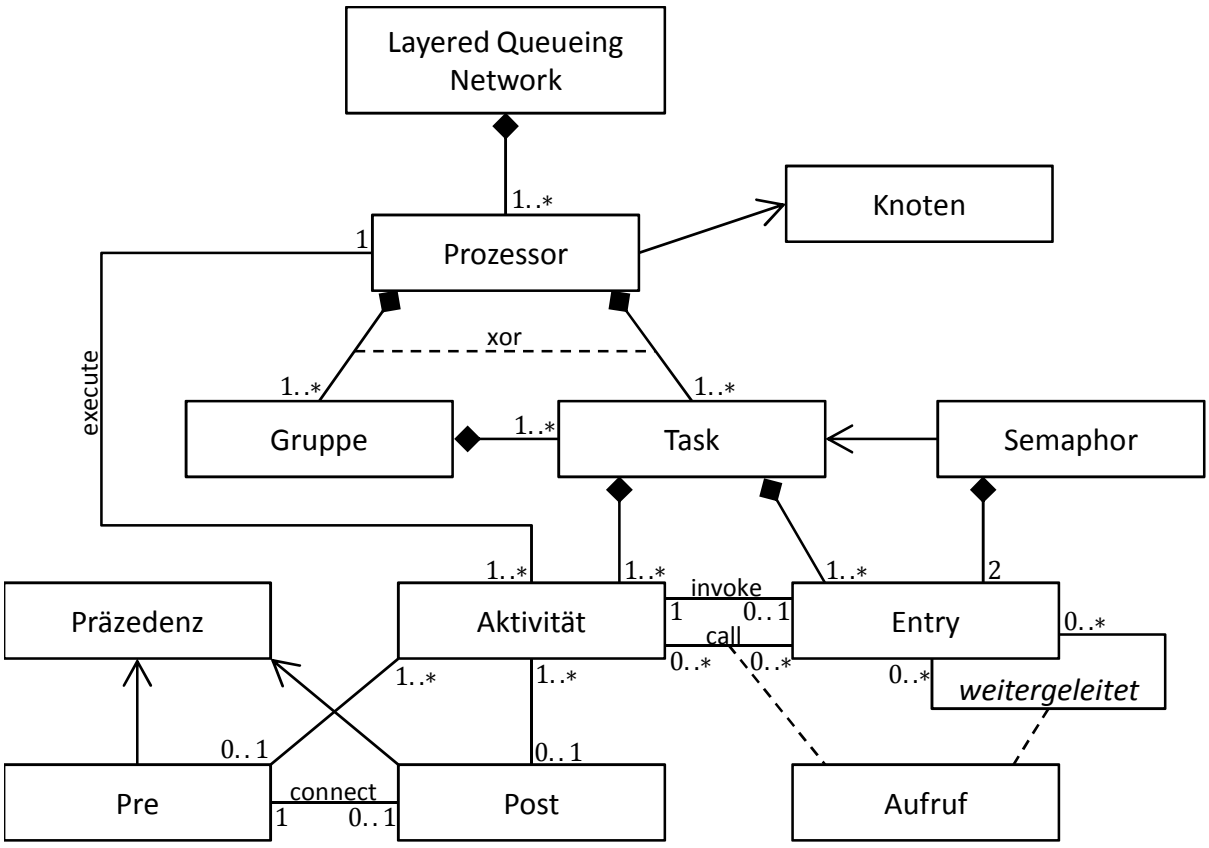


Abbildung 2-22: LQNS-Meta-Modell
Quelle: Franks (2011)

2.5.4.2 Layered Queueing Simulator (LQsim)

Der Layered Queueing Simulator (LQsim), der auch in dieser Arbeit zur Simulation des LQN-Modells verwendet wird, greift auf vordefinierte Schablonen zurück, die den Elementen des LQNS-Meta-Modells (siehe Abbildung 2-22) entsprechen. Das daraus resultierende Simulationsmodell entspricht dem eingegebenen LQN-Modell. Die verwendeten Templates greifen auf Elemente des Parasol-Simulators (Neilson 1991) zurück.

Parasol ist ein ereignisdiskreter Simulator zur Simulation von komplexen, verteilten Systemen zur Leistungsanalyse sowie zur Emulation der Ausführung von Programmen in nebenläufigen Systemen (Neilson 1991). Damit verteilte Systeme einfach modelliert werden können, stellt Parasol eine virtuelle Maschine zur Verfügung, die Komponenten wie Hardware-Locks, System-Busse oder Prozessoren anbietet. Die von LQsim verwendeten Elemente von Parasol sind in Abbildung 2-23 dargestellt und lassen sich wie folgt beschreiben:

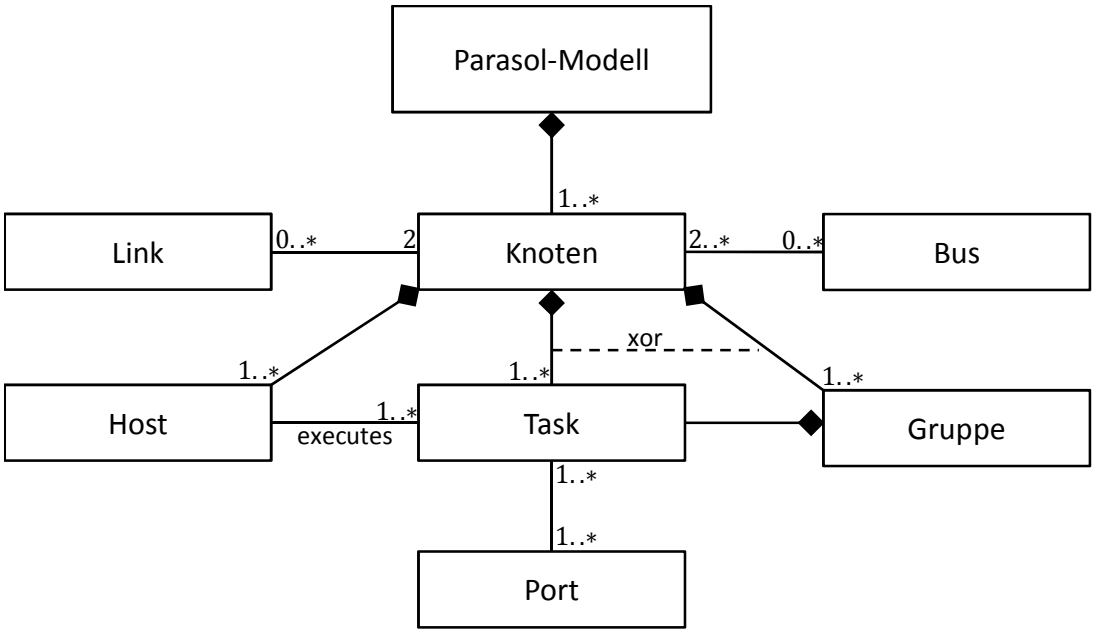


Abbildung 2-23: Metamodell von Parasol
Quelle: Li/Franks (2009)

- *Knoten*: Ein Knoten ist eine Sammlung von einen oder mehreren Hosts, die durch Busse oder Links verknüpft sind. Es können entweder einzelne CPUs oder Multi-Prozessoren modelliert werden.

- *Bus*: Ein Bus ist ein Kommunikationskanal zur Verknüpfung von einen oder mehreren Knoten. Die Kommunikation erfolgt bidirektional.

- *Link*: Ein Link verbindet zwei Knoten für eine unidirektionale Kommunikation.

- *Gruppe*: Eine *Gruppe* (von Tasks) wird, ebenso wie bei dem LQNS-Meta-Modell, für ein spezielles Scheduling-Verfahren verwendet (Li/Franks 2009).

- *Host*: Ein Host bzw. dessen Prozessoren führen die Tasks aus.

- *Task*: Tasks kommunizieren untereinander, indem sie Nachrichten an den entsprechenden Port senden.

- *Port*: Jeder Task besitzt einen Standard-Port für den Empfang der zu versendenden Nachrichten. Weitere Ports können je nach Bedarf definiert werden.

Die Tasks von Parasol werden von LQsim verwendet, um sämtliche Task-Typen in dem LQN-Modell zu modellieren. Zum besseren Verständnis ist die Hauptschleife des Task-Templates im Listing 2-1 dargestellt, der periodisch bei den Referenz-Tasks (reinen Clients) oder immer dann, wenn eine Nachricht bei einem der anderen Tasks ankommt, aufgerufen wird.

```
1  procedure server_cycle( mesg msg_in )
2  begin
3     bool not_done := true;
4     while not_done do
5        compute(msg_in.entry);
6        entry dst := choose(msg_in.entry);
7        if dst /= nil then
8           mesg msg_out;
9           msg_out.entry := dst;
10          msg_out.reply_to := reply_port;
11          send(dst.std_port, msg_out);
12          receive(reply_port)
13       else
14          not_done := false
15       fi
16    od;
17    send(msg_in.reply_to, nil);
18 end
```

Listing 2-1: Hauptschleife des Task-Templates
Quelle: Franks (2011)

Die Entries des LQN-Modells werden mittels unterschiedlicher Nachrichtentypen modelliert. Dies ist in dem Programmablauf in Zeile 5 und 6 bei dem Entry-Attribut der Eingangsnachricht ersichtlich. Über dieses Attribut wird unter anderem auch die Bedienzeit für diese Nachricht gesetzt. Die Schleife in dem Programmablauf bedient alle drei Typen von Aufrufen (synchron, asynchron, weitergeleitet). Bei asynchronen Aufrufen wird beispielsweise Zeile 12, der Empfang der Antwort, sowie Zeile 17, eine optionale Antwort, übersprungen (vgl. Franks 2011).

Als Eingabedatei dient dem LQsim (ebenso wie dem LQNS) eine Textdatei, die sowohl die einzelnen Komponentendefinitionen, als auch deren Parametrisierung enthält. Die einzelnen Elemente folgen einer vordefinierten Reihenfolge, und zwar beginnt die Datei mit einem allgemeinen Informationsabschnitt, gefolgt von den einzelnen Komponenten Prozessoren, Tasks, Entries und Aktivitäten.

2.6 Benchmarks

In dieser Arbeit wird die Annahme geteilt, dass Applikations-Benchmarks Werte mit einer hohen Übertragbarkeit und Praxisrelevanz liefern (Lilja 2000, 116). Die Anforderungen an den verwendeten Workload bei der Evaluation des zu entwickelnden LQN-Modells sollen daher von Applikations-Benchmarks abgeleitet werden. Für einen Workload aus der Praxis, der für einen Applikations-Benchmark erforderlich ist, sorgt eine am SAP UCC angebotene Fallstudie, die von den Dozenten im SAP-UA-Programm verwendet wird (siehe Kapitel 5.1). Für eine höhere Übertragbarkeit werden die Performance-Messungen an einem Portalsystem mit einer Standardkonfiguration durchgeführt. In den folgenden Abschnitten werden nun die verschiedenen Benchmark-Typen und ihre wesentlichen Merkmale vorgestellt.

Der Begriff Benchmark ist in vielen verschiedenen Wissenschaftsbereichen verbreitet (Wöhe 2005). In der Informatik dient er der Messung von Systemen, Hardware- und Softwarekomponenten und erlaubt durch einen Vergleich der Ergebnisse, eine Aussage über deren Performance zu geben. Nach Prechelt (2001, 43) ist ein Benchmark „[…] ein genau definierter Anwendungsfall für ein Programm oder eine Methode, der eine Vorschrift einschließt, wie das Ergebnis der Anwendung quantifiziert werden kann. Die resultierende, beliebig reproduzierbare Größe heißt Benchmark-Ergebnis". Benchmark-Werte spiegeln somit die Leistungsfähigkeit des Systems wieder. Der Unterschied zu Performance-Messungen im Allgemeinen liegt darin, dass Benchmarks standardisierten Kriterien unterliegen. Benchmark-Ergebnisse werden hauptsächlich dazu verwendet, den eigenen Fortschritt zu messen und einen Vergleich mit der Konkurrenz durchzuführen (vgl. Hoetzel et al. 1998).

Der Begriff des Benchmarking kommt ursprünglich aus der Landvermessung, bei der mittels Benchmarking ein Festpunkt gesetzt wird, der als Ausgangs- und Bezugspunkt für weitere Messungen dienen soll. In der IT-Branche wurde der Begriff bereits Ende der siebziger Jahre durch eine umfangreiche Studie von Xerox geprägt (Söbbing 2006, 445). Benchmarking ist „[...] das Vorgehen, ein oder mehrere Benchmark-Ergebnisse zu ermitteln und damit die Leistungsfähigkeit eines Programms oder einer Methode zu charakterisieren." (Prechelt 2001, 43). Im wirtschaftswissenschaftlichen Bereich gilt Benchmarking als ein Wettbewerbsanalyseinstrument zum „[…] kontinuierliche[n] Vergleich von Produkten, Dienstleistungen sowie Prozessen und Methoden mit (mehreren) Unternehmen, um die Leistungslücke zum sog. Klassenbesten (Unternehmen, die Prozesse, Methoden etc. hervorragend beherrschen) systematisch zu schließen" (Alisch 2004).

Technische Benchmarks von Computersystemen können unter dem Einfluss einer generierten Last, CPU-Leistung, Grafikfähigkeit sowie Input/Output-Verhalten, die Leistung von Rechnernetzen und vieles mehr messen (Siebert/Kempf/Maßalski 2008, 9). In den Anfängen wurden als Leistungskennzahl Rechenoperationen pro Zeiteinheit angegeben. So konnte beispielsweise die erste auf der Von-Neumann-Architektur basierende Rechenmaschine einen Rechenschritt pro Sekunde durchführen (Knuth 1970). Im weiteren Verlauf wurden bald „Floating Point Operations per Second" (FLOPS) als Maßeinheit verwendet, welches der zentralen Bedeutung der Berechnung von Fließkommazahlen geschuldet ist (Henning 2006). Dieser Wert entspricht dem rechnerisch erzielbaren Durchsatz eines Rechners, welcher in der Praxis aufgrund von diversen Einflussfaktoren (Ressourcenengpässe, Konfigurationseinschränkungen) selten erreicht wird (Gray 1991). Daraus entstand die Notwendigkeit, weitere Benchmarks zu entwickeln (Weicker 1990).

2.6.1 Anforderungen an Benchmarks

Die Grundidee des Benchmarkings ist die Feststellung und das Auffinden von Unterschieden in den Prozessen, Systemen und Methoden. Durch einen Vergleich durch jeweils gleiche Benchmarks können Verbesserungsmöglichkeiten aufgezeigt werden. Es ist wichtig zu wissen, welche Anforderungen an einen Benchmark bestehen, wie zum Beispiel die Lauffähigkeit auf allen eingesetzten Plattformen eines Vergleichunternehmens. Die folgenden Anforderungen an Benchmarks sollen nach Versick (2010, 31ff.) erfüllt sein, um sie nutzbringend einsetzen zu können:

- *Portabilität*: Der Benchmark muss auf allen Plattformen, die miteinander verglichen werden, lauffähig sein.
- *Repräsentativität*: Unabhängig von der Anzahl an Applikationen müssen Benchmarks repräsentative Ergebnisse liefern.
- *Wiederholbarkeit*: Bei identischen Bedingungen müssen Benchmarks gleichbleibende Ergebnisse liefern.
- *Verifizierbarkeit*: Ergebnisse der Benchmark-Läufe müssen nachvollziehbar und überprüfbar sein.

Des Weiteren sollen Benchmarks eine gute Skalierbarkeit aufweisen und somit auch bei steigender Systemleistung relevante Ergebnisse liefern. Zudem soll ein Benchmark den Nutzer darin unterstützen, das System zu verstehen, Quellen der Leistungsunterschied aufzudecken und Optimierungen durchzuführen (Heinrich/Lehner 2005, 462).

2.6.2 Benchmark-Standards

Wie bereits in Kapitel 2.3.1 angeführt, gibt es keinen offiziellen Standards für Performance-Metriken, allerdings wurde eine Reihe von Standard-Benchmarks für bestimmte Szenarien entwickelt, die zum Leistungsvergleich weltweit Verwendung finden. Zwei bekannte Organisationen, die selbstentwickelte Benchmarks freigeben sowie Ergebnisse sammeln und veröffentlichen, sind SPEC[4] (Standard Performance Evaluation Corporation) und TPC[5] (Transaction Processing Performance Council).

Die SPEC-Benchmarks betreffen insbesondere technische Anwendungsgebiete, mit denen vorwiegend die Leistung von Arbeitsplatzrechnern und Mikroprozessoren ermittelt werden. Zu einem der bekanntesten SPEC-Benchmarks zählt der CPU-Benchmark, womit die Rechenleistung unabhängig von Betriebssystem, Architektur etc. verglichen werden kann (Rechenberg/Pomberger/Pirklbauer 2006, 460). Ziel der SPEC-Organisation ist es, „die Industrie mit realistischen Meßlatten auszurüsten, um die Leistung fortschrittlicher Rechensysteme zu messen“ (Haas/Zorn 1995, 215).

Das Ziel des internationalen Gremiums TPC ist die Definition von transaktionsorientierten Benchmarks für nachprüfbare, objektive Leistungsdaten. Ähnlich wie bei SPEC müssen sowohl neue Anwendungen als auch eng vernetzte und kooperierende Systeme berücksichtigt werden. TPC verwaltet zum einen die Ergebnisse von Datenbankanwendungen (dazu zählen entscheidungsunterstützende Systeme sowie Transaktionsverarbeitung), zum anderen Paper-and-Pencil-Benchmarks. Paper-and-Pencil-Benchmarks sind Benchmarks, die Problemklassen algorithmisch darstellen und die Implementierung unter bestimmten Regeln den Entwicklern überlassen. Die ermittelten Leistungsmaße werden normalerweise in der Transaktionsrate TPS (Transaktionen pro Sekunde) oder in Preis-/Leistungskenngrößen angegeben. Ein großer

[4] Standard Performance Evaluation Corporation, http://www.spec.org

[5] Transaction Processing Performance Council, http://www.tpc.org

Vorteil von TPC-Benchmarks ist, dass sie systemunabhängig sind, dafür aber auch sehr komplex und aufwendig (Haas/Zorn 1995, 216; Rechenberg/Pomberger/Pirklbauer 2006, 460).

2.6.3 Benchmark-Typen

Nachfolgend wird eine grundsätzliche Einordnung der Benchmarks durchgeführt (vgl. Lilja 2000, 112-116).

Instruction-Mix

Ein *Instruction-Mix* spezifiziert die Arten und die relative Häufigkeit der durchgeführten Befehle, wie sie auch im realen Workload vorkommen (Hu/Gorton 1997). Danach wird die mittlere Operationszeit T aus den Operationszeiten von n Befehlen berechnet:

$$T = \sum_{i=1}^{n} p_i \cdot t_i$$

t_i stellt hierbei die Operationszeit des Befehls i dar, p_i die relative Häufigkeit des Auftretens (Gewicht) des Befehls i, wobei gelten muss:

$$\sum_{i=1}^{n} p_i = 1, \qquad p_i \leq 1$$

Ein großer Vorteil der Instruktion-Mixes ist die relativ einfache Entwicklung. Allerdings können keine heterogenen Architekturen betrachtet werden (beispielsweise CPUs mit RISC- und CISC-Architektur). Zudem kann die Leistungsfähigkeit des Gesamtsystems nicht bewertet werden, solange nicht die CPU den Engpass des Systems darstellt, da diese Art von Benchmarks lediglich die Geschwindigkeit von CPUs misst. Nicht zuletzt erschweren es weiterentwickelte Technologien, wie zum Beispiel Caches oder Pipelines, die Leistung einer Recheneinheit mittels Instruction-Mixes richtig zu bewerten. Einer der bekanntesten Vertreter dieses Benchmark-Typs ist der von IBM entwickelte „Gibson Mix“ (Gibson 1970). Aufgrund der genannten Nachteile ist dieser Benchmark-Typ allerdings weitestgehend obsolet (Hu/Gorton 1997).

Synthetische Benchmarks

Synthetische Benchmarks sind Programme, die dazu dienen, realen Workload zu simulieren. Sie konsumieren eine bestimmte Menge an Systemressourcen oder senden bestimmte Anfragen an das System. Im SAP-Umfeld stellt der „Zachmanntest“ einen synthetischen Benchmark für Hauptspeicher-Operationen dar, allerdings werden I/O-Operationen nicht behandelt (Boegelsack/Wittges/Krcmar 2010; Kühnemund 2007). Weitere Vertreter von synthetischen Benchmarks sind Programme wie „Whetstone“ und „Dhrystone“ (Weiss 1993) sowie „NFSStone“ (Shein/Callahan/Woodbuy 1989), ein Benchmark für das NFS („Network File System“), oder „IOStone“ (Park/Becker 1990), einem I/O-Benchmark.

Der Hauptvorteil von synthetischen Benchmarks liegt vor allem in der schnellen Implementierung und schnellen Anpassbarkeit, sodass ein großes Spektrum von realen Lastszenarien mittels einer Menge von Kontrollparametern abgedeckt werden können. Allerdings stellen sie

oft ein sehr einfaches Abbild des realen Lastmusters dar und können somit nicht immer die gegebenen Systemeigenschaften darstellen, wie zum Beispiel die akkurate Abbildung von Festplatten-Caches (Hu/Gorton 1997; Jain 1991). Eine Übertragbarkeit der Ergebnisse in den Alltagsbetrieb wird vom synthetischen Benchmark daher nicht erwartet (Jehle 2010).

Kernel-Benchmarks

Zeitgleich mit den synthetischen Benchmarks wurden Kernel-Benchmarks entwickelt. Dabei werden die rechenintensiven Teile des Quellcodes extrahiert und lauffähig gemacht. Periphere Zusätze, wie zum Beispiel die Benutzerinteraktion, werden außen vorgelassen. Anstelle der Benutzerschnittstellen werden zur Reproduzierbarkeit definierte Parameter übergeben (Jehle 2010). Vertreter von Kernel-Benchmarks sind beispielsweise der „Tower of Hanoi“ (Staples 1987), „Livermore Loops“ (McMahon 1986) oder „LinPack“. (Dongarra et al. 1979). LinPack wird heute noch verwendet, um die Leistungsfähigkeit von Supercomputern zu messen. Die Ergebnisse werden halbjährlich in einer Top-500-Liste[6] veröffentlicht. Als Basis des LinPack-Benchmarks, einem alten FORTRAN-Programm, dient die Lösung des Gleichungssystems $Ax = b$ mittels Gauß-Elimination. Der Einsatz des LinPack-Benchmarks erlaubt die Darstellung der Entwicklung und bildet eine Rangfolge, allerdings können keine Aussagen über die eigentliche Systemleistung getroffen werden.

Applikations-Benchmarks

Viele Computersysteme wurden für ein bestimmtes Anwendungsszenario konzipiert, wie zum Beispiel Reservierungssysteme, Bankensysteme oder allgemein transaktionsorientierte Systeme. Für die Evaluation der Performance solcher Systeme reichen synthetische Programme nicht aus. Daher entsteht die Notwendigkeit, reale Funktionen, die vom System ausgeführt werden, in den Benchmark zu implementieren.

Seit den siebziger Jahren wurden Debit-Credit-Benchmarks, eine spezielle Form von Applikations-Benchmarks, immer beliebter. Diese wurden dahingehend entworfen, transaktionsorientierte Systeme zu vergleichen. Die dabei am häufigsten verwendete Metrik ist die Preis-Performance-Rate. Der Preis stellt hierbei sämtliche Kosten für den Kauf, die Installation und die Wartung des Systems (sowohl Hardware als auch Software) über einen bestimmten Zeitraum hinweg dar. Die Performance wird mittels TPS gemessen. Jede Transaktion beinhaltet Datenbanklese- und –schreiboperationen. Der erste bekannte Debit-Credit-Benchmark wurde als *TP1* bekannt (Anonymous et al. 1985) und gilt als De-Facto-Standard für transaktionsorientierte Systeme und relationale Datenbanken. Aus dem TP1-Benchmark entstand der von dem Transactions Processing Performance Council (TPC) entwickelte TPC-A-Benchmark (TPC 1989). Dieser fordert, dass alle Anfragen von realen Terminals gesendet werden und die durchschnittliche Ankunftszeit von Anfragen bei zehn Sekunden liegt. Im Gegensatz dazu generiert der später entwickelte TPC-B-Benchmark (TPC 1990) so viele Anfragen wie möglich mittels eines internen Treibers.

Der Hauptvorteil von Applikations-Benchmarks liegt vor allem darin, dass sie auf dem tatsächlichen System laufen und echte Funktionen ausführen. Dabei bieten sie die Möglichkeit,

[6] Siehe auch http://top500.org

echte Systemeffekte zu analysieren, wie zum Beispiel administrative und verarbeitende Funktionen der zentralen Recheneinheit oder die tatsächliche Geschwindigkeit der verschiedenen I/O-Geräte. Bei einer treffenden Repräsentation des realen Workloads bieten Applikations-Benchmarks eine genaue Messung der Leistungsfähigkeit des Systems, da reale, sprich repräsentative Funktionen auf dem tatsächlichen System ausgeführt werden. Sind sie hingegen schlecht konfiguriert oder unpassend gewählt, ist ihre Aussagekraft verschwindend gering, da beispielsweise ein Engpass aufgezeigt wird, der im realen Lastszenario nicht auftritt oder umgekehrt (Joslin 1965).

SAP-Benchmarks

SAP-Standard-Applikations-Benchmarks unterscheiden sich von anderen IT-Benchmarks, indem sie als Messgröße keine Standardmetrik wie zum Beispiel TPS verwenden, sondern die Messung der Leistungsfähigkeit eines Systems mit der produktiv verwendeten Geschäftsanwendung, welche bei Kundenimplementierungen verwendet wird, vereinen. Mittels der Durchführung von kompletten Geschäftsprozessen werden anwendungsspezifische und geschäftsrelevante Performance-Indikatoren aufgezeigt. So werden zum Beispiel die maximalen Benutzerinteraktionen pro Stunde oder komplett durchgeführte Auftragspositionen pro Stunde gemessen (Marquard/Götz 2008).

Die Architektur von SAP-Standard-Applikations-Benchmarks wurde dahingehend ausgelegt, dass es für das System keine Rolle spielt, ob es sich um reale oder simulierte Benutzer handelt. Beispielsweise bleiben alle Systemüberwachungsprogramme, die auch im produktiven Einsatz Verwendung finden, eingeschaltet. Der Lasttreiber (Benchmark-Treiber) wird auf einem externen System ausgeführt, sodass keine zusätzliche Last auf dem unter Beobachtung stehenden System entsteht (Janssen/Marquard 2007).

Der erzielte Durchsatz wird anschließend auf die einzelnen Hardwarekomponenten, wie z.B. CPU und Hauptspeicher, abgebildet und als eine hardwareunabhängige Einheit dargestellt, den sogenannten SAP Application-Performance-Standard (SAPS) (Kay 2001). Diese sind von dem SAP-Sales-and-Distribution-Benchmark (SD-Benchmark) abgeleitet und sind definiert als:

$$100\ SAPS = 2000\ vollständig\ durchgeführte\ Auftragspositionen\ pro\ Stunde$$

In technischen Werten ausgedrückt entsprechen 100 SAPS 6.000 Dialogschritten mit 2.000 Verbuchungen bzw. 2.400 SAP-Transaktionen. Da sich diese Einheit über die Jahre hinweg nicht geändert hat, kann die Leistungsentwicklung über Jahrzehnte hinweg beobachtet werden (Schneider 2008, 276f.).

Die SAP-Standard-Applikations-Benchmarks haben zu einem Wettbewerb der Hersteller geführt, den höchsten SAPS-Wert zu erreichen (Prior 2003). Die Benchmark-Ergebnisse wurden unter anderem dadurch optimiert, dass durch die Verwendung von Caches Zugriffe auf den Hintergrundspeicher nicht mehr notwendig sind. Zudem werden alle Operationen auf Hauptspeicherzugriffe und CPU-Operationen reduziert, sodass keine langsamen I/O-Operationen anfallen. Dies erhöht zwar die Anzahl an durchgeführten Auftragspositionen pro Stunde und verbessert somit das Benchmark-Ergebnis, allerdings leidet darunter die Repräsentativität eines realen Workloads (Jehle 2010).

Bei dem ersten Benchmark für J2EE-basierte SAP-Systeme handelte es sich um einen Kerneltest, der verschiedene Berechnungen innerhalb der Applikationsschicht ausführt und deren Bearbeitungszeit misst. Später wurde ein weiterer J2EE-Benchmarks für Java-basierte SAP-Lösungen entwickelt (SAP 2011h), der die Grundfunktionen eines Portalsystems untersucht. Allerdings kann dieser Benchmark kaum für die eigenen Bedürfnisse angepasst werden, da die ausgeführten Funktionen sowie deren Reihenfolge und zusätzliche Parameter wie zum Beispiel Denkzeiten der anfragesendenden Benutzer nicht konfiguriert werden können.

Für individuell gestaltete Lastszenarien verweist SAP auf die Verwendung von Werkzeugen von Drittherstellern (Jehle 2010, 45f.). Infolgedessen ist bei der Entscheidung über das zu verwendende Lastmuster für die Evaluation des LQN-Modells die Wahl auf die bereits bei Jehle (2010) eingesetzte Portalfallstudie gefallen.

2.7 Kapazitätsplanung

Der in dieser Arbeit vorgestellte Ansatz zur Modellierung und Simulation eines SAP-Netweaver-Portal-Systems kann für die Kapazitätsplanung in Hinblick auf das benötigte Performance-Modell und die daraus resultierende Performance-Vorhersage verwendet werden. Der Ansatz liefert somit einen Beitrag für die technische Komponente der Kapazitätsplanung. Mit der Erstellung eines Kostenmodells kann eine Kosten-Performance-Analyse durchgeführt werden, die als Grundlage für einen Konfigurations-, Investment- und Personalplan dienen kann. Obwohl die Kapazitätsplanung in dieser Arbeit nicht detailliert behandelt wird, werden im folgenden Abschnitt die wesentlichen Merkmale kurz dargestellt.

Die Kapazitätsplanung beschäftigt sich mit der Frage, wann die Systemauslastung gesättigt ist und welche kostengünstigen Methoden den Sättigungspunkt am längsten hinauszögern (Menascé/Almeida 2002). Bei der Kapazitätsplanung versucht man somit, Hardware- und Softwarekomponenten sowie entsprechende Konfigurationen so zu wählen, dass sie bestimmten Performance-Charakteristiken genügen. Begrenzte Leistungskapazitäten und ein steigender Bedarf führen früher oder später zu einer Sättigung, bei der das System bzw. die Systemlandschaft nahe der Vollauslastung arbeitet. Dies führt wiederum zu einem Anstieg der Antwortzeiten und somit zur Unzufriedenheit der Benutzer. Kapazitätsplanung erfordert eine sehr gute Kenntnis der verwendeten Software. Daher bieten viele große Hersteller eigene Werkzeuge und Leitfäden an, wie ihre Software konfiguriert werden kann.

Menascé/Almeida (2002) haben einen allgemeinen Kapazitätsplanungsprozess entwickelt (siehe Abbildung 2-24), der häufig zum Bemessen von Systemen verwendet wird (Risse 2006), wie zum Beispiel bei der Konfiguration eines datenintensiven, verteilten Informationssystems für Erdbeobachtungen (Gomaa/Menascé/Kerschberg 1996).

Zu Beginn wird die Systemumgebung analysiert und ein Verständnis für die gegebene Hardware, Software, Konfigurationen, etc. geschaffen. Danach wird der Workload charakterisiert, ein Modell entwickelt, anschließend validiert und ggf. in einem iterativen Prozess kalibriert und erneut validiert. Das Workload-Modell wird im nächsten Schritt für eine Workload-Prognose verwendet. Dieselben Schritte werden bei der Erstellung des Performance-Modells durchgeführt. Dabei wird in sehr vielen Fällen auf Warteschlangenetzwerke zurückgegriffen, die eine flexible und umfangreiche Methode zur Abbildung von diversen Systemen darstellen (Lazowska et al. 1984; Robertazzi 2000).

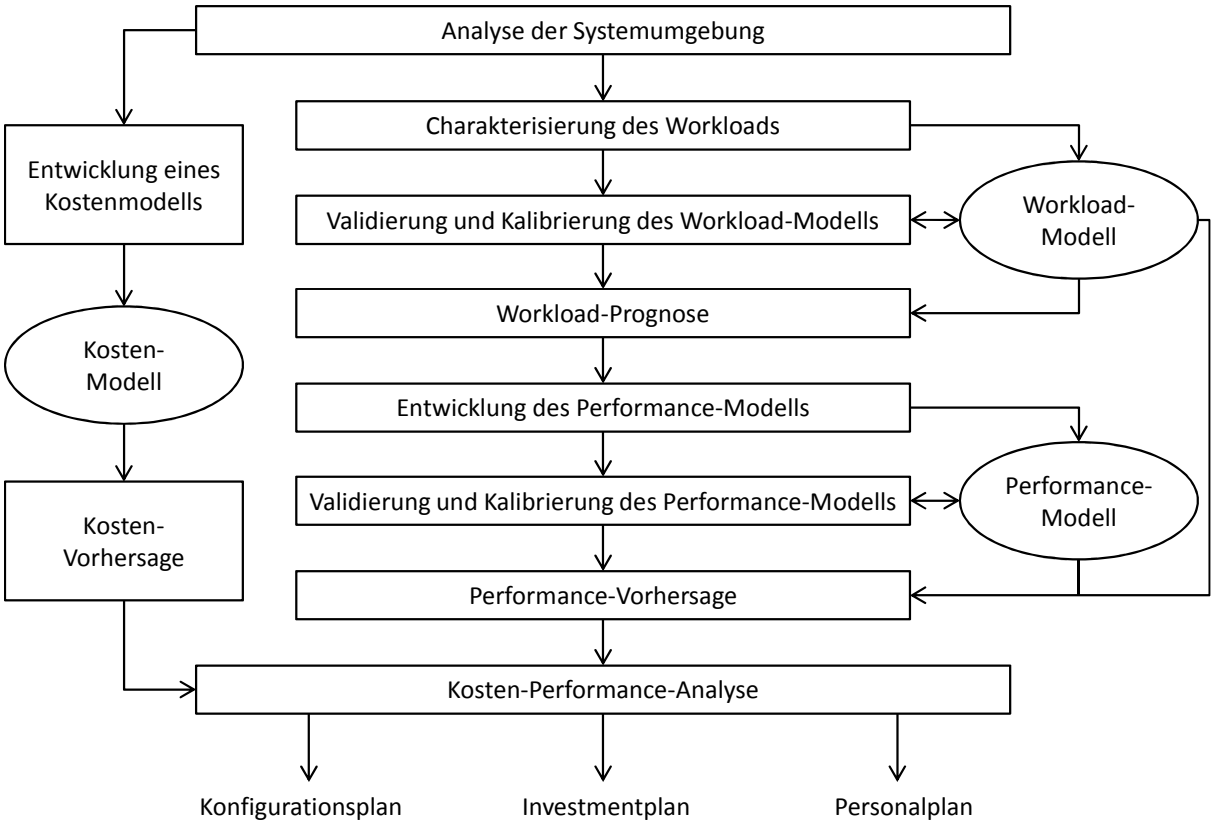


Abbildung 2-24: Kapazitätsplanungsprozess
Quelle: Menascé/Almeida (2002, 179)

Neben den Workload- und Performance-Modellen wird ein Kostenmodell entwickelt, sodass Kostenschwankungen bei der Auswahl verschiedener Systemgrößen und –architekturen dargestellt werden können. Daraus abgeleitet ergibt sich eine Kosten-Performance-Analyse, welche in den verschiedenen Szenarien von Hardware- und Softwarekonfigurationen Verwendung findet. Hieraus resultiert wiederum ein Konfigurationsplan, der notwendige Hardware- und Softwarekonfigurationen spezifiziert. Zudem werden ein Investmentplan für zukünftige Investitionen sowie ein Personalplan für evtl. erforderliche Personalzuwächse entwickelt.

Für die Konfiguration eines J2EE-Anwendungs-Servers wurde eine weitere Methodik von Raghavachari/Reimer/Johnson (2003) vorgeschlagen. Hierbei wurden Anwendungscharakteristiken und Konfigurationswerte während einer willkürlichen Untersuchung der Konfigurationsdaten untersucht. Es wurde allerdings keine genaue Beschreibung des Untersuchungsprozesses gegeben. Zur Optimierung der Konfiguration einer J2EE-Anwendung haben Kounev/Weis/Buchmann (2004) verschiedene J2EE-Konfigurationen entwickelt und diese mit der Standardkonfiguration verglichen.

2.8 Zusammenfassung

In diesem Kapitel wurden die theoretischen Grundlagen der in dieser Arbeit durchgeführten Performance-Modellierung und Simulation eines SAP-Netweaver-Portal-Systems zusammenfassend dargestellt. Dabei wurde zu Beginn das zu untersuchende System in die Domäne ERP-Systeme und im Speziellen in den Bereich der Unternehmensportale eingegliedert. Als architekturale Basis dient das von SAP entwickelte Netweaver-Framework, welches in technologischer Hinsicht aus einem ABAP- und einem Java-Stack besteht. Das Netweaver-Portal

ist dem Java-Stack zuzuordnen und verwendet als Benutzerschnittstelle einen herkömmlichen Web-Browser.

Daraufhin wurde ein Einblick in die Messtheorie gegeben, die die Grundlage für die in dieser Arbeit durchgeführten Messungen darstellt. Dabei wurden im Speziellen die grundlegenden statistischen Kennzahlen dargestellt sowie die Analyse von und der Umgang mit potentiellen Messfehlern beschrieben. Es konnte gezeigt werden, dass eine Plausibilitätskontrolle der Messwerte einer automatisierten Elimination von Ausreißern vorzuziehen ist, da neben tatsächlichen Messfehlern auch architekturbedingte Einflüsse zu untypischen Werten führen können.

Im Abschnitt zur Performance-Evaluation von Rechnersystemen wurde die Gleichwertigkeit von System, Performance-Metriken und Workload in Bezug auf die Auswahl der Performance-Evaluations-Methode dargestellt und im Anschluss die verschiedenen Ansätze vorgestellt.

Die in dieser Arbeit eingesetzten LQNs basieren auf den Warteschlangennetzen. Hierbei wurden die verschiedenen Elemente vorgestellt, die bei der Modellierung in Kapitel 4 Verwendung finden, sowie der eingesetzte Simulator beschrieben.

Die Abschnitte zu den Benchmarks und der Kapazitätsplanung, die den Abschluss dieses Kapitels bilden, konnten einen Einblick in die Bereiche der vergleichenden Leistungsanalyse sowie der Frage, wann die Systemauslastung gesättigt ist, geben. Die Kapazitätsplanung erweitert die technische Leistungsanalyse um den Kostenaspekt zu einer Kosten-Performance-Analyse und trägt somit der ökonomischen Betrachtung des Leistungsverhaltens Rechnung.

3 Systemarchitektur und Monitoring

In diesem Kapitel wird zunächst die Architektur des SAP-Netweaver-Portal-Systems dargestellt. Dabei wird der Abarbeitungsablauf von Benutzeranfragen, die am System ankommen, erläutert und auf verschiedene Komponenten eingegangen, die einen Einfluss auf die Gesamtperformance des Systems nehmen und im Warteschlangenmodell modelliert werden. Danach werden die für die Leistungsanalysen eingesetzten Monitoring-Werkzeuge, sowohl des SAP-Netweaver-AS-Java als auch auf Betriebssystemebene, vorgestellt.

3.1 System-Architektur des SAP-Netweaver-AS-Java

Wie bereits in Kapitel 2.1.1 erwähnt, basiert das SAP-Netweaver-Portal auf den SAP-Netweaver-AS-Java (kurz AS-Java), dem J2EE-Applikations-Server von SAP. Die Gesamtheit der einzelnen Java-Komponenten eines SAP-Systems wird auch als Java-Cluster bezeichnet. Die einzelnen Komponenten sind:

- Eine zentrale Java-Instanz mit einem Dispatcher und mindestens einem Java-Server-Prozess. Der Java-Dispatcher nimmt dabei die Benutzeranfragen entgegen und verteilt sie anhand eines Round-Robin-Algorithmus. Optional können auch mehrere Java-Server konfiguriert werden.

- Die Central-Services, die den Message-Server und den Enqueue-Server beinhalten. Der Message-Server verwaltet die Liste der vorhandenen Dispatcher und Server-Prozesse innerhalb eines Java-Clusters. Er stellt den Dispatchern die Lastverteilungsinformationen zur Verfügung und stellt zudem eine Kommunikationsschnittstelle zwischen den Knoten des Java-Clusters dar. Der Enqueue-Server verwaltet logische Sperren von Datenbankbereichen im Hauptspeicher, die von einem Java-Server-Prozess bei der Abarbeitung einer Applikation gesetzt werden. Zudem wird er von dem SAP-System zur internen Synchronisation verwendet.

- Eine Datenbank für die zentrale Datenhaltung.

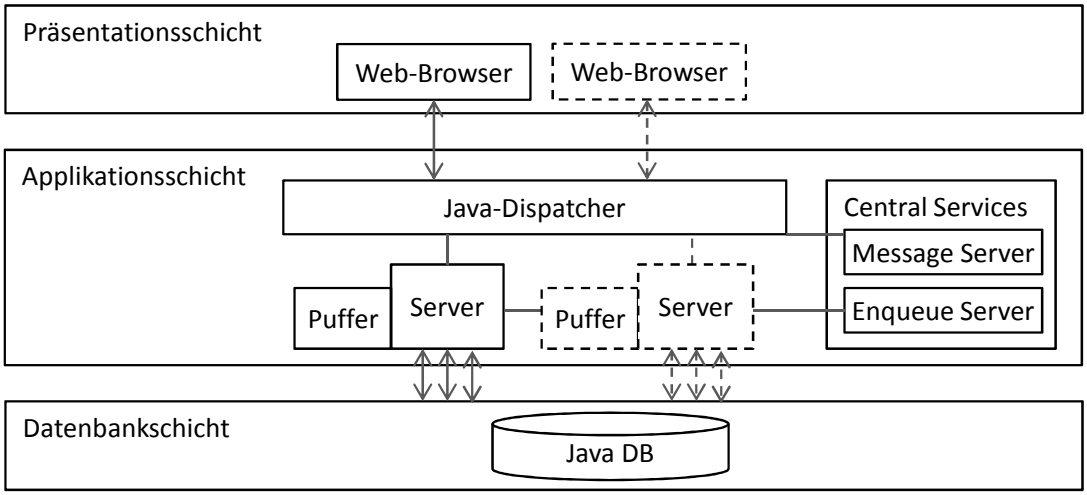


Abbildung 3-1: Aufbau eines Java-Clusters
Quelle: eigene Darstellung

Wie in Abbildung 3-1 dargestellt, wird als Benutzerinterface standardmäßig ein normaler Web-Browser verwendet. Die eingehenden Anfragen werden vom Java-Dispatcher an den Server-Prozess weitergeleitet, wo die eigentliche Abarbeitung der Anfrage stattfindet. Im Gegensatz zum SAP-Netweaver-AS-ABAP (kurz AS-ABAP) sind die Cluster-Knoten des AS-Java nebenläufig. Somit können von einem Java-Server-Prozess mehrere Benutzeranfragen gleichzeitig abgearbeitet werden.

Während der Abarbeitung der Benutzeranfragen ist es oft notwendig, auf die Daten im entsprechenden Schema der Datenbank zuzugreifen. Daher ist jeder Java-Server-Prozess mehrfach (über den Datenbank-Connection-Pool) mit dem Datenbankschema verbunden. Puffer beschleunigen dabei die Abarbeitung der Benutzeranfragen, da die in dem Puffer zwischengespeicherten Daten einen Zugriff auf die Datenbank vermeiden. Jeder Java-Server-Prozess verwaltet seinen eigenen Puffer.

3.1.1 Abarbeitung von Anfragen

Einkommende Benutzeranfragen werden von „Server Socket Listeners“ im Java-Dispatcher angenommen. Sollten mehr Anfragen ankommen als weitergeleitet werden können, werden sie zuerst in einer Socket-Warteschlange festgehalten. Sollte auch diese voll sein (die Warteschlange hat standardmäßig eine Größe von 200 Anfragen), wird eine Fehlermeldung zurückgegeben. Vor allem bei Lasttests, die im oberen Lastbereich arbeiten, kann dies immer wieder vorkommen und wird von entsprechenden Log-Dateien festgehalten.

Wird die Anfrage akzeptiert, wird sie an den „Connections Manipulator“ weitergeleitet. Dieser führt eine protokollspezifische Liste der eingehenden Anfragen. Entsprechende Monitore (siehe Tabelle 3-1) geben Auskunft über die Anzahl an Verbindungen:

Monitor	Beschreibung
CurrentPoolSize	Anzahl der Verbindungen im Verbindungspool.
HTTPConnectionsCount	Anzahl aktueller http-Verbindungen.

Tabelle 3-1: Wichtige Monitore des Connections-Manipulator
Quelle: eigene Darstellung in Anlehnung an SAP (2011a)

Jede Anfrage im Verbindungs-Pool wird einem System-Thread im System-Thread-Manager zugeordnet. Der System-Thread-Manager hat standardmäßig eine Kapazität von 100 System-Threads, allerdings dienen die einzelnen System-Threads nicht nur Benutzeranfragen, sondern auch der internen Kommunikation. Die *WaitingTasksQueue* beinhaltet Anfragen, die nicht sofort einem System-Thread zugeordnet werden konnten. Eine Auswahl wichtiger Monitore des System-Thread-Managers ist in Tabelle 3-2 aufgelistet.

Monitor	Beschreibung
CurrentThreadPoolSize	Derzeitige Größe des Thread-Pools. Die Größe wird dynamisch an den momentanen Bedarf angepasst.
InitialThreadPoolSize	Initiale Pool-Größe.
MinimumThreadPoolSize	Minimale Pool-Größe.
MaximumThreadPoolSize	Maximale Pool-Größe.
WaitingTasksCount	Anzahl der in der *WaitingTasksQueue* vorhandenen Aufga-

	ben.
WaitingTasksQueueOverflow	Anzahl der Aufgaben, die bei einer vollen *WaitingTasks-Queue* keinen Platz mehr gefunden haben.
WaitingTasksQueueSize	Derzeitige Größe der *WaitingTasksQueue*. Die Größe wird auch hier dynamisch an den momentanen Bedarf angepasst.
ActiveThreadsCount	Anzahl der derzeit aktiven Threads im System.

Tabelle 3-2: Wichtige Monitore des System-Thread-Managers
Quelle: eigene Darstellung in Anlehnung an SAP (2011a)

Speziell für Leistungsanalysen, die eine hohe Anzahl an Anfragen an den Dispatcher senden, sind die beiden Monitore *ActiveThreadsCount* und *MaximumThreadPoolSize* sowie die Anzeige der *WaitingTasksQueueOverflow* von Bedeutung. Sollte bei letzterer ein Wert größer Null gemeldet werden, wurden Anfragen nicht korrekt bearbeitet und ein Fehler zurückgegeben.

Der HTTP-Provider-Service legt die Daten der Anfrage in die Session-Queue ab und gibt anschließend den System-Thread wieder frei. Zudem wird in diesem Service entschieden, an welchen Java-Server-Prozess die Anfrage weitergeleitet wird und die Verbindung zum Client aufrechterhalten. Sollte das Zeitintervall, welches über den Wert *KeepaliveTimeout* definiert wird, überschritten werden, wird die Verbindung zum Client abgebrochen. Der HTTP-Provider-Service stellt somit eine der Kernkomponenten des Java-Dispatchers dar, da nicht nur die Annahme der Anfragen von außen erfolgt, sondern auch die Weiterleitung an einen Java-Server-Prozess und das Halten der Verbindung über diesen Dienst erfolgt. Tabelle 3-3 zeigt die wichtigsten Monitore des HTTP-Provider-Services.

Monitor	**Beschreibung**
ConnectionsInKeepAlive	Derzeitige Anzahl an aktiven Verbindungen.
ConnectionsClosedByClient	Anzahl der Verbindungen, die durch den Client beendet bzw. abgebrochen wurden.
ConnectionsClosedByServer	Anzahl der Verbindungen, die durch den Server geschlossen wurden. Dies kann sowohl durch das Überschreiten des *KeepaliveTimeout*-Parameters als auch über einen entsprechenden Funktionsaufruf erfolgt sein.
AllRequestsCount	Anzahl der insgesamt angekommenen Anfragen seit die Instanz gestartet wurde.
ConnectionsReadingHeaders	Anzahl der Anfragen, die gelesen wurden, um sie einem Java-Server-Prozess zuzuordnen.
ConnectionsReadingResponse	Anzahl der derzeit verarbeiteten Anfragen, deren Antwort vom Java-Server-Prozess noch aussteht, um sie an den Client zurückzuschicken.

Tabelle 3-3: Wichtige Monitore des http-Provider-Services im Dispatcher
Quelle: eigene Darstellung in Anlehnung an SAP (2011a)

Die zuvor erwähnte Session-Queue, in der die Anfrage abgelegt wird, ist Bestandteil des Cluster-Managements. Dieser Manager ist für die Steuerung der Kommunikation zwischen Java-Dispatcher und Java-Server-Prozess verantwortlich. Auch hier existieren verschiedene Monitore, die einen Status über den derzeitigen Systemstand wiedergeben (siehe Tabelle 3-4).

Monitor	Beschreibung
TotalSessionBytesReceived	Summe der in dieser Sitzung erhaltenen Bytes.
TotalSessionBytesSent	Summe der in dieser Sitzung gesendeten Bytes.
CurrrentSessionQueueSize	Derzeitige Anzahl an Sessions in der Session-Queue.
MaxSessionQueueSize	Die Größe der Session-Queue. Standardmäßig hat sie eine größe von 256.
AverageSessionProcessTime	Hier wird die durchschnittliche Verweilzeit in der Warteschlange angegeben.

Tabelle 3-4: Wichtige Monitore des Cluster-Managers
Quelle: eigene Darstellung in Anlehnung an SAP (2011a)

Der gesamte Verlauf der Anfrage-Abarbeitung im Java-Dispatcher ist zusammenfassend in Abbildung 3-2 grafisch dargestellt:

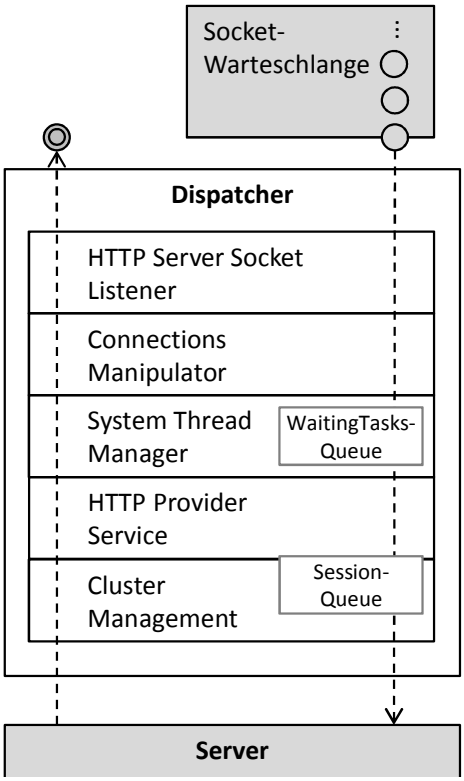


Abbildung 3-2: Request-Abarbeitung im Dispatcher
Quelle: eigene Darstellung in Anlehnung an Tschense (2004, 22)

Nachdem die Anfrage von dem Cluster-Manager des Java-Dispatchers an den Java-Server übergeben wurde, wird sie dort von dem Gegenstück, dem Server-Cluster-Management, entgegengenommen. Die Monitore des serverseitigen Cluster-Managers entsprechen denen des Dispatcher-Cluster-Managers.

Ebenfalls identisch mit dem dispatcherseitigen Server-Thread-Manager sind die Monitore des (serverseitigen) System-Thread-Managers, der die Anfrage von dem Cluster-Manager erhält. Auch hier ist die standardmäßige maximale Anzahl an simultanen Threads 100, wobei nicht alle den Benutzeranfragen zur Verfügung stehen, da auch interne Kommunikations-Threads damit verwaltet werden.

Der HTTP-Provider-Service des Servers nimmt die Anfragen vom Server-Thread-Manager entgegen und zählt die eingehenden Anfragen seit Serverstart. Zudem werden Response-Codes zurückgegeben, die Aufschluss darüber geben, welchen Status die einzelnen Anfragen besitzen. In Tabelle 3-5 sind die wichtigsten Monitore des serverseitigen HTTP-Provider-Services aufgelistet.

Monitor	Beschreibung
AllRequestsCount	Summe der Anfragen seit dem Start des Servers.
ResponsesFromCacheCount	Summe der Anfragen, deren Antworten aus dem Cache erfolgt sind.
2xxResponsesCount	Summe der Anfragen, die einen Response-Code *successful* erhalten haben.
3xxResponsesCount	Summe der Anfragen, die einen Response-Code *redirection* erhalten haben.
4xxResponsesCount, 5xxResponsesCount	Summe der Anfragen, die einen Response-Code *error* erhalten haben.
POST	Summe der Anfragen, die über POST gesendet wurden.
GET	Summe der Anfragen, die über GET gesendet wurden.

Tabelle 3-5: Wichtige Monitore des http-Provider-Services im Server
Quelle: eigene Darstellung in Anlehnung an SAP (2011a)

Im Anschluss wird die Anfrage an einen Applikations-Thread übergeben. Ähnlich wie der System-Thread-Manager besitzt auch der Application-Thread-Manager einen Pool an Threads sowie eine Warteschlange (*WaitingTasksQueue*). Der Unterschied zu den System-Threads liegt darin, dass hier nur die Benutzeranfragen verwaltet werden. Daher ist auch die Größe des Pools kleiner und standardmäßig auf 40 simultane Threads angelegt. Die Größe ist zwar konfigurierbar, sollte aber im Regelfall nicht höher gesetzt werden. Vielmehr sollte bei Bedarf die Konfiguration eines weiteren Java-Server-Prozesses vorgezogen werden (Tschense 2004, 26f.). Die wichtigsten Monitore des Application-Thread-Managers sind in Tabelle 3-6 dargestellt.

Monitor	Beschreibung
CurrentThreadPoolSize	Derzeitige Größe des Thread-Pools. Die Größe wird dynamisch an den momentanen Bedarf angepasst.
InitialThreadPoolSize	Initiale Pool-Größe des Application-Thread-Pools.
MinimumThreadPoolSize	Minimale Pool-Größe des Application-Thread-Pools.
MaximumThreadPoolSize	Maximale Pool-Größe des Application-Thread-Pools.
WaitingTasksCount	Anzahl der in der *WaitingTasksQueue* vorhandenen Benutzeranfragen.
WaitingTasksQueueOverflow	Anzahl der Aufgaben, die bei einer vollen *WaitingTasksQueue* keinen Platz mehr gefunden haben.
WaitingTasksQueueSize	Derzeitige Größe der *WaitingTasksQueue*. Die Größe wird auch hier dynamisch an den momentanen Bedarf angepasst.
ActiveThreadsCount	Anzahl der derzeit aktiven Applikatons-Threads im System.

Tabelle 3-6: Wichtige Monitore des Application-Thread-Managers
Quelle: eigene Darstellung in Anlehnung an SAP (2011a)

Je nach Anfragetyp wird der Thread an den Web-Container oder Enterprise-Java-Beans-Container (EJB-Container) weitergeleitet. In diesem Container wird somit die eigentliche Abarbeitung der Anfrage durchgeführt. Der Web-Container wird dabei der Präsentationsschicht zugeordnet, da er Servlets und Java-Server-Pages enthält, wo hingegen der EJB-Container der Applikationsschicht angehört und Session- sowie Entity-Beans enthält.

Sollten während der Bearbeitung der Aufgabe (sprich bei der Abarbeitung des Programms) Datenbankzugriffe erforderlich sein, werden diese dem JDBC-Connector-Service übergeben. Der Connection-Pool ist dabei standardmäßig auf 10 simultane Verbindungen begrenzt. Nachdem das Programm abgearbeitet wurde, wird das Ergebnis über das Cluster-Management wieder zurück an den Dispatcher gesendet, der die Antwort wiederum an den Client weitergibt. Die Abarbeitung einer Anfrage in einem Java-Server ist zusammenfassend in Abbildung 3-3 grafisch dargestellt.

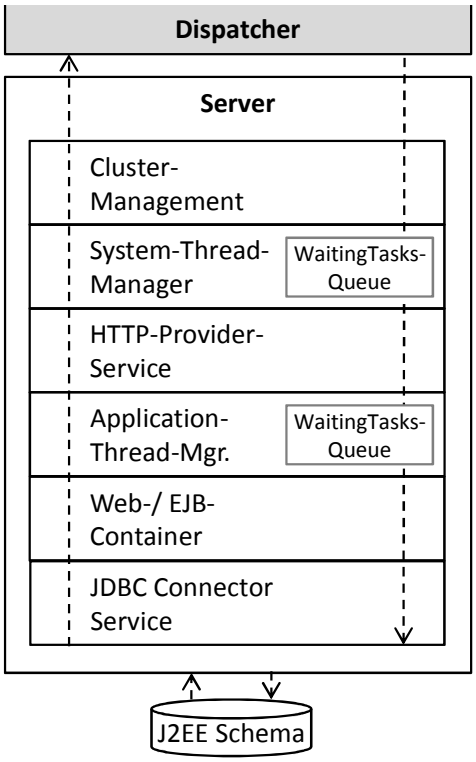


Abbildung 3-3: Request-Abarbeitung im Java-Server
Quelle: eigene Darstellung in Anlehnung an Tschense (2004, 25)

3.1.2 Transaktionsverwaltung

Ähnlich zum AS-ABAP wird auch im AS-Java das ACID-Prinzip („Atomicity, Consistency, Isolation, Durability“) (Haerder/Reuter 1983; Kemper/Eickler 2011) bei der transaktionsorientierten Verarbeitung der Aufgaben verfolgt. Der Java-Transaction-Service ist dabei verantwortlich für die Verwaltung der durchgeführten Transaktionen im AS-Java. In diesem Service wurden die zwei J2EE-Standards Java-Transaction-API (JTA) und Java-Transaction-Service (JTS) implementiert.

Im AS-Java werden durchgeführte Änderungen, die der Benutzer über den Web-Browser in das Benutzer-Interface einträgt, nicht sofort persistent in der Datenbank gespeichert. Erst

wenn der Benutzer seine Eingaben speichert, wird die Java-Transaktion abgeschlossen und über eine Datenbanktransaktion gespeichert. Abbildung 3-4 illustriert diesen Sachverhalt.

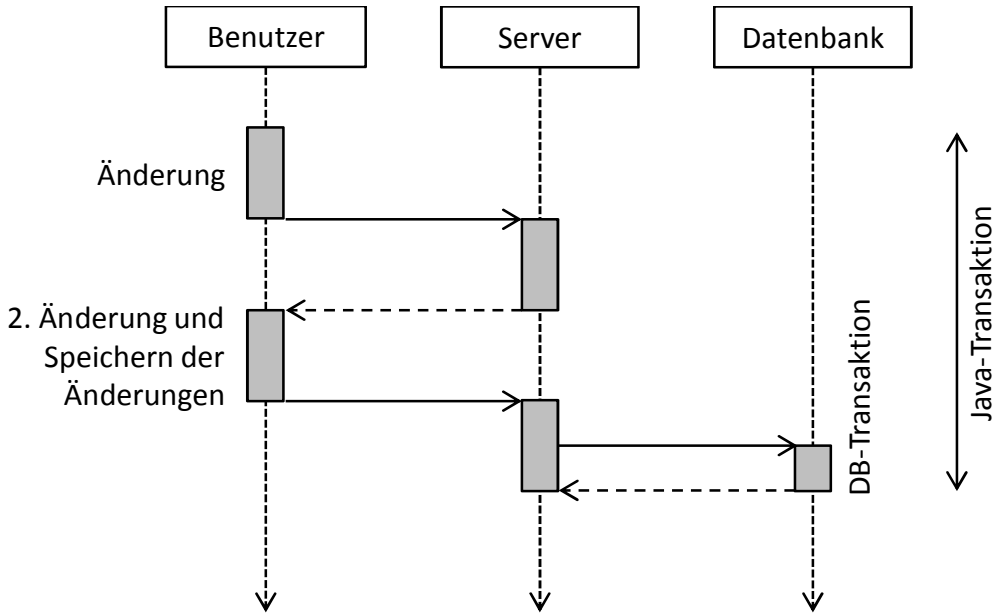


Abbildung 3-4: Java-Transaktionen
Quelle: eigene Darstellung

3.1.3 Sperrtabellenverwaltung

Die in einer Java-Transaktion durchgeführten Datenbankaufrufe können Sperren erfordern, die über die Sperrtabelle des Enqueue-Servers verwaltet werden. Da Sperreinträge eines Programmes die Abarbeitung anderer Aufgaben verzögern und somit die Performance stark beeinflussen können, wird die Sperrverwaltung in diesem Kapitel dargelegt.

Die Sperrverwaltung ist grundsätzlich Aufgabe des J2EE-Frameworks und dementsprechend im J2EE-Standard spezifiziert. Das Verhalten ist dabei jedoch abhängig von der verwendeten Datenbankplattform. Der Enqueue-Server wurde daher analog zum ABAP-Stack eingeführt, um ein einheitliches Sperrmanagement zu schaffen, welches datenbankunabhängig ist.

Um einen Eintrag in der Sperrtabelle hinzuzufügen, spricht das Java-Programm das entsprechende Interface des Application-Locking-Service an. Der Application-Locking-Service bietet dabei verschiedene Möglichkeiten, auf welche Art und Weise die Sperren gesetzt werden (SAP 2011c). Neben der klassischen Tabellensperre, bei der eine komplette Datenbanktabelle oder ein Tabellenbereich gesperrt wird, existieren logische und serverinterne Sperren, die jedoch eine untergeordnete Rolle im Zuge der Leistungsbetrachtung spielen und daher in dieser Arbeit nicht weiter betrachtet werden.

Die Sperranfrage wird, sobald sie von dem Application-Locking-Service entgegengenommen wurde, an den Enqueue-Server weitergeleitet, der zuerst überprüft, ob bereits eine Sperre für das angefragte Objekt besteht und setzt danach gegebenenfalls die gewünschte Sperre in der Sperrtabelle.

Die Sperrtabelle liegt im „Shared Memory“ des Hauptspeichers. Ein Eintrag weist dabei die in Abbildung 3-5 dargestellte Struktur auf.

Eigen-tümer_1 - ID - Zähler	Eigen-tümer_2 - ID - Zähler	Backup-ID - ID - Flag	Elementarsperre		
			Sperr-modus - X,E,S,O	Name - Name der Tabelle	Argument - Sperr-argumente

Abbildung 3-5: Struktur eines Sperrtabellen-Eintrags
Quelle: eigene Darstellung in Anlehnung an SAP (2011k)

Das Eigentümer-Feld beinhaltet eine eindeutige ID und einen Zähler, der angibt, wie oft diese Sperre bereits durch diesen Eigentümer gesetzt wurde. Das Backup-Feld entscheidet, ob die Sperre gespeichert wird und somit auch nach einem Neustart des Enqueue-Servers vorhanden ist. Die nächsten Felder geben an, welcher Bereich gesperrt werden soll und in welcher Art und Weise. Dabei gibt der Name die Tabelle an, bei der bestimmte Felder gesperrt werden sollen. Zudem können zusätzliche Sperrargumente eingetragen werden. Der Sperrmodus entscheidet über die Art der Sperre. Die verschiedenen Modi sind in Tabelle 3-7 angegeben.

Typ	**Beschreibung**
S *(Shared lock)* - Lesesperre	Es können mehrere Lesesperren auf ein und dasselbe Feld zeigen, allerdings blockieren sie Schreibsperren-Anfragen. Daher kann auch eine Lesesperre Einfluss auf die Performance nehmen, da Schreibvorgänge durch Lesesperren verzögert werden.
E *(Exclusive lock)* - Schreibsperre	Schreibsperren sind exklusiv. Allerdings können kumulative Sperren gesetzt werden. Kumulative Sperren sind dann möglich, wenn die Elementarsperre (Name, Argument, Sperrtyp) identisch ist und kein Sperreintrag des Typs X für das betreffende Feld vorliegt.
X *(eXclusive but not cumulative lock)* – Exklusive Schreibsperre	Schreibsperren, bei denen auch kumulative Sperren untersagt sind.
O *(Optimistic lock)* - Optimistische Schreibsperre	Optimistische Sperren sind vorerst Lesesperren. Erst sobald eine tatsächliche Änderung der Daten erfolgt, wird die optimistische Sperre in eine exklusive umgewandelt.

Tabelle 3-7: Sperrmodi
Quelle: eigene Darstellung in Anlehnung an SAP (2011k)

Für die Anzeige, die Bearbeitung und das Aufzeichnen von Sperrtabellen-Einträgen kann sowohl der Netweaver-Administrator als auch die SAP-Konsole (siehe Abbildung 3-6) verwendet werden.

Abbildung 3-6: Anzeige der Sperreinträge mittels SAP-Konsole
Quelle: eigene Darstellung

Zum Aktivieren bzw. Deaktivieren der Aufzeichnungen sind folgende Befehle verfügbar:

- *ENABLE_SERVER_LOGGING*: Sperreintrag-Logging aktivieren
- *DISABLE_SERVER_LOGGING*: Sperreintrag-Logging deaktivieren
- *ENABLE_LOCKINGSTAT*: Lock-Statistiken aktivieren
- *DISABLE_LOCKINGSTAT*: Lock-Statistiken deaktivieren

3.1.4 Tabellenpuffer

Wie bereits zu Beginn des Kapitels erwähnt, besitzt jeder Java-Server-Prozess seinen eigenen Tabellenpuffer, allerdings teilen sich die einzelnen Java-Server-Threads den Puffer. Nicht zu verwechseln ist der Tabellenpuffer in einer Java-Instanz mit der Pufferung im Datenbanksystem (siehe Abbildung 3-7). Wichtig ist dieser Unterschied auch für die Performance-Modellierung, da der jeweilige Puffer unterschiedliche Ressourcen belastet. Da in dieser Arbeit die Datenbank als Black-Box betrachtet wird, wird nur auf den Tabellenpuffer der Java-Instanz eingegangen.

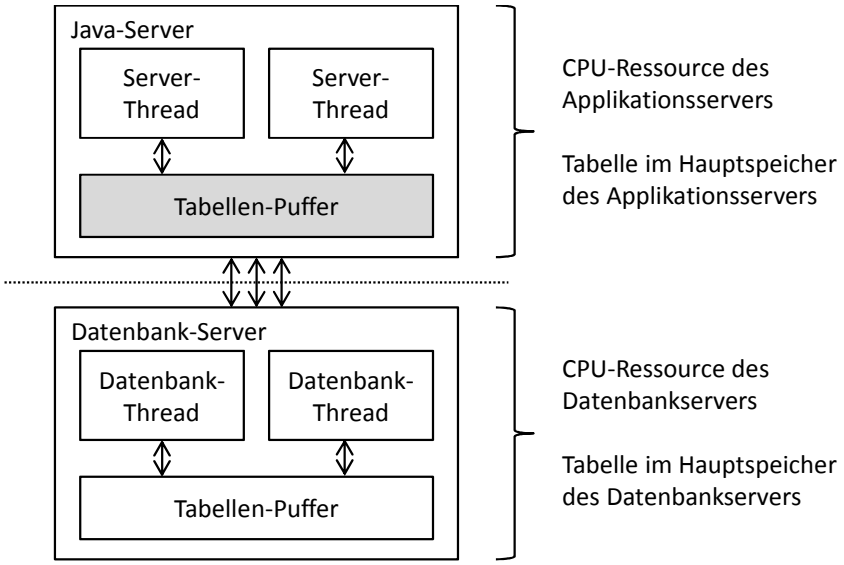


Abbildung 3-7: Tabellenpuffer im Java-Server-Prozess
Quelle: eigene Darstellung

Es werden grundsätzlich drei verschiedene Arten von Tabellenpufferung unterschieden:

- *Einzelsatzpufferung*: Einzelsatzpufferung wird dann angewendet, wenn bei einem Zugriff der gesamte Primärschlüssel (also alle Felder, die Teil des Primärschlüssels sind) verwendet wird. Jeder Datensatz wird dann gepuffert, sobald er das erste Mal gelesen wird. Sobald ein weiterer Zugriff erfolgt, kann er direkt vom Tabellenpuffer gelesen werden.

- *Vollständige Pufferung*: Bei der vollständigen Pufferung wird die gesamte Tabelle gepuffert, sobald ein Lesezugriff auf einen Datensatz der Tabelle erfolgt.

- *Generische Pufferung mit n Schlüsselfeldern*: Bei der generischen Pufferung wird abhängig von n die Anzahl der Schlüsselfelder als der zu puffernde Bereich angesehen. Mit anderen Worten werden alle Datensätze gepuffert, deren n Schlüsselfelder identisch mit den Schlüsselfeldern des abgefragten Datensatzes sind. In Abbildung 3-8 sind zwei Beispiele der generischen Pufferung abgebildet: links unten wird nur ein Schlüsselfeld ($n = 1$) ausgewertet und alle Datensätze mit demselben Wert in diesem Feld gepuffert. Bei dem zweiten Beispiel (unten rechts) werden $n = 2$ Schlüsselfelder ausgewertet, das dritte Schlüsselfeld des Primärschlüssels allerdings außer Acht gelassen. Dadurch lässt sich je nach Tabellenart und -verwendung (Größe, Art der Zugriffe, etc.) der Pufferbereich auf ein sinnvolles Maß beschränken.

Sobald ein Eintrag geändert wird, müssen die entsprechenden Einträge in den Tabellenpuffern der einzelnen Java-Instanzen aktualisiert werden. Dieser Vorgang wird als Puffersynchronisierung bezeichnet. Die Invalidierung veralteter Einträge geschieht je nach Art der Tabellenpufferung für die entsprechende Region.

Schlüssel 1	Schlüssel 2	Schlüssel 3

Vollständige Pufferung

Schlüssel 1	Schlüssel 2	Schlüssel 3
000	eins	alpha
111	zwei	beta
111	drei	gamma
111	zwei	delta
111	zwei	epsilon
222	zwei	zeta

Einzelsatzpufferung

Schlüssel 1	Schlüssel 2	Schlüssel 3
000		
111		
111		
111		
111		
222		

Generische Pufferung mit einem Schlüsselfeld

Schlüssel 1	Schlüssel 2	Schlüssel 3
000	eins	
111	zwei	
111	drei	
111	zwei	
111	zwei	
222	zwei	

Generische Pufferung mit zwei Schlüsselfeldern

Abbildung 3-8: Arten der Tabellenpufferung
Quelle: eigene Darstellung

Die einzelnen Objekte im Tabellenpuffer können über den Netweaver-Administrator eingesehen (siehe Abbildung 3-9) oder über die Aktivierung der Puffer-Traces aufgezeichnet werden.

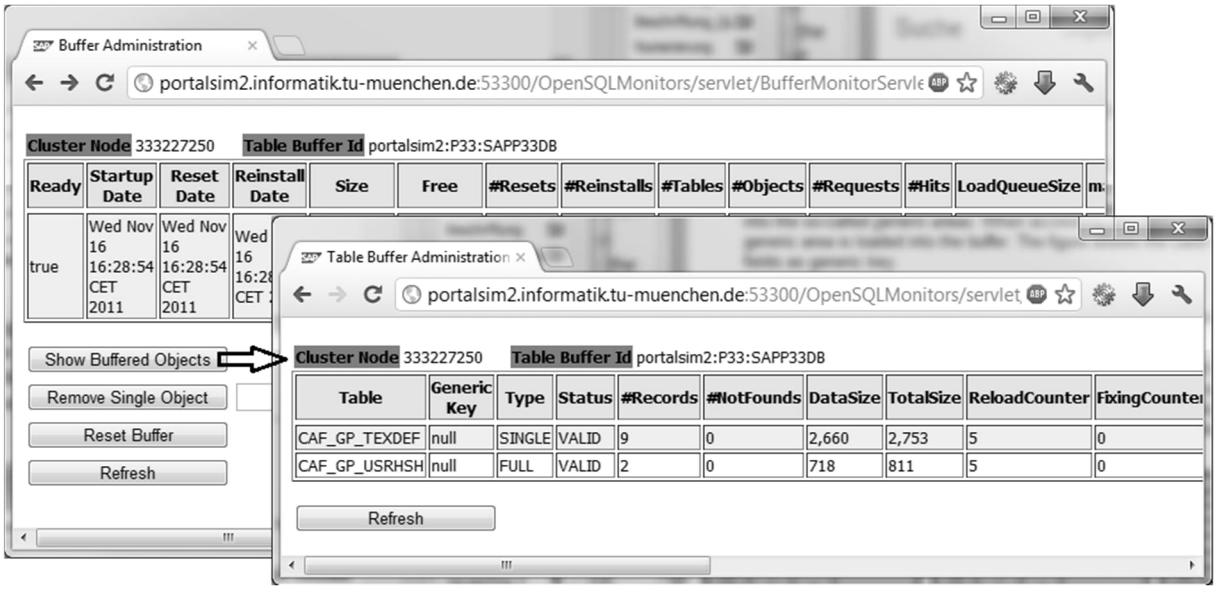


Abbildung 3-9: Anzeige der Tabellen-Puffer
Quelle: eigene Darstellung

Die wichtigsten zur Verfügung stehenden Daten der einzelnen Einträge sind in Tabelle 3-8 aufgelistet.

Feld	Beschreibung
Puffer-Details	
Size	Größe des Tabellenpuffers
#Displacements	Anzahl an Verdrängungen
#Requests	Anzahl der Anfragen
#Hits	Anzahl der Treffer
DisplData	Verdrängte Datenobjekte der letzten Verdrängung
DisplObjects	Verdrängte Pufferobjekte der letzten Verdrängung
DisplTables	Verdrängte Tabellen der letzten Verdrängung
TraceOn	Flag, ob Buffer-Trace aktiviert ist
Objekt-Details	
Table	Name der gepufferten Tabelle
Status	Status (valide, invalide)
Type	Art der Pufferung (voll, generisch, Einzelsatz)
Generic Key	Anzahl der Schlüsselfelder (bei generischer Pufferung)
DataSize	Größe der Puffereinträge in Bytes
TotalSize	Größe der gesamten Tabelle im Tabellenpuffer in Bytes
Used	Anzahl der Zugriffe
Inval	Anzahl der durchgeführten Invalidierungen

Tabelle 3-8: Auswahl relevanter Daten im Tabellenpuffer-Monitor
Quelle: eigene Darstellung

Für den Anwendungsentwickler ist es möglich, den Puffer zu umgehen, indem im SQL-Statement die Angabe

/@ SAP BYPASSING BUFFER */*

nach dem Select-Statement angegeben wird. Für die Workload-Modellierung ist es wichtig zu sehen, ob die Datenbankoperationen eines Lastschrittes den Puffer verwenden oder nicht. Daher müssen die SQL-Statements des SQL-Traces (siehe Kapitel 3.2.4) ausgewertet werden und Datenbankoperationen, die explizit keinen Puffer verwenden, entsprechend modelliert werden.

Eine gute Pufferkonfiguration ist Voraussetzung für ein leistungsfähiges Portalsystem. Auf der einen Seite sollte genügend Platz zur Verfügung gestellt werden, um Verdrängungen zu vermeiden, auf der anderen Seite sollte unnötig bereitgestellter Platz vermieden werden. Da man jedoch die zukünftig angestoßenen Portaloperationen nur abschätzen kann, lässt sich der benötigten Platz nicht genau vorhersagen. Die Anpassungen der Konfiguration, die man zur Leistungsverbesserung von Tabellenpuffern durchführt, sind somit ein iterativer Prozess.

3.1.5 Garbage-Collector

Die folgende Beschreibung des Speichermanagements richtet sich nach der Funktionsweise der „Java Virtual Machine“ (JVM) von IBM (2011l), da die in dieser Arbeit verwendeten

Systeme für die Leistungsanalysen auf einer IBM-Infrastruktur basieren. Die JVMs von SUN und SAP weisen jedoch ähnliche Funktionsweisen auf. Zudem wird speziell auf die von SAP für ERP-Systeme und Portale empfohlene Garbage-Collector-Konfiguration eingegangen.

Grundsätzlich kann einer J2EE-Engine nicht mehr Speicher zur Verfügung gestellt werden, als auf dem System freier physikalischer Speicher vorhanden ist. Der Speicherbereich, in dem die Objekte eingelagert werden, wird als Heap bezeichnet. Dieser wird in einen Young-Generation-Bereich (auch Nursery-Generation genannt) und einen Old-Generation-Bereich (auch Tenured-Generation genannt) unterteilt (siehe Abbildung 3-10).

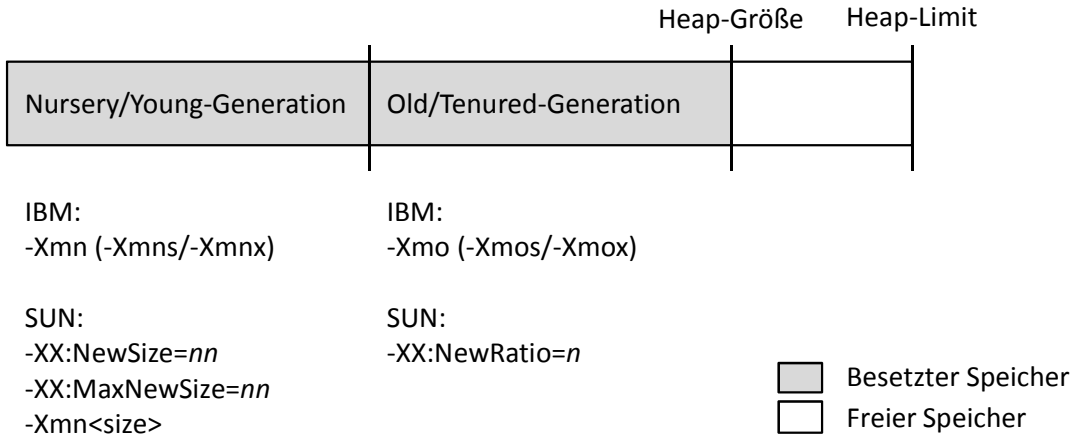


Abbildung 3-10: Aufbau des Heap-Speichers
Quelle: eigene Darstellung

Sobald kein Speicher mehr allokiert werden kann, führt ein Allokationsfehler zu einem Garbage-Collector-Lauf. Der Garbage-Collector (GC) identifiziert nicht mehr benötigte Objekte im Heap-Speicher (die als „Garbage“ bezeichnet werden) und gibt sie frei, sodass sie der JVM wieder zur Verfügung stehen. Grundsätzlich führt dies zu einem Ausführungsstopp der laufenden Java-Threads und der Garbage-Collector erlangt die exklusive Kontrolle über die virtuelle Maschine. Dabei durchläuft der Garbage-Collector folgende Phasen:

- *Mark*: Während der Markierungsphase werden alle Objekte, die noch eine Referenz aufweisen markiert, sodass der Rest in die Kategorie „Garbage“ fallen muss.

- *Sweep*: Während der Sweep-Phase werden die Speicherbereiche identifiziert, die nicht mehr referenziert werden, und zur erneuten Verwendung zur Verfügung gestellt. Speicherbereiche, die kleiner als 512 Byte sind, werden nicht in die Liste aufgenommen, sondern erst dann wiederverwendet, wenn benachbarte Objekte frei werden.

- *Compact*: Die Compaction-Phase ist optional und kann eingesetzt werden, um den Speicherbereich zu defragmentieren.

Da SAP das Generationenmodell (wie in Abbildung 3-10 dargestellt) im Heap-Speicher der JVM empfiehlt, wird bei einer Portalinstallation die IBM-JVM-Option „Generational and Concurrent GC“ aktiviert. Dieser GC-Modus ist für kurzlebige Objekte optimiert (welches typisch für transaktionsorientierte Applikationen ist) und reduziert die Phase, in der der Garbage-Collector exklusiven Zugriff auf die JVM erhält.

Die Grundannahme dieser Methode ist, dass viele Objekte sehr früh zu „Garbage“ werden und somit der Fokus auf die junge Generation gelegt wird. Dazu wird, wie bereits dargestellt, der Speicherbereich in einen Young-Generation- und einen Tenured-Generation-Bereich aufgeteilt. Der Young-Generation-Bereich wird wiederum in zwei logische Einheiten aufgeteilt: den Allocate-Bereich und den Survivor-Bereich. Sobald der Allocate-Bereich voll ist, wird ein GC-Lauf (auch GC-Zyklus genannt) gestartet. Während des sogenannten Scavenge-Vorgangs werden alle erreichbaren Objekte abhängig von ihrem Alter in den Tenured-Bereich oder den Survivor-Bereich kopiert. Objekte, die nicht erreichbar sind, bleiben unberührt. Sobald die Kopiervorgänge abgeschlossen sind, werden die Rollen der beiden Bereiche (Allocate und Survivor) vertauscht, sodass der ehemalige Allocate-Bereich, der inzwischen keine erreichbaren Objekte mehr aufweist, als Survivor-Bereich für den nächsten Scavenge-Vorgang zur Verfügung steht. Dieser Vorgang ist in Abbildung 3-11 grafisch dargestellt.

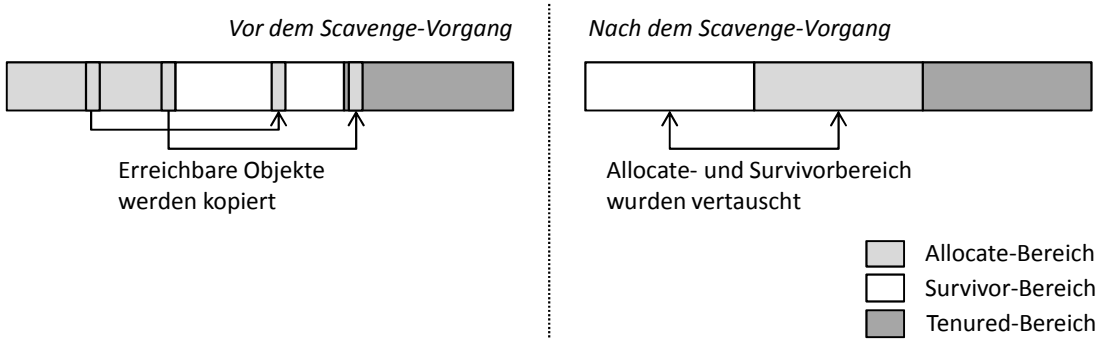


Abbildung 3-11: Scavenge-Vorgang
Quelle: eigene Darstellung

In der Tenured-Generation findet sich zudem ein spezieller Bereich für die Allokation von großen Objekten (>64KB), die sogenannte Large-Object-Area (LOA). Dieser Bereich wird immer dann verwendet, wenn aufgrund von Fragmentierung in der Small-Object-Area (SOA) kein Speicherbereich groß genug für das aufzunehmende Objekt ist. Mit der Reservierung eines zusammenhängenden Speicherbereichs für große Objekte wird versucht, die Anzahl der Compaction-Phasen (die den SOA Bereich defragmentieren und so zu größeren, zusammenhängenden, freien Speicherbereichen führen würden) zu reduzieren und somit einen Performance-Gewinn zu erzielen.

Die GC-Aktivitäten können in eine Log-Datei geschrieben werden, damit die Leistungsdaten später analysiert und ausgewertet werden können. Dazu muss der Schalter *–verbose:gc* als Parameter übergeben werden. Zudem können folgende Optionen angegeben werden:

- *[DIR_PATH]*: das Verzeichnis, in dem die Log-Datei geschrieben werden soll.
- *[FILE_NAME]*: der Dateiname der Log-Datei.
- *X*: Anzahl der vorgehaltenen Dateien.
- *Y*: Anzahl der GC-Läufe, die pro Datei festgehalten werden.

Abbildung 3-12 zeigt einen beispielhaften GC-Zyklus und die dabei aufgezeichneten Performance-Daten.

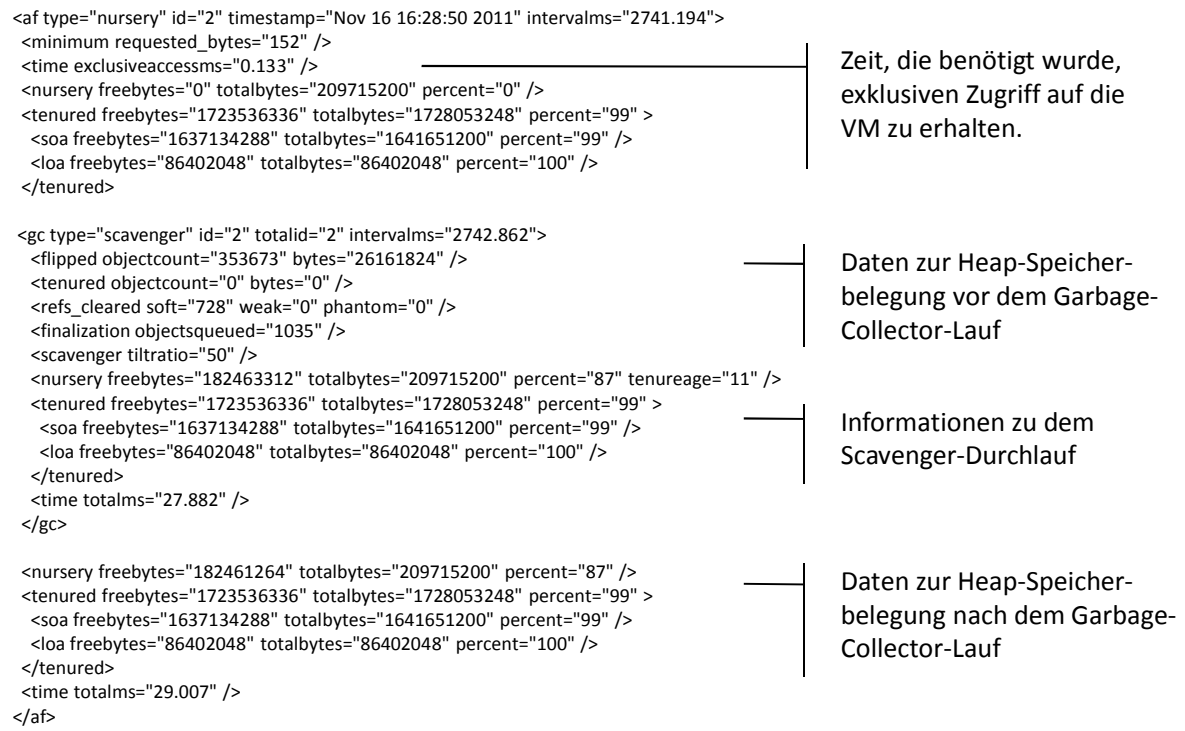


Abbildung 3-12: Beispiel eines Garbage-Collector-Laufs in der Logdatei
Quelle: eigene Darstellung

Betrachtet man die Speicherverwaltung aus der Sicht der Performance-Analyse einer Java-Applikation, unabhängig von dem Workload und der Laufzeitkonfiguration, können drei Performance-Indikatoren unterschieden werden (Cheng 2008):

- *Framework-Space*: Der Framework-Space benennt die Größe des freien Speichers, der der JVM nach dem Starten für Applikationen zur Verfügung steht.
- *User-Session-Space*: Der User-Session-Space gibt die Menge an Speicher an, die von einem eingeloggten, inaktiven Benutzer allokiert wurde und nicht von anderen Benutzern geteilt wird.
- *Processing-Space (pro Benutzerinteraktion)*: Der Processing-Space stellt den dynamischen Anteil an Speicherverbrauch dar und errechnet sich aus der durchschnittlichen Menge an gesammelten Garbage-Speicher pro Benutzerinteraktion.

Abbildung 3-13 stellt den Messvorgang einer Java-Applikation bei einem Mehr-Benutzer-Lasttest dar. Damit die gemessenen Daten nicht verfälscht werden, wird ein voller GC-Lauf (Mark, Sweep, Compact) im Vorfeld durchgeführt. Der Framework-Space kann nach dem Starten (und Warm-Up) der JVM festgestellt werden. Der User-Session-Space errechnet sich mittels

$$UserSessionSpace = \frac{(M - FrameworkSpace)}{n}$$

wobei Mdie Speichergröße und n die Anzahl der eingeloggten Benutzer darstellen. Der Processing-Space errechnet sich mittels

$$ProcessingSpace = \frac{Gesammelte\ Menge\ an\ Garbage\ f\ddot{u}r\ k\ Benutzerinteraktionen}{k}$$

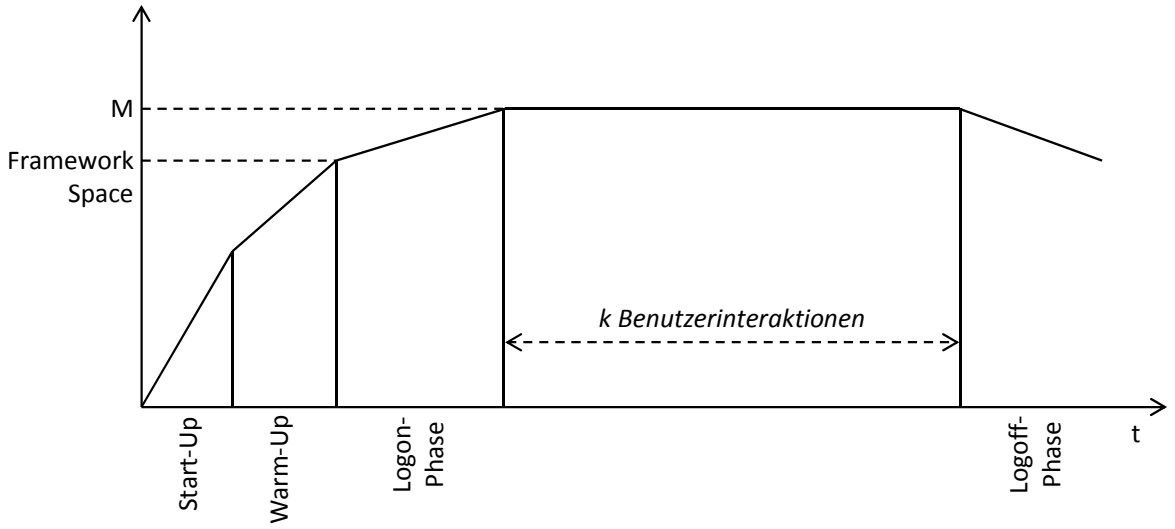


Abbildung 3-13: Messverfahren bei einem Mehrbenutzer-Lasttest einer Java-Applikation
Quelle: Cheng (2008)

Wie bereits erwähnt, eignet sich diese Performance-Betrachtung vor allem für eine konkrete Java-Anwendung, beispielsweise für die Code-Optimierung. Für die Leistungsanalyse eines SAP-Netweaver-Portalsystems unter einer bestimmten Last- und Laufzeitkonfiguration, wie sie in dieser Arbeit durchgeführt wird, haben diese Indikatoren wenig Aussagekraft.

Wie in Kapitel 5.8 noch dargestellt wird, nimmt der Garbage-Collector einen entscheidenden Einfluss auf die Performance des Portalsystems, insbesondere im Hochlastbereich. Daher ist es unabdingbar, das Verhalten des Garbage-Collectors zu analysieren. Zur Evaluierung des GC-Einflusses können vor allem zwei Indikatoren herangezogen werden (Cheng 2008):

- *GC-Dauer*: Die GC-Dauer gibt die (durchschnittliche) Laufzeit eines GC-Zyklus an. Ein hoher Wert hat einen negativen Einfluss auf die System-Performance.

- *GC-Intervall*: Das GC-Intervall gibt die (durchschnittliche) Zeit zwischen den GC-Zyklen an. Unter der Annahme, dass die durchschnittliche Lebensdauer eines Objektes konstant bleibt, ist ein GC-Zyklus umso effizienter, je höher das Intervall zwischen zwei GC-Zyklen ist.

Abhängig von dem Workload und GC-Konfiguration sowie den genannten GC-Indikatoren nimmt der Garbage-Collector einen wesentlichen Einfluss auf die System-Performance und somit die Antwortzeit einzelner Benutzer-Tasks.

3.2 Monitoring des SAP-Netweaver-AS-Java

Im vorherigen Kapitel (3.1) wurden die Architektur des AS-Java dargestellt und dabei die einzelnen Komponenten sowie leistungsbeeinflussende Elemente (Sperrobjekte, Speichermanagement, Pufferung) beschrieben. In diesem Kapitel wird die Monitoring-Infrastruktur eines AS-Java-Systems aufgezeigt. Dabei wird das Hauptaugenmerk auf die für diese Arbeit relevanten Leistungsdaten gelegt.

Grundsätzlich sollte unterschieden werden, zu welchem Zweck die Überwachung bestimmter Systemeigenschaften dient:

- Dem Entdecken von Problemen, wie zum Beispiel durch Verfügbarkeits-Überwachung oder dem Aufdecken von Ressourcen-Engpässen. Die Datensammlung ist dabei meist unvollständig, da nur kritische Werte aufgenommen werden.

- Der allgemeinen Systemanalyse, beispielsweise hinsichtlich Antwortzeiten der einzelnen Komponenten oder Ressourcen-Verbrauch einer bestimmten Applikation. Die Datensammlung erfolgt dabei laufend und ist unabhängig von einer speziellen Problembetrachtung.

Für die in dieser Arbeit durchgeführten Performance-Messungen – zum einen für die Parametrisierung des Warteschlangenmodells, zum anderen für die Bewertung der Simulationsergebnisse – sind vor allem die Werkzeuge zur allgemeinen Systemanalyse von Bedeutung, da nicht nur Problemsituationen aufgedeckt, sondern die Leistungsdaten kontinuierlich gesammelt werden sollen. Abbildung 3-14 zeigt die verschiedenen Monitoring-Werkezeuge und ordnet sie dabei ihrem Detaillierungsgrad zu. Die drei abgerundeten Blöcke stellen den unterschiedlichen Verwendungszweck dar.

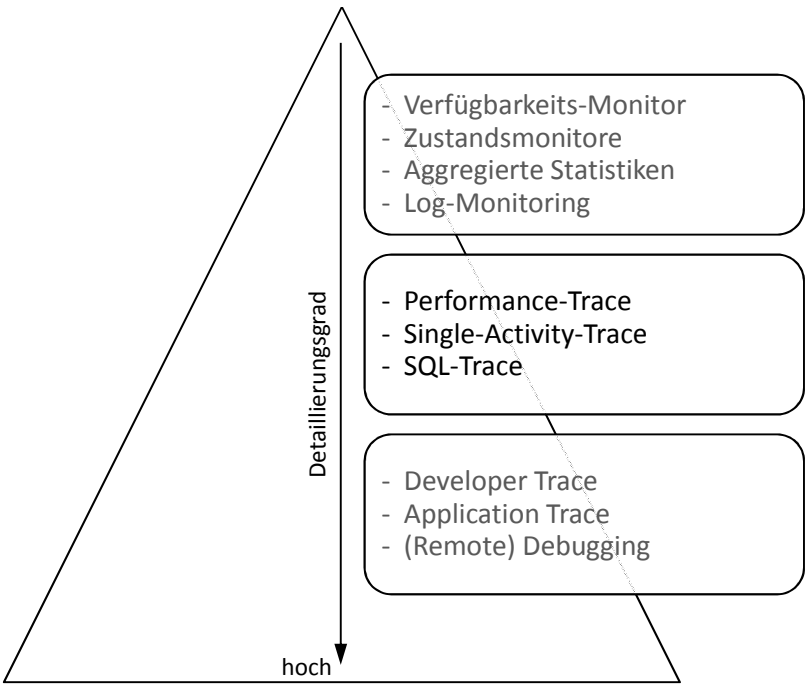


Abbildung 3-14: Monitoring-Werkzeuge im SAP-Netweaver-AS-Java
Quelle: eigene Darstellung

Der erste Block (oben) beinhaltet Werkezeuge zur allgemeinen Überwachung des Systemzustands:

- Verfügbarkeitsmonitoring der Applikationen und Java-Engines wird über den sogenannten „Generic Request and Message Generator“ (GRMG) erreicht.

- Zustandsmonitore überwachen die einzelnen Java-Engines.

- Log-Dateien protokollieren die einzelnen Aktivitäten auf einer allgemeinen Ebene.
- Aggregierte Statistiken beinhalten verdichtete Leistungsdaten, die zur Übersicht der Systemleistung genutzt werden. Angezeigt werden Antwortzeiten, Wartezeiten, CPU-Zeiten, Netzzeiten, usw.

Der dritte Block (unten) beinhaltet Entwicklerwerkzeuge für die Sammlung von Leistungsdaten auf Quellcode-Ebene. So zeigt beispielsweise der Applikations-Trace einzelne Methodenaufrufe sowie ihre Bearbeitungszeiten.

Der zweite Block (Mitte) umfasst schließlich Werkzeuge zur Systemanalyse auf Komponentenebene und wird im Folgenden näher betrachtet.

3.2.1 Java-Application-Response-Time-Measurement

Das „Java Application Response Time Measurement“ (JARM) bildet die technische Basis der Leistungsdatenerhebung und ermöglicht primär das Sammeln von Antwortzeiten einer Java-Applikation. Neben den Antwortzeiten der einzelnen Komponenten werden auch der Benutzer, der die Anfrage gesendet hat, sowie die Menge an Daten, die transferiert wurden, angegeben. Das JARM-Monitoring ist, im Gegensatz zu dem Performance-Trace (siehe Kapitel 3.2.3), auch bei Produktivsystemen standardmäßig eingeschaltet und kann somit als nicht intrusiv gewertet werden. Die Leistungsdaten, die in der JARM-Ansicht zur Verfügung gestellt werden, sind in Tabelle 3-9 dargestellt.

Element	**Beschreibung**
Benutzeranfrage	
Request Name	Name der Anfrage
Response Time	Dauer der Bearbeitung (Antwortzeit, ms)
Outbound Data	Menge an gesendeten Daten (Bytes)
Components	Anzahl an Komponenten
Comp (Max Time)	Komponente mit der höchsten Bearbeitungszeit
Data	Die Menge an Daten, die bei der Komponente mit der höchsten Bearbeitungszeit gesendet wurden.
Status	Monitoring-Status: 0 kein Fehler. > 0 Fehler bei der Erfassung der Daten.
Start Time	Zeitstempel, wann die Bearbeitung der Anfrage gestartet wurde.
User	Benutzer-ID
Description	Beschreibung der Anfrage
Level	Monitoring-Level (Detailstufe)
Komponente	
Component Name	Name der Komponente
Avg Gross Time	Durchschnittliche Bruttozeit (ms) – Dies beinhaltet die Bearbeitungszeit der Komponente und evtl. Sub-Komponenten.
Avg Net Time	Durchschnittliche Nettozeit (ms) – Dies beinhaltet nur die Bearbeitungszeit innerhalb dieser Komponente.

Avg Outbound Data	Durchschnittliche Menge an Daten, die gesendet wurde (Bytes).
Calls	Anzahl an Komponenten-Aufrufen
Sum Gross Time	Summe der Bruttozeiten aller Aufrufe (ms)
Sum Net Time	Summe der Nettozeiten aller Aufrufe (ms)
Sum Outbound Data	Summe der gesendeten Daten (Bytes)
Comp Gross Time Provided	Anzahl der Aufrufe, die ordnungsgemäß beendet wurden
Comp Net Time Provided	Anzahl der Aufrufe, bei denen alle Sub-Komponenten ordnungsgemäß beendet wurden
Comp Data Provided	Anzahl der Aufrufe, die Daten erzeugt haben
Properties	Eigenschaften der Komponente
Description	Beschreibung der Komponente
Benutzer	
User Name	Benutzername
Avg Time	Durchschnittliche Zeit einer Anfrage
Total Time	Aufsummierte Zeit aller Anfragen (ms)
Num of Requests	Anzahl der gesendeten Anfrage
First Request	Zeitstempel der ersten Anfrage, die von diesem Benutzer gesendet wurde.
Last Request	Zeitstempel der letzten Anfrage, die von diesem Benutzer gesendet wurde
Threads	
Thread	Der Name des Threads
User	Benutzer-ID
Request	Name der Anfrage
Req Time	Bereits investierte Bearbeitungszeit für diese Anfrage (ms)
Component	Der Name der derzeit ausgeführten Komponente
Comp Time	Bearbeitungszeit innerhalb dieser Komponente (ms)
Action	Derzeit durchgeführte Aktion
Start Time	Zeitstempel der ersten Anfrage in diesem Thread
Sum Active Time	Aufsummierte Bearbeitungszeiten aller Anfragen in diesem Thread.

Tabelle 3-9: Java-Application-Response-Time-Measurement-(JARM)-Daten
Quelle: eigene Darstellung in Anlehnung an SAP (2011e)

Die zur Verfügung gestellte JARM-Ansicht (siehe Abbildung 3-15) kann jedoch nur als Übersicht dienen, da Durchschnittswerte berechnet und ausgegeben werden. Für die Performance-Analyse von Portalen ist sie allerdings hilfreich, um einen Überblick über das Leistungsverhalten zu erhalten. Speziell bei der Durchführung der Performance-Messungen konnten damit die Threads sowie die gesendeten Benutzeranfragen überwacht und potentielle Abbrüche schnell erkannt werden. Erst wenn die Überprüfung dieser aggregierten Daten keine Anomalien aufzeigten, wurden die erhobenen Leistungsdaten im Detail untersucht.

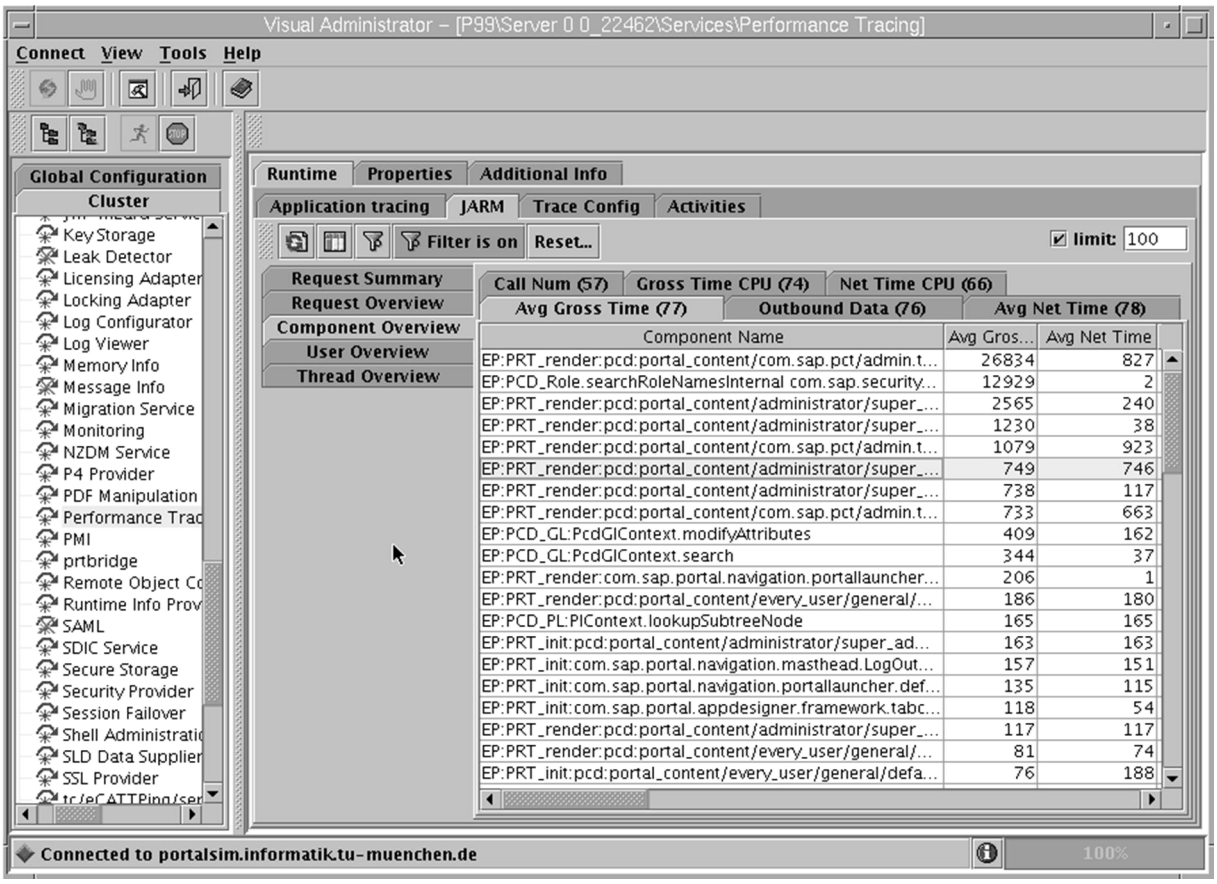


Abbildung 3-15: Komponentenansicht der JARM-Daten
Quelle: eigene Darstellung

3.2.2 Single-Activity-Trace

Der Single-Activity-Trace (SAT) verwendet die von JARM zur Verfügung gestellten Daten, allerdings werden hierbei alle erfassten Leistungsdaten aufgezeichnet. Es können somit die Benutzeranfragen einzeln analysiert und die logischen Abarbeitungsschritte (Komponenten) für jede Anfrage zusammen mit ihren Performance-Daten erfasst werden. Der Trace wird über die Logging-API des SAP-Systems automatisch in eine Datei *sat.trc.<n>* geschrieben, die standardmäßig unter folgendem Pfad zu finden ist:

/usr/sap/<*SAPSID*>/JC<*J2EE-Instanz*>/j2ee/cluster/server<*n*>/log

Die Trace-Datei kann über den Log-Viewer, einer mitgelieferten Java-Applikation angezeigt und als Comma-Separated-Value-Datei (CSV-Datei) exportiert werden. Das Ursprungsformat trennt die einzelnen Werte eines Eintrags mit dem Rautenzeichen.

Für die Zuordnung der Einträge zu einer bestimmten Transaktion wird neben den Leistungsdaten auch eine Transaktions-ID erfasst. Dadurch können diese den Einträgen anderer Traces, beispielsweise des SQL-Traces (siehe Kapitel 3.2.4), zugeordnet werden (siehe Abbildung 3-16).

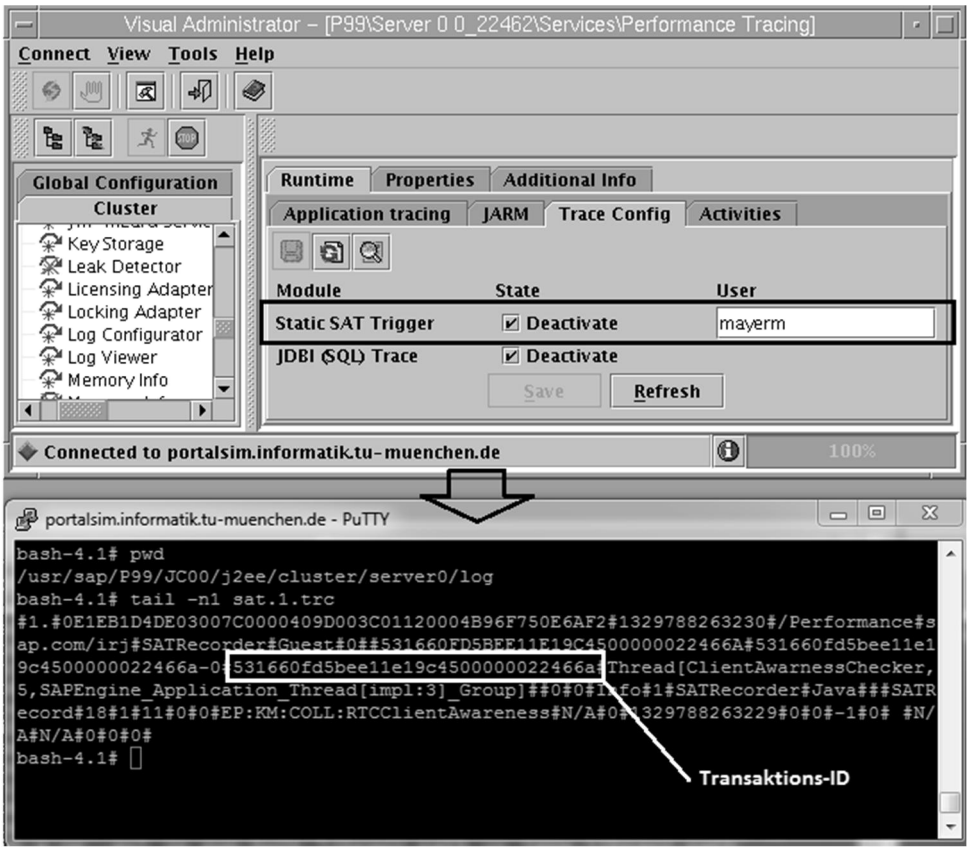


Abbildung 3-16: Single-Activity-Trace
Quelle: eigene Darstellung

3.2.3 Funktionaler Trace

Der funktionale Trace greift auf die von JARM zur Verfügung gestellten Daten und „Distributed Statistical Records“ (DSRs) zu. DSR ist ein Konzept, das ursprünglich für den ABAP-Stack entwickelt und später auf J2EE-Systeme erweitert wurde. Statistiken und Trace-Daten, die von dem DSR-Dienst innerhalb der J2EE-Engine generiert werden, werden über einen speziellen Agenten des „Computing Center Management Systems“ (CCMS) an ein zentrales Monitoring-System („Central Monitoring System“, CEN) gesendet.

Abbildung 3-17 zeigt den Ablauf, wie die Statistikdatensätze und die Rohdaten in das zentrale Monitoring-System gelangen. Dieser lässt sich wie folgt beschreiben:

1. In der J2EE-Engine ist der DSR-Dienst für die erhobenen Leistungsdaten verantwortlich und kommuniziert mit der Java-DSR-API, einer Java-Bibliothek, die für die Verwaltung der DSR-Datensätze verantwortlich ist.

2. Die Datensätze werden mittels der Write-API vom Hauptspeicher in das Dateisystem unter dem Pfad /<J2EE Installationsverzeichnis>/prfclog/dsr geschrieben.

3. Der CCMS-Agent liest die Trace-Dateien und gibt sie an das Monitoring-System weiter. Dabei werden aggregierte Statistikdaten und Einzeldatensätze (Statistikrohdaten) unterschieden:

- Die Statistikdaten werden vom DSR-Collector stündlich gesammelt, aggregiert und in die CCMS-Datenbank geschrieben. Über die SAP-Transaktion ST03G können diese angezeigt werden. Damit die Leistungsdaten, die in der letzten Stunde gesammelt und folglich noch nicht in die Datenbank geschrieben wurden, ebenfalls aufgenommen werden, können die Daten auch direkt vom DSR-Collector gelesen werden.

- Einzeldatensätze (Statistikrohdaten) und Performance-Traces können über die SAP-Transaktion STATTRACE (auch funktionaler Trace genannt) abgerufen werden. Der funktionale Trace bietet die feinste Granularität der erfassten Leistungsdaten.

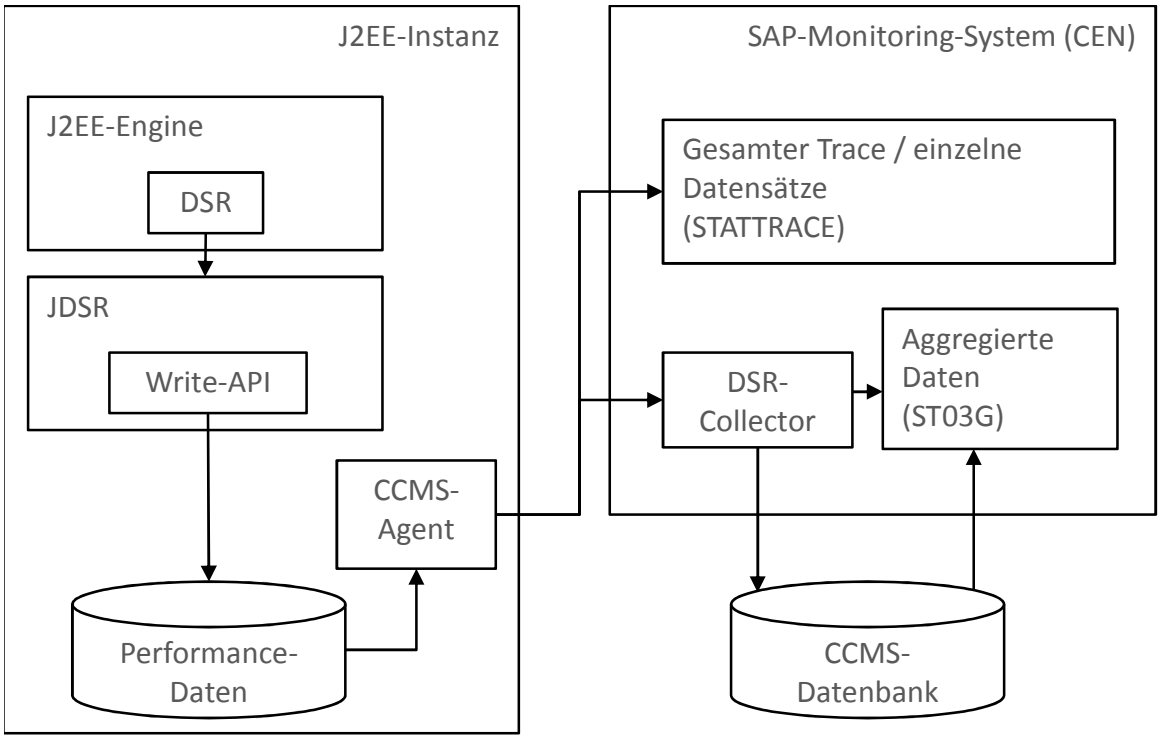


Abbildung 3-17: Architektur des zentralen Monitoringsystems (CEN)
Quelle: eigene Darstellung

In den DSR-Datensätzen wird ebenfalls festgehalten, auf welchem Service-Typ sich die Leistungsdaten beziehen. Die Servicearten einer J2EE-Instanz teilen sich in Web-Anfragen, EJB-Anfragen, Sicherheits-Tasks und System-Tasks auf (vgl. Container-Typen des Java-Servers in Kapitel 3.1.1). Zudem werden Datensätze zu Datenbankaufrufen festgehalten.

Da sich Portaloperationen über verschiedene Instanzen, unterschiedlichen technologischen Stacks (ABAP, Java) und Hosts erstrecken können, ist es notwendig, dass die Leistungsdaten den „Logical Units of Work“ (LUW), mit anderen Worten der logischen Sicht von einzelnen Benutzeraktionen, zugeordnet werden können. Dazu wird eine Transaktion-ID („Global Unique Identifier“, GUID) generiert und an die nachfolgende Komponente übergeben (siehe Abbildung 3-18).

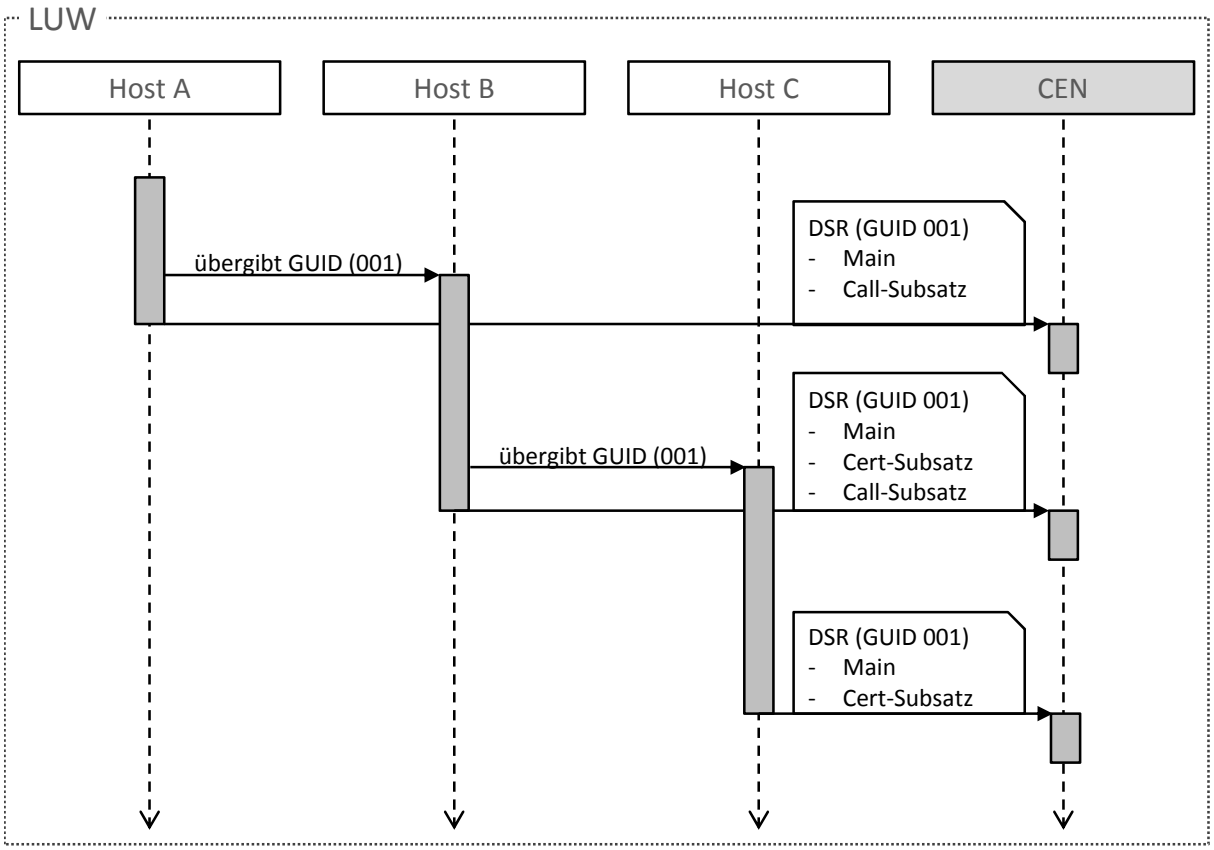


Abbildung 3-18: Zuordnung der Leistungsdaten zu logischen Arbeitsschritten
Quelle: eigene Darstellung in Anlehnung an SAP (2011f)

Neben der Transaktion-ID ist der DSR-Datensatz in einen Hauptteil („Main“) und, falls zutreffend, in Unterdatensätze gegliedert. Ein Call-Subsatz beinhaltet Informationen zu der aufgerufenen Komponente, wogegen ein Cert-Subsatz die Quelle der LUW spezifiziert. Jeder DSR-Datensatz kann eine beliebige Anzahl an aufgerufenen Komponenten und somit Call-Subsätze aufweisen.

Für die in dieser Arbeit durchgeführten Messungen ist dieses Prinzip vor allem für die Zuordnung der Datenbankaufrufe sehr hilfreich. Für jeden Datenbankaufruf wird ein Call-Subsatz erstellt, sodass die einzelnen Datensätze der Benutzeraktion (LUW) zugeordnet werden können. Somit errechnet sich nach SAP (2011g) die reine Bearbeitungszeit mittels:

$$Bearbeitungszeit = Antwortzeit - \Sigma\, Wartezeiten - \Sigma\, Datenbankzeiten$$

Da die J2EE-Engine für die DSRs nicht transparent ist, werden neben den DSR-Daten noch zusätzliche Performance-Traces geschrieben und somit eine noch detailliertere Erfassung der Performance-Daten erreicht (siehe Abbildung 3-19).

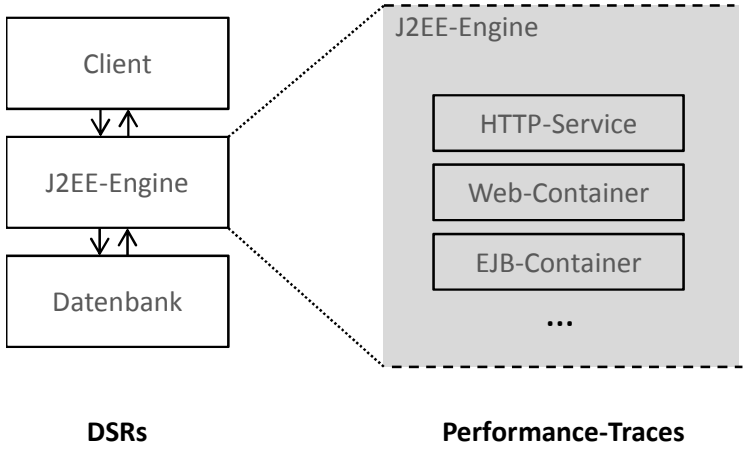


Abbildung 3-19: Granularität der DSR-Datensätze und Performance-Traces
Quelle: eigene Darstellung

Über die SAP-Transaktion STATTRACE kann der funktionale Trace (Statistikrohdaten und Traces) angezeigt werden. Der funktionale Trace ist eine Erweiterung der Datenselektionstransaktion STAD, die Statistikrohdaten nur für ein lokales ABAP-System wiedergeben kann. Zudem wird der globale Systemlastmonitor (SAP-Transaktion ST03G) ergänzt, der nur aggregierte Daten anzeigt. Zur Auswahl sowie Darstellung der Daten werden folgende Funktionen bereitgestellt:

- Über die Systemauswahl können die Systeme ausgewählt werden, für die die Statistikdatensätze und Traces analysiert werden sollen.
- Mittels Datenselektion können die zu analysierenden Daten ausgewählt werden. Dabei kann nach folgenden Kriterien gefiltert werden:
  - Startdatum
  - Startzeit
  - Lesezeitraum
  - Transaktions-ID (GUID)

  Die gefilterten Statistiksätze und Traces werden anschließend in der Analysesicht angezeigt.
- Die Darstellung der Analysesicht kann hierarchisch oder als Liste erfolgen. In der Hierarchieanzeige wird zunächst eine Liste mit Initialsystemen angezeigt. Wenn eine Benutzeraktion (LUW) in weiteren Komponenten oder Instanzen verarbeitet wurde, finden sich in den Ebenen darunter die entsprechenden Aufrufe sowie deren Performancedaten.

Zu den einzelnen Datensätzen können diverse Detailansichten aufgerufen werden (siehe Abbildung 3-20):

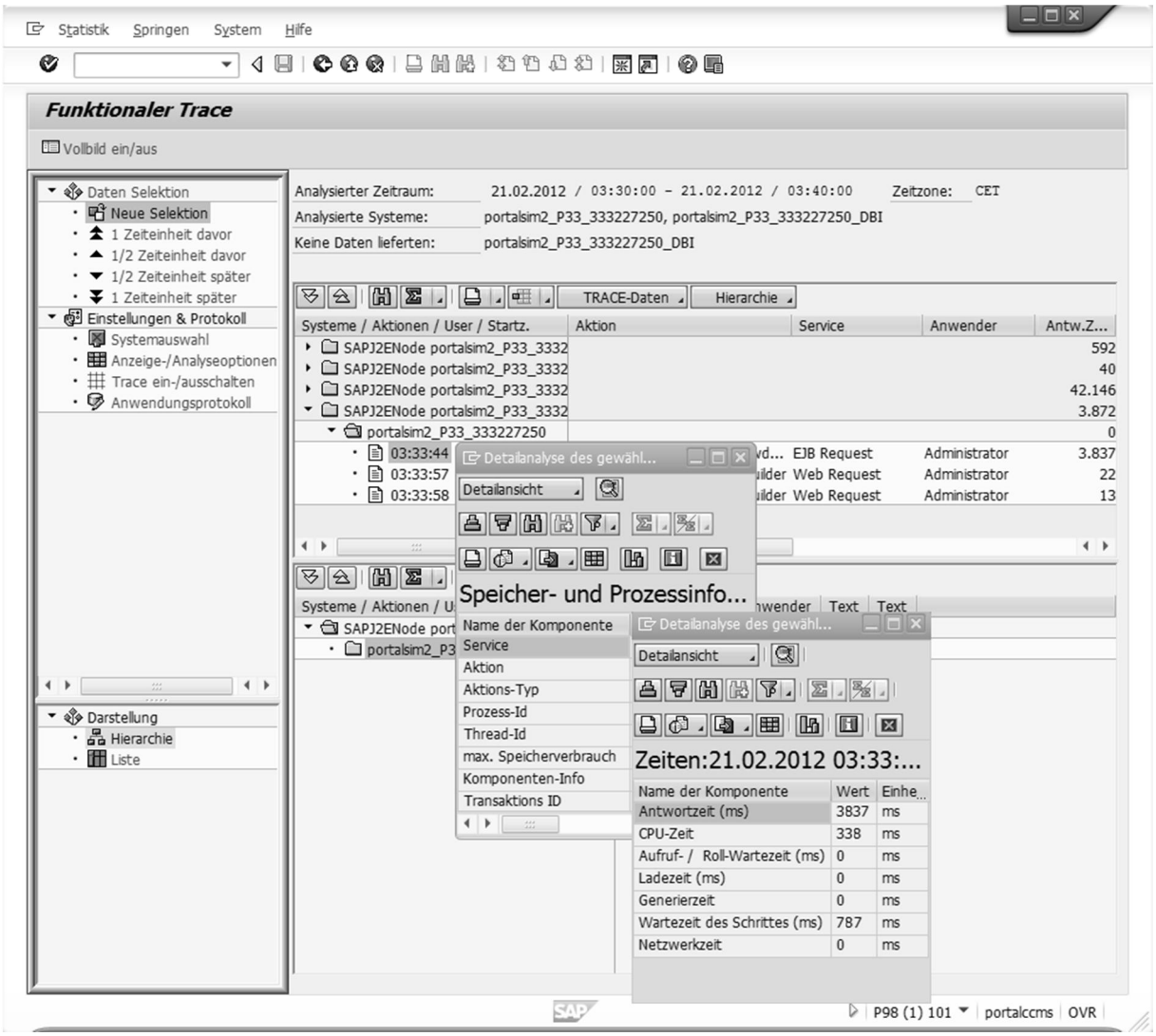


Abbildung 3-20: Anzeige des funktionalen Trace
Quelle: eigene Darstellung

- Zeiten (CPU, Ladezeit, Wartezeit, Netzwerkzeit, etc.)
- Speicher- und Prozessinformationen, bei denen unter anderem die Transaktions-ID (GUID) festgehalten wird.
- Datenbank-Prozeduren (Request-Typ, Anzahl der Aufrufe, Datenbankzeit, etc.)
- In der Client-Info-Detailansicht ist der Cert-Subrecord zu finden, der Informationen zur Quelle der LUW (dem Beginn der Benutzeraktion) bereitstellt.

Für die Auswertung der Einzeldatensätze können die Performance-Daten exportiert und beispielsweise in Excel grafisch analysiert werden (siehe Abbildung 3-21).

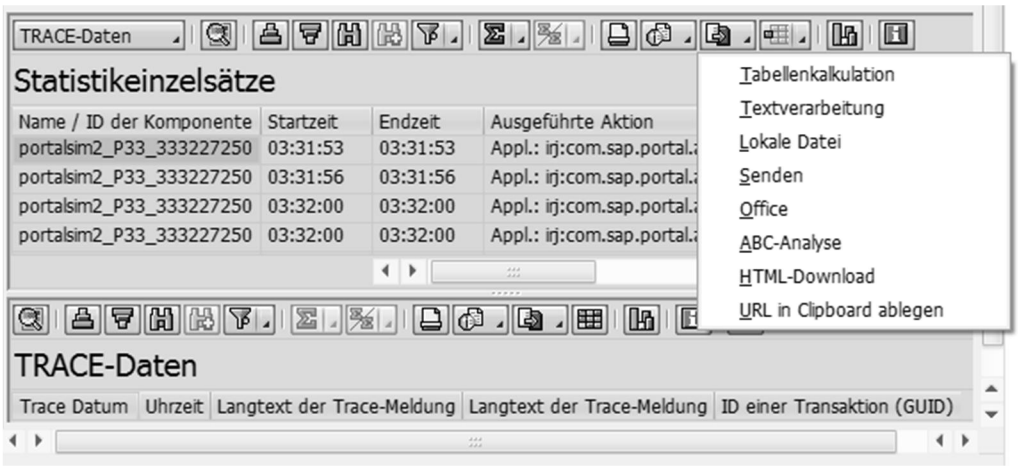


Abbildung 3-21: Export der Performance-Daten zur externen Analyse
Quelle: eigene Darstellung

Für die in dieser Arbeit durchgeführten Performance-Analysen spielt der funktionale Trace eine zentrale Bedeutung, da die Leistungsdaten der einzelnen Komponenten zur Verfügung gestellt werden.

3.2.4 SQL-Trace

Der SQL-Trace protokolliert alle Datenbankaufrufe mit Open- und Native-SQL-Statements. Neben seinem Einsatz bei der Entwicklung von Java-Programmen, wird der SQL-Trace auch für die Performance-Analyse verwendet. Nicht zuletzt ist der SQL-Trace wichtig für die Erkennung von Tabellenpuffer-Operationen, die beispielsweise bei der Verdrängung oder Invalidierung (siehe Kapitel 3.1.4) auftreten und somit nicht Inhalt der eigentlichen Aufgabe bzw. Benutzeraktion sind. Für die korrekte Parametrisierung des Warteschlangenmodells müssen daher inhaltliche und infrastrukturelle Datenbankaufrufe getrennt werden.

Der SQL-Trace kann über die Web-Applikation „Open SQL Monitors“ aktiviert und die aufgezeichneten Daten evaluiert werden (siehe Abbildung 3-22). Die zur Verfügung gestellten Daten können anhand unterschiedlichster Parameter (Sitzungs-ID, DSR-Transaktions-ID, Benutzer, Dauer, Applikation, usw.) gefiltert oder auch als XML-Datei exportiert werden. Eine entsprechende Document-Type-Definition-Datei (DTD-Datei) wird ebenfalls bereitgestellt. Die für die Performance-Analyse relevanten Daten sind in Tabelle 3-10 aufgelistet.

Attribut	Beschreibung
Statement	SQL-Statement oder JDBC-Methodenaufruf.
Time	Startzeitpunkt des JDBC-Methodenaufrufs.
Location	Name der JDBC-Methode.
Duration	Dauer des Methodenaufrufs in Mikrosekunden.
J2EE User	J2EE-User, der den Methodenaufruf durchgeführt hat.
J2EE Session	J2EE-Sitzungs-ID.
J2EE Application	Name der J2EE-Applikation, die den Methodenaufruf initiiert hat.
DSR Transaction ID	Zugehörige Transaktions-ID.
Database ID	String, der die verwendete Datenbankverbindung angibt. Dieser besteht aus dem Datenquellennamen und dem Daten-

	bankbenutzer.
Thread	Thread, der den JDBC-Aufruf durchgeführt hat.
Prepared Statement ID	Eindeutige ID des SQL-Statements.
ResultSetID	Eindeutige ID des Ergebnisdaten-Sets.
Input Parameter	Übergebene Parameter des Methodenaufrufs.
Table Names	Alle Tabellennamen, die bei diesem JDBC-Aufruf involviert waren.
DB Error Code, *DB Error SQL State*	Falls ein SQL-Fehler auftritt, wird hier der Code und Status angegeben.

Tabelle 3-10: SQL-Trace-Informationen für die Performance-Analyse
Quelle: eigene Darstellung

Sollten nun bei einer Benutzeraktion des für die Performance-Evaluation verwendeten Workloads detaillierte Informationen zu den durchgeführten Datenbankaufrufen benötigt werden, können diese über die Transaktions-ID entsprechend zugeordnet werden. Vor allem für die bereits erwähnte Analyse von inhaltlichen und infrastrukturellen Datenbankaufrufen sind die im SQL-Trace aufgezeichneten Daten sehr wertvoll. Ein Beispiel zum Aufruf einer bestimmten Transaktions-ID ist in Abbildung 3-22 dargestellt.

Abbildung 3-22: SQL-Trace einer bestimmten Transaktions-ID
Quelle: eigene Darstellung

3.2.5 Black-Box-Monitoring

Neben der Leistungsdatenanalyse auf Komponentenebene ist es zu Kontrollzwecken sinnvoll, Daten über die einzelnen Benutzeranfragen sowie deren Dauer (Antwortzeit) auf der Ebene der HTTP-Requests zu sammeln (HTTP-Access-Trace), ohne sich die einzelnen Komponenten- oder Datenbankzeiten im Detail anzuschauen. Dies kann unter anderem zur Vergleichsanalyse der aufsummierten Komponentenzeiten dienen, aber auch bei den in dieser Arbeit durchgeführten Messungen des Portalsystems können so die Antwortzeiten (ohne Netzwerkzeit vom Client zum Server) der einzelnen Benutzeraktionen protokolliert werden.

Wie bereits in Kapitel 3.1.1 erwähnt, bietet der http-Provider-Service eine solche Funktionalität. Standardmäßig werden die Daten im folgenden Format festgehalten:

[Datum] – Client-IP : Request Return-Code Bytes

Zusätzlich können noch eine Reihe weiterer Daten gesammelt werden, unter anderem die bereits angesprochene Antwortzeit des Aufrufs. Interessant für die Performance-Analyse ist vor allem auch der Parameter, ab welcher Dauer (in Millisekunden) der HTTP-Request in den Trace geschrieben werden soll. Dadurch kann die Anzahl der analysierten Benutzeraktionen erheblich reduziert werden, was vor allem bei sehr umfangreichen Workloads sehr hilfreich sein kann (siehe Abbildung 3-23).

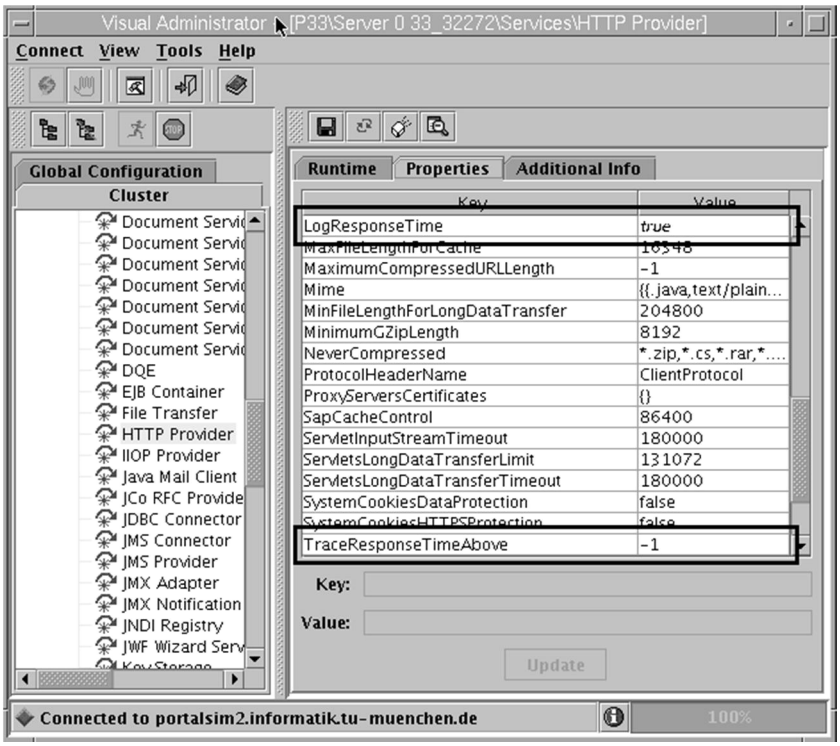


Abbildung 3-23: Aktivierung des HTTP-Access-Trace inklusive Antwortzeiten
Quelle: eigene Darstellung

Die Trace-Datei findet sich unter

./log/system/httpaccess/responses.trc

und kann für jeden Java-Server und Dispatcher einzeln konfiguriert werden.

3.2.6 Monitoring des Garbage-Collectors

Wie bereits in Kapitel 3.1.5 beschrieben, können über den IBM-JVM-Parameter *verbose:gc* Informationen zum Verhalten des Garbage-Collectors protokolliert werden. Die Daten werden standardmäßig in die Server-Log-Datei *std_server#.out* geschrieben, die unter */usr/sap/<SID>/JC#/work* zu finden ist. Die Log-Datei kann später über diverse Analyseprogramme, die von IBM bereitgestellt werden, ausgewertet werden. Ebenso ist es möglich, die GC-Daten grafisch darzustellen. Zwei der Analyseprogramme sind bei den Garbage-Collector-Analysen in dieser Arbeit zum Einsatz gekommen und sollen in diesem Kapitel kurz vorgestellt werden.

Die zwei vorgestellten Hilfsprogramme geben Aufschluss über den GC-Overhead und dessen Einfluss auf die Bearbeitungszeit und somit Gesamtantwortzeit der durchgeführten Tasks. Speziell im Hochlastbereich spielt das Verhalten des Garbage-Collectors eine entscheidende Rolle, da eine zunehmende Anzahl an GC-Pausen immer weniger Ausführungszeit der Applikationen zulassen. Dies kann zu stark ansteigenden Antwortzeiten und im schlimmsten Falle zu einem sogenannten Java-Resonanz-Effekt, bei dem die CPU-Auslastung zwischen 0 und 100 Prozent oszillieren kann (Cheng/Morrison 2007; Oracle 2011), führen.

Pattern Modeling and Analysis Tool

Das „Pattern Modeling and Analysis Tool for IBM Java Garbage Collector“ (PMAT) (IBM 2011g) ist ein Java-Programm, das die Protokolldatei mit den detaillierten GC-Daten einliest. Dabei werden folgende Daten berücksichtigt:

- Java-Heap-Größe
- Java-Heap-Verbrauch
- GC-Zeiten
- Zeiten der GC-Mark-Phasen
- Zeiten der GC-Sweep-Phasen
- Zeiten der GC-Compact-Phasen
- Zur Verfügung stehender Java-Heap-Speicher
- Gewonnener Java-Heap-Speicher

Sobald die Daten eingelesen wurden, wird ein Diagnose-Algorithmus gestartet, der eventuelle Speicherprobleme erkennt, wie beispielsweise Java-Heap-Fragmentierungen oder zu große Speicheranfragen, die nicht bedient werden konnten (siehe Abbildung 3-24).

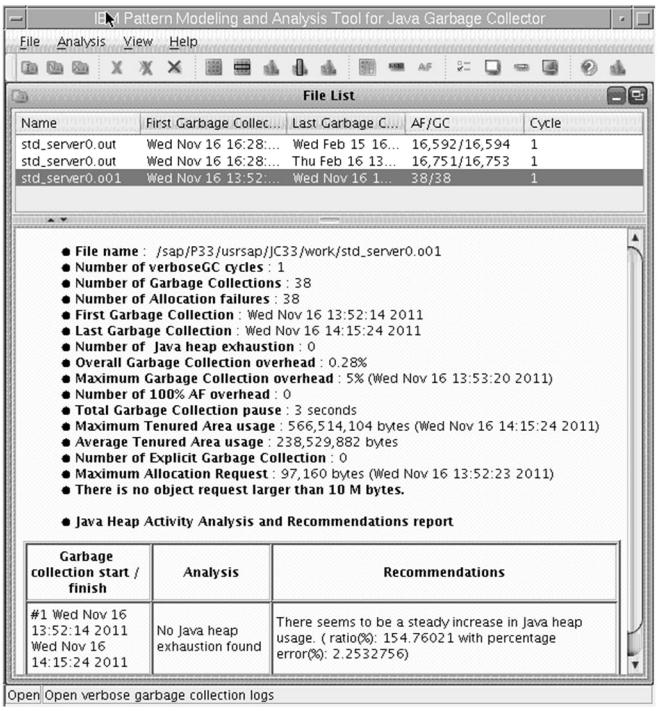


Abbildung 3-24: Pattern Modeling and Analysis Tool
Quelle: eigene Darstellung

Garbage Collector and Memory Analyzer

Der „Garbage Collector and Memory Analyzer“ *(IBM 2011c)* ist ebenfalls ein Analysewerkzeug, das von IBM bereitgestellt wird. Es liest die GC-Logs aus und stellt diverse grafischen Ausgaben zur Verfügung, die das GC-Verhalten über eine gewisse Zeitspanne darstellen (siehe Abbildung 3-25).

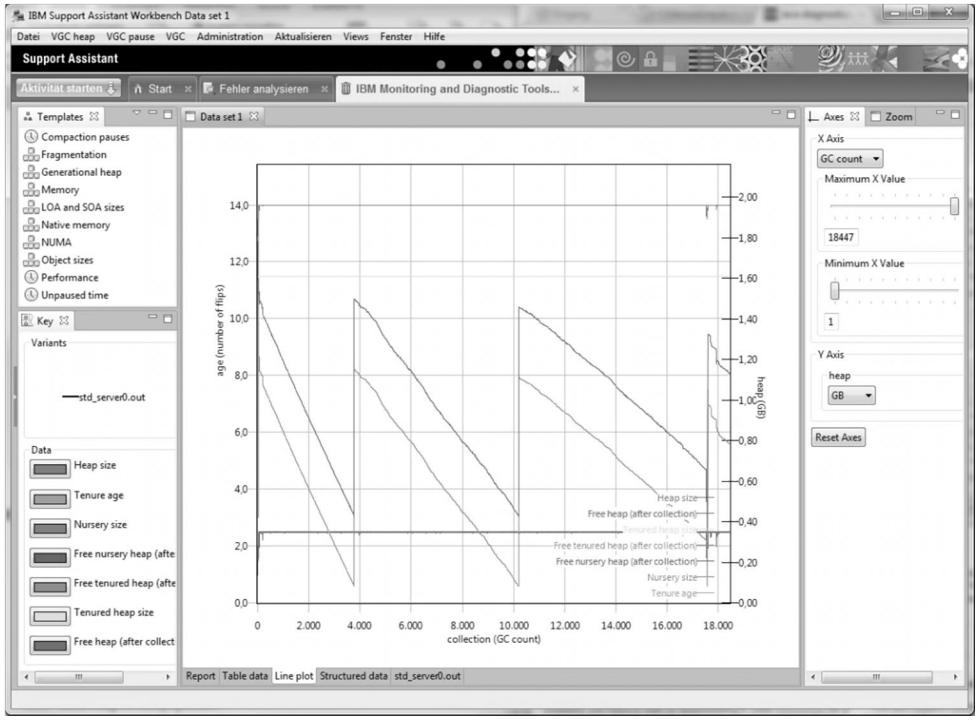


Abbildung 3-25: Garbage Collector and Memory Analyzer
Quelle: eigene Darstellung

3.3 Monitoring auf Betriebssystemebene

Neben der Überwachung des SAP-System-Verhaltens ist ein weiterer, ebenso wichtiger Aspekt das Performance-Verhalten auf Betriebssystemebene. Es wird zwischen drei fundamentalen Systemressourcen unterschieden, die aus der Von-Neumann-Architektur folgen (Eilert et al. 2003, 16):

- CPU
- Hauptspeicher
- I/O-Subsystem

Im Folgenden wird ein kurzer Überblick über die verschiedenen Betriebssystem-Hilfsmittel zur Überwachung der Systemressourcen gegeben. Dabei bezieht sich die Beschreibung auf das in dieser Arbeit eingesetzte Betriebssystem AIX (IBM 2011b), wobei viele der vorgestellten Tools auch für andere Unix-Derivate zur Verfügung stehen, wenngleich in leicht veränderter Form.

Tabelle 3-11 gibt einen kurzen Überblick über die in AIX vorhandenen Performance-Tools für die drei fundamentalen Systemressourcen sowie die explizit dargestellten Netzwerk- und Prozessmonitore.

Ressource	**Programm**
CPU	topas, mpstat, vmstat, iostat, sar, time/timex
Hauptspeicher	topas, vmstat, lsps, ipcs, ps
I/O-Subsystem	topas, iostat, vmstat, lvmstat, lsps, lsdev, lspv/lsvg/lslv
Netzwerk	topas, netstat, atmstat, entstat, tokstat, fddistat, nfsstat
Prozesse/Threads	topas, pstat, ps

Tabelle 3-11: Monitoring-Tools auf Betriebssystemebene (AIX)
Quelle: eigene Darstellung

Einige Betriebssystem-Hilfsmittel überschneiden sich in ihrer Funktionalität und liefern Performance-Daten für verschiedene Systemressourcen. Ein generelles Werkzeug zur Leistungsüberwachung ist „topas“ (IBM 2011h). Vor allem bei den in dieser Arbeit verwendeten virtuellen Hosts, sogenannten logischen Partitionen (LPAR) (Harris et al. 2005), bietet „topas“ die Möglichkeit zwischen virtuellen Ressourcen und Ressourcen des physikalischen Hosts zu unterscheiden.

Zusätzlich zur interaktiven Wiedergabe der aktuell verwendeten Systemressourcen mittels „topas“ werden zwei Hilfsmittel zur Aufzeichnung und Ausgabe der Daten bereitgestellt: „topasrec“ (IBM 2011k) und „topasout“ (IBM 2011j).

„topasrec“ erstellt während der Aufzeichnung eine Binärdatei mit den lokalen System-Metriken, Cross-Eletronic-Circuit-Metriken (CEC-Metriken) und Cluster-Metriken. Bei der Verwendung von „topasrec“ wurden folgende Parameter verwendet:

topasrec -C -c <Anzahl> -s <Intervall> -o <Ausgabedatei>

Der Parameter „-C“ gibt an, dass auch die CEC-Metriken aufgezeichnet werden, „-c“ spezifiziert die Anzahl der Aufzeichnungen, „-s“ definiert das Intervall zwischen den Aufzeichnungen und anhand des Parameters „-o“ wird die Ausgabedatei festgelegt.

Die aufgezeichnete Binärdatei kann im Anschluss mit dem Hilfsmittel „topasout“ zu einer CSV-Datei konvertiert bzw. kopiert werden. Dabei wurden die Parameter „-c“ für die Ausgabe in eine CSV-Datei und „–R <Typ>“ zum Spezifizieren der gewünschten Daten verwendet. Tabelle 3-12 stellt die verfügbaren Leistungsdaten dar:

<Typ>	**Beschreibung**
summary	Gibt eine Übersicht zur Ressourcenverwendung wieder.
detailed	Gibt einen detaillierten Bericht zur Ressourcenverwendung aus.
lan	Gibt an, wieviele Daten über das Netzwerk gesendet bzw. empfangen wurden.
disk	Gibt an, wieviele Daten auf Disks geschrieben bzw. gelesen wurden.
poolinfo	Gibt Informationen über den geteilten Prozessor-Pool des CEC wieder.
mempool	Stellt Informationen zu den Hauptspeicher-Pools der logischen Partitonen und dem physikalischen Host dar.
adapter	Gibt an, wieviele Daten zu den Adaptern der logischen Partition geschrieben bzw. gesendet wurden.
vadapter	Gibt Daten zu den virtuellen Adaptern wieder.
vios	Gibt Informationen zu dem Durchsatz der Virtual-I/O-(VIO)-Server.
vios_adapter	Gibt Informationen zu den Adaptern des Virtual-I/O-Servers wieder.

Tabelle 3-12: Leistungsdaten-Berichte des Betriebssystem-Hilfsmittels topasout
Quelle: eigene Darstellung

Damit die mit „topasrec“ bzw. „topasout“ geschriebenen Leistungsdaten ohne größeren Aufwand grafisch dargestellt werden können, wird von IBM die Excel-Datei „Topas_cec_analyser_v1.0.xls“ zur Verfügung gestellt (IBM 2011i). Diese liest die mit dem Parameter „-c“ generierte CSV-Datei ein und bereitet die Daten anschließend auf.

Die mit den Aufzeichnungstool verbrauchten Systemressourcen würden an sich die Ergebnisse der Messung verfälschen, allerdings werden zum einen dieses oder ähnliche Betriebssystemmittel auch im operativen Betrieb eingesetzt, zum anderen kann der Ressourcenverbrauch der Messwerkzeuge als sehr gering bis unerheblich eingestuft werden (der CPU-Verbrauch im Testsystem schwankte je nach Konfiguration zwischen 0,1 und 0,2 Prozent).

In den folgenden Kapiteln werden spezielle Hilfsmittel zur Aufzeichnung der einzelnen Systemressourcen aufgeführt, da diese im Vergleich zu „Topas“ zusätzliche Leistungsdaten liefern.

3.3.1 Monitoring der CPU

Für die Performance-Analyse ist es wichtig, dass bei den Aufzeichnungen der CPU-Auslastung zwischen Benutzerprozessen, Systemprozessen, I/O-Wartezeit und Leerlauf unterschieden wird. Formal errechnet sich somit die Gesamtzeit wie folgt:

$$t_{gesamt} = \sum_{k=1}^{n} t_{Benutzer} + \sum_{l=1}^{m} t_{System} + t_{I/O} + t_{Leerlauf}$$

Das Betriebssystem-Hilfsmittel „mpstat“ (IBM 2011f) liefert neben weiteren Daten diese Unterscheidung der Prozessordaten:

```
bash-4.1# mpstat 1 1 > mp.out
bash-4.1# cat mp.out

System configuration: lcpu=12 ent=0.3 mode=Uncapped

cpu  min  maj  mpc  int   cs  ics   rq  mig lpa sysc us sy wa id   pc  %ec  lcs
  0   33    0    0  812 1616  508    1    1 100 2122 37 57  0  6 0.04 14.5  629
  1    0    0    0   53  184  153    0    0 100   27  3  6  0 90 0.01  3.7   55
  2    0    0    0   20    0    0    0    0   -    0  0  1  0 99 0.01  3.3   21
  3    0    0    0   28    0    0    0    0   -    0  0  2  0 98 0.01  3.4   29
  4    0    0    0   59    0    0    0    0   -    0  0 64  0 36 0.00  0.2   59
 11    0    0    0   33    0    0    0    0   -    0  0 58  0 42 0.00  0.1   33
  U    -    -    -    -    -    -    -    -   -    -  -  -  0 75 0.22 74.7    -
ALL   33    0    0 1005 1800  661    1    1 100 2149  5  9  0 86 0.08 25.3  826
bash-4.1#
```

Abbildung 3-26: Exemplarische mpstat-Ausgabe
Quelle: eigene Darstellung

Wie man in Abbildung 3-26 sehen kann, werden die Prozessorzeiten in

- „us“ für die Benutzerprozesse,
- „sy“ für die Systemprozesse,
- „wa“ für die I/O-Wartezeiten und
- „id“ für die Leerlaufzeiten

unterschieden. Die Summe dieser Werte muss auf die Gesamtzeit und somit 100 Prozent kommen.

Insgesamt baut sich der verwendete „mpstat“-Befehl aus folgenden Paramatern zusammen:

mpstat <Intervall> <Anzahl> > <Ausgabedatei>

Ebenso wie bei „topas“ berechnet sich die Gesamtlaufzeit aus der Anzahl der zu erhebenden Stichproben multipliziert mit dem angegebenen Intervall.

Der Einfluss von „mpstat“ auf das Testsystem ist ebenso sehr gering. In den verschiedenen Testsystem-Konfigurationen schwankte der CPU-Ressourcen-Bedarf bei einem Intervall von 1 Sekunde zwischen 0,05 und 0,1 Prozent. Auch hier gilt die Aussage, dass die Überwachung

der Prozessorauslastung im Regelbetrieb durchgeführt wird und somit ein praxisgetreues Szenario nicht verfälscht.

Für die Aufzeichnung der Daten wurden die Daten anstelle des Standard-Ausgabe-Streams in eine Datei umgeleitet und anschließend in Excel importiert. Aufgrund der Blockstruktur (gefüllt mit Leerzeichen) konnten die einzelnen Spalten getrennt eingelesen und anschließend grafisch dargestellt werden.

3.3.2 Monitoring des Hauptspeichers

Für die Überwachung des Hauptspeichers bietet „vmstat“ (IBM 2011m) die Möglichkeit, zusätzlich Daten zur „Active Memory Expansion“ (AME) (IBM 2011a) zu erhalten. Active-Memory-Expansion ist eine von IBM entwickelte Technologie für IBM-Power7-Systeme, die Hauptspeicherdaten komprimiert und somit eine größere Hauptspeicherkapazität anbieten kann, als physisch vorhanden ist (siehe Abbildung 3-27).

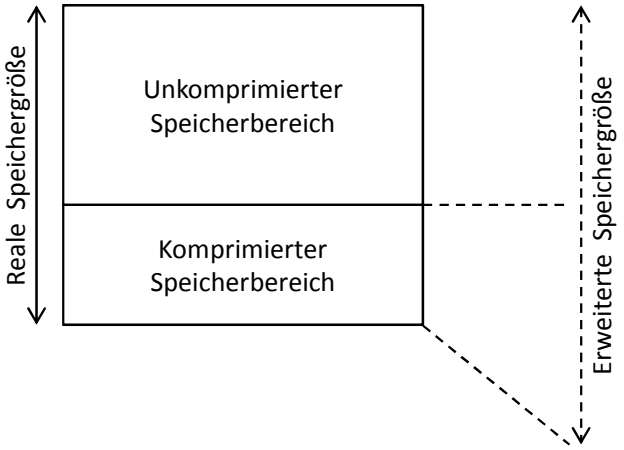


Abbildung 3-27: Funktionsprinzip der Active Memory Expansion
Quelle: eigene Darstellung

Die Kompressionstechnologie wird vollständig vom System verwaltet und ist für die laufenden Applikationen transparent. Sie wird am SAP UCC der TU München auch in der produktiven Systemlandschaft eingesetzt und aufgrund des Praxisbezugs des eingesetzten Workloads auch in der in dieser Arbeit eingesetzten Testsystemlandschaft aktiviert.

Der Einfluss der AME auf den CPU-Ressourcenverbrauch wurde in einer Studie von IBM (Michel 2010, 10) unter anderem für ERP-Systeme getestet und zeigte bis zu einer Hauptspeichererweiterung von 73 Prozent keinen nennenswerten bzw. messbaren Einfluss auf die CPU-Leistung. Der Anstieg betrug in diesem Intervall nur 1 Prozent. Es sei an dieser Stelle allerdings erwähnt, dass die CPU-Auslastung ohne AME bei 60 Prozent lag und somit kein Hochlastszenario gewählt wurde.

Damit in „vmstat“ die zusätzlichen AME-Felder angezeigt werden, wurde der Parameter „-c“ verwendet. Dieser bewirkt bei der Ausgabe folgende zusätzlichen Informationen:

- *„csz“*: Derzeitige Größe des komprimierten Speicherbereichs.
- *„cfr“*: Menge an freien Speicherseiten im komprimierten Speicherbereich.

– *„dxm“*: Fehlmenge an erweiterter Speichergröße.

In Abbildung 3-28 ist eine exemplarische „vmstat“-Ausgabe abgebildet. Für eine Beschreibung der einzelnen Werte sei an dieser Stelle an die „vmstat“-Dokumentation (IBM 2011m) verwiesen.

```
bash-4.1# cat vmstat.out

System Configuration: lcpu=12 mem=10496MB tmem=8192MB ent=0.30 mmode=dedicated-E

 kthr                          memory                              page              faults                cpu
------- ------------------------------------------------------ --------------------- ------------------ -----------------------
  r   b        avm        fre        csz        cfr        dxm    ci    co    pi    po    in     sy    cs us sy id wa    pc    ec
  4   1    2318621     314695     124856       8750          0     0     0     0     0    25   2233   903  0  0 99  0  0.00   1.0
bash-4.1#
```

Abbildung 3-28: Exemplarische vmstat-Ausgabe
Quelle: eigene Darstellung

Der Einfluss von „vmstat“ auf den Ressourcenverbrauch verhält sich wie bei „mpstat“ im Bereich zwischen 0,05 und 0,1 Prozent und kann ebenso als Bestandteil eines praxisbezogenen Szenarios gewertet werden.

Ebenso analog zu „mpstat“ können bei „vmstat“ Intervall und Anzahl angegeben werden, sodass sich folgende Befehlszeile für die in dieser Arbeit durchgeführten Messungen ergibt:

vmstat –c <Intervall> <Anzahl> > <Ausgabedatei>

Die in die Datei geschriebenen Aufzeichnungen wurden im Anschluss in Excel importiert und ausgewertet.

3.3.3 Monitoring des I/O-Subsystems

Für die Überwachung und Aufzeichnung des I/O-Subsystems wurde das Betriebssystem-Hilfsmittel „iostat“ (IBM 2011e) verwendet. Bei dem verwendeten Testsystem kam das Storagesystem „XIV“ (IBM 2011d) zum Einsatz, deren logische Festplattenpartitionen über Fibre-Channel-Adapter am virtuellen I/O-Server des IBM-Power7-Hosts angebunden sind und an die logische Partition über virtuelle Fibre-Channel-Adapter weitergereicht werden.

Der Aufruf des Überwachungs-Tools erfolgt ebenfalls mittels Angabe eines Intervalls und der Anzahl der Messungen.

iostat <Intervall> <Anzahl> > <Ausgabedatei>

Eine Beispielausgabe ist in Abbildung 3-29 dargestellt. Die Werte spiegeln eine Momentaufnahme wider, im Gegensatz zum Aufruf von „iostat“ ohne zusätzliche Parameter, wobei die letzten beiden Werte (gelesene und geschriebene KB) einen aufsummierten Wert darstellen.

```
tty:      tin         tout    avg-cpu: % user % sys % idle % iowait physc % entc
          0.0        967.6                50.5  10.2   39.3      0.0   0.3   89.5

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk1           4.0     779.3     14.0          0       389
hdisk2           0.0       0.0      0.0          0         0
hdisk0           0.0       0.0      0.0          0         0
hdisk3           4.0     779.3     14.0          0       389
```

Abbildung 3-29: Exemplarische iostat-Ausgabe
Quelle: eigene Darstellung

Der Prozentwert „%tm_act“ gibt an, wie lange das I/O-System für den betrachteten Zeitraum aktiv war, „Kbps“ stellt die Übertragungsrate in Kilobyte pro Sekunde dar und „tps“ gibt die Anzahl an I/O-Vorgängen pro Sekunde an. Der Ressourcenverbrauch von „iostat“ bewegt sich in einem ähnlichen Rahmen wie die vorangegangenen Überwachungs-Tools und schwankt ebenso zwischen 0,05 und 0,1 Prozent CPU-Zeit.

Die Aufzeichnung und Auswertung der Daten erfolgt analog zu „mpstat“ und „vmstat“. Somit kann die Nutzung der fundamentalen Systemressourcen aus Betriebssystemsicht festgehalten und analysiert werden.

3.4 Zusammenfassung

In diesem Kapitel wurden die notwendigen Informationen und Hilfsmittel für die Messung und Modellierung eines SAP-Portal-Systems dargestellt und beantworten somit die in Forschungsfrage 1 gestellte Frage, wie ein SAP-Netweaver-Portal-System für die Performance-Modellierung und Simulation charakterisiert werden kann und welche Analyseinstrumente die benötigten Informationen bereitstellen.

Im ersten Teil dieses Kapitels wurde die Architektur des SAP-Netweaver-Portal-Systems analysiert. Dabei wurden neben dem Aufbau einer Java-Server-Instanz einzelne Komponenten des Systems betrachtet, die in der Systemdokumentation (bspw. SAP Help 2011) sowie in den Schulungsunterlagen für SAP-Basis-Administratoren (SAP Education 2011) ausführlich beschrieben sind. Über die Analyse dieser Materialien, den theoretischen Grundlagen aus Kapitel 2 und den Erfahrungen am SAP UCC der TU München konnten die Bestandteile der Systemarchitektur identifiziert werden, die das Leistungsverhalten wesentlich beeinflussen.

Im zweiten Teil dieses Kapitels wurde die Monitoring-Architektur des SAP-Netweaver-AS-Java vorgestellt und die Hilfsmittel für die Datenaufzeichnung der einzelnen Komponenten dargelegt. In diesem Zusammenhang wurde ebenso auf die Möglichkeit des Black-Box-Monitorings eingegangen, welches die gesamte Ausführungszeit einer Benutzeranfrage am Serversystem aufzeichnet, sowie die Analyse des Garbage-Collectors mittels bereitgestellter Hilfsmittel vorgestellt.

Der letzte Teil dieses Kapitels beschäftigt sich mit den Hilfsmitteln zur Leistungsanalyse auf Betriebssystemebene. Dabei wurden die drei Performance-Faktoren CPU, Hauptspeicher und I/O behandelt. Während die I/O-Betrachtung aufgrund des Black-Box-Ansatzes des Datenbanksystems eine untergeordnete Rolle in dieser Arbeit spielt und der Hauptspeicher ausreichend dimensioniert wird, ist die Leistungsanalyse der CPU-Ressource auf Betriebssysteme-

bene nicht zuletzt für die Analyse des Garbage-Collector-Verhaltens bzw. dessen Auswirkungen auf die Systemperformance ein sehr wichtiger Bestandteil. Zudem kann das Verhältnis zwischen CPU-Zeit für Verwaltungsaufgaben des Betriebssystems und nutzbarer User-CPU-Zeit betrachtet werden. Vor allem im Hochlastbereich spielt dies eine immer größere Rolle.

4 LQN-Modell

In diesem Kapitel wird die Modellierung der Systemkomponenten und des Workloads sowie deren Parametrisierung beschrieben.

Wie bereits in Kapitel 2.3.3.1 erwähnt, ist einer der drei Gründe für Messungen am System die Datensammlung zur Parametrisierung des Warteschlangenmodells. Ein Lasttest mit mehreren Benutzern simuliert dabei einen bestimmten Workload mit einer spezifizierten Anzahl an simultan agierenden Benutzern. Diese Mehr-Benutzer-Lasttests können gestaffelt durchgeführt werden, indem die Anzahl der Benutzer mit jedem Schritt erhöht wird, sodass das System schrittweise unter einer höheren Last steht. Die einzelnen Messergebnisse können anschließend in ein Diagramm eingetragen werden, um den Verlauf aufzuzeigen. Unter der selbst-evidenten Annahme, dass eine zunehmende Anzahl an simultan agierenden Benutzern die Systemlast ansteigen lässt, kann somit beispielsweise der Anstieg der verwendeten Systemressourcen beobachtet werden. Dieser wird grundsätzlich in einen exponentiellen, linearen und logarithmischen Verlauf kategorisiert, je nachdem, ob die Systemlast zunehmend, gleichbleibend oder abnehmend ansteigt.

Der Hauptzweck von Mehr-Benutzer-Lasttests und Simulationen ist die Analyse der Skalierbarkeit des Systems bei simultan durchgeführten Benutzer-Tasks. Hierbei können sich die durchgeführten Operationen der Benutzer unterscheiden oder lediglich die Anzahl der Benutzer variiert werden, ohne die Art und Reihenfolge der durchgeführten Benutzeranfragen zu verändern. Letzteres wurde für die Fallstudie in dieser Arbeit verwendet, da der Workload den Inhalten einer SAP-Netweaver-Portal-Schulung entspricht, die von einer gewissen Anzahl von Studenten durchgeführt wird, wobei jeder Student dieselben Aufgaben durchführt. Die Tatsache, dass sich die Systemlast nur in einer Dimension verändert (nämlich der Anzahl der simultanen Benutzer), hat zudem den Vorteil, dass die Messergebnisse nicht durch zusätzliche Wechselwirkungen beeinflussen werden, die beispielsweise nur sporadisch auftreten und entsprechend analysiert werden müssten.

Aufgrund der simultanen Anfragen, die am System ankommen, kommt es bei steigender Anzahl an Benutzern zu einer zunehmenden Anzahl von Wartesituationen. Warteschlangennetze (siehe Kapitel 2.4) und im Speziellen die LQNs (siehe Kapitel 2.5) stellen hierbei eine bewährte Methode dar, Warteschlangensituationen zu modellieren. Mittels analytischer Verfahren oder der in dieser Arbeit angewandten Simulation können anschließend ein bestimmtes Szenario berechnet und die Leistungsdaten ermittelt werden.

Die sukzessiv durchgeführten Benutzeranfragen (als Teil des modellierten Workloads) werden nicht direkt aufeinanderfolgend, sondern erst nach einer spezifizierten, konstanten Denkzeit des Benutzers gestartet. Eine einzelne Benutzeranfrage wird auch als Interaktionsschritt bezeichnet. Die Summe der einzelnen Interaktionsschritte ergibt den gesamten modellierten Workload, wie zum Beispiel einen abgebildeten Geschäftsprozess oder in diesem Falle die abgebildeten Schulungsinhalte.

Die wesentlichen Erweiterungen zu bereits bestehenden Ansätzen spiegeln sich vor allem in der Modellierung des Sperrmechanismus eines SAP-Netweaver-Java-Systems und der Betrachtung des Garbage-Collectors wieder. Sowohl im Bereich des Java-Speichermanagements

(siehe zum Beispiel Franks/Lau/Hrischuk (2011), Ufimtsev (2006), Ufimtsev/Murphy (2006) oder Xu et al. (2005)), als auch bei der Behandlung des SAP-Sperrmanagements (siehe zum Beispiel Rolia et al. (2009) oder (Cheng 2008)) konnten keine expliziten Ansätze in der Literatur identifiziert werden. Die Einflüsse dieser Komponenten finden sich stets implizit in den Bearbeitungszeiten wieder.

In Zusammenarbeit mit Gradl (2012) wurde aus diesem Grunde ein Konzept zur Modellierung des SAP-Sperrmechanismus entwickelt, welches sowohl auf den ABAP- als auch auf den Java-Stack anwendbar ist und in Kapitel 4.2.3 näher erläutert wird. Damit wird es möglich, das Verhalten des Sperrmechanismus nachzubilden. Bei der impliziten Parametrisierung der Sperrwartezeiten in den Bearbeitungszeiten könnten diese nicht dynamisch, abhängig von der parallelen Nutzung des Systems, mit einbezogen werden.

4.1 Grundannahmen

Zunächst werden in Kapitel 4.1.1 Annahmen bezüglich des Systemverhaltens getroffen, die sich aus der Warteschlangentheorie ableiten lassen. Dies stellt eine Voraussetzung für das in dieser Arbeit entwickelte Warteschlangenmodell dar, da sich nur bei entsprechendem Systemverhalten der Einsatz eines Warteschlangennetzes anbietet. Wie die im weiteren Verlauf dieser Arbeit dargestellten Messergebnisse zeigen werden, erfüllt ein SAP-Portalsystem diese Voraussetzungen weitestgehend.

In Kapitel 4.1.2 werden die betrachteten Einflussgrößen anhand der in der Literatur beschriebenen Hauptfaktoren sowie der vorhandenen bzw. messbaren Leistungsdaten eingegrenzt. Dies ist notwendig, da die Modellierung aller Einflussgrößen, falls überhaupt definierbar, nicht praktikabel ist.

4.1.1 Systemverhalten

Betrachtet man die CPU-Zeit für einen einzelnen Interaktionsschritt, die CPU-Nutzlast und die Antwortzeiten der wegen der zunehmenden Anzahl von simultanen Benutzern ansteigenden Systemlast, ergeben sich folgende Prämissen, die sich direkt aus der Warteschlangentheorie ableiten:

1. Die CPU-Zeit für einen bestimmten Interaktionsschritt eines Benutzers ist gleichbleibend und somit unabhängig von der Anzahl der simultan agierenden Benutzer.

   Mit anderen Worten bleibt die reine Bearbeitungszeit konstant und ist somit unabhängig von der Systemauslastung. Würde die CPU-Zeit für einen bestimmten Interaktionsschritt zunehmen, beispielsweise aufgrund von Busy-Waiting-Situationen (z.B. Dijkstra 1971), würde die gesamte CPU-Nutzlast übermäßig ansteigen und somit zu einer stark erhöhten Antwortzeitkurve führen.

   In Abbildung 4-1 ist dieser Sachverhalt symbolisch dargestellt. Würde bei steigender CPU-Last (y-Achse), hervorgerufen durch eine steigende Anzahl von Benutzern (x-Achse), die reine CPU-Zeit für einen definierten Interaktionsschritt ansteigen, wäre diese von äußeren Systembedingungen abhängig. Diese Abhängigkeit würde von dem Warteschlangenmodell nicht erfasst werden, da sich eine konstante Netto-Bearbeitungszeit, unabhängig von der Systemlast, direkt von den Grundannahmen der

Warteschlangentheorie ableitet. Mit anderen Worten bleibt in einem Warteschlangensystem die Zeit, in der sich ein Element in einer Bedieneinheit aufhält, konstant und ist nicht von äußeren Einflüssen abhängig. Lediglich die Wartezeit, bevor eine Bedieneinheit frei wird und das Element prozessieren kann, erhöht die Gesamtabarbeitungszeit (vgl. Kapitel 2.4).

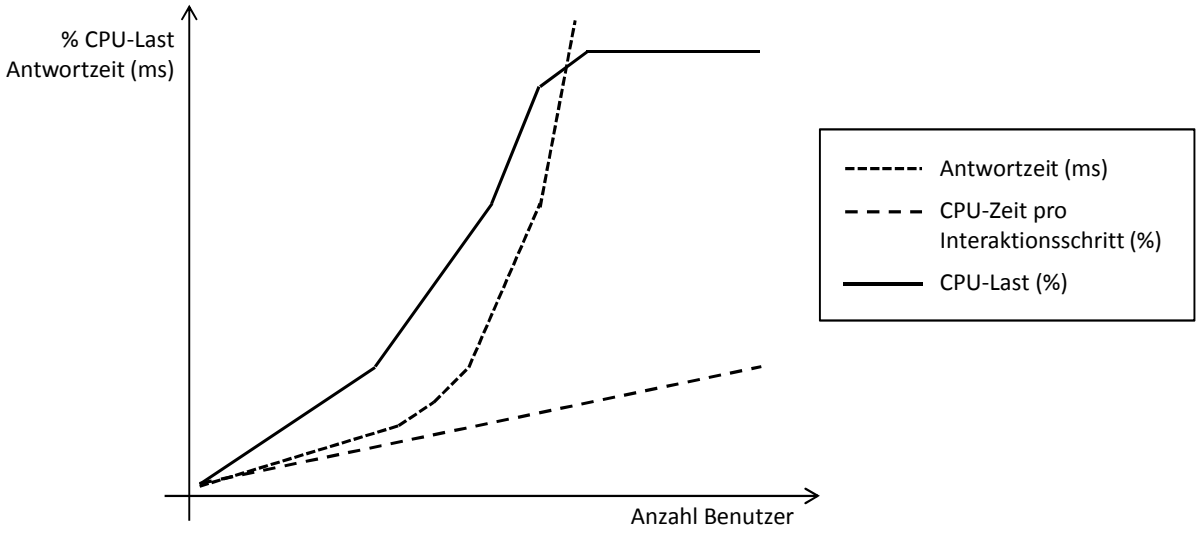


Abbildung 4-1: Systemverhalten bei zunehmender CPU-Zeit pro Interaktionsschritt
Quelle: eigene Darstellung

2. Die Nutzlast der CPU steigt linear zur Anzahl der simultan agierenden Benutzer an.

   Die CPU-Nutzlast steigt nur dann linear zur Anzahl der simultanen Benutzer an, wenn das System die CPU-Ressourcen verwenden kann und nicht aufgrund von anderweitigen Einflüssen derart beschäftigt ist, dass nur ein Teil der vorhandenen CPU-Ressourcen genutzt wird. Die Antwortzeit würde auch in diesem Falle übermäßig schnell ansteigen. Sollte das System ab einer gewissen Auslastung (und somit ab einer gewissen Anzahl von simultanen Benutzern) keine weiteren (vorhandenen) CPU-Ressourcen nutzen, würde die Simulation des Warteschlangenmodells ab diesem Zeitpunkt falsche Ergebnisse liefern, da der CPU im LQN-Modell weiterhin Ressourcen zur Verfügung stehen.

3. Das Antwortzeitverhalten folgt der Annahme aus dem Warteschlangenmodell.

   Wie bereits im Kapitel über Warteschlangennetze dargestellt, beschreibt in einem Warteschlangenmodell eine mathematische Verteilung das Ankunftszeitverhalten sowie das Bedienzeitverhalten. Aus dem Markov-Ketten-Modell $M/M/n$ (siehe Kendall-Notation in Kapitel 2.4), welches zum Beispiel für symmetrische Multiprozessor-Systeme verwendet wird, ergibt sich ein Antwortzeitverhalten, das zuerst linear und ab einem gewissen Punkt exponentiell ansteigt. Der exponentielle Anstieg ist dabei meist durch starke Wartesituationen bedingt.

Aus den drei soeben erläuterten Annahmen ergibt sich somit folgendes Systemverhalten: Die CPU-Zeit pro Interaktionsschritt bleibt konstant und ist unabhängig von der Anzahl simultaner Benutzer und somit von der Systemauslastung (Annahme 1). Die CPU-Nutzlast steigt linear mit der Anzahl an Benutzern an und kann bei entsprechender Auslastung 100% erreichen (Annahme 2). Das Antwortzeitverhalten setzt sich aus einer linear ansteigenden Phase und einem exponentiellen Anstieg zusammen (Annahme 3). In Abbildung 4-2 werden diese Annahmen bildlich zusammengefasst. Während die CPU-Zeit pro Interaktionsschritt gleichbleibend und somit von äußeren Systembedingungen unabhängig ist, steigen sowohl CPU-Last (bis zu 100%) als auch die Antwortzeit bei einer zunehmenden Anzahl von Benutzern. Der exponentielle Verlauf der Antwortzeitenkurve ab einer bestimmten Systemlast wird durch das verstärkte Eintreten von Wartesituationen erwartet.

Nur dann, wenn das Systemverhalten des SAP-Portals diesen Annahmen folgt, kann das propagierte Warteschlangenmodell sinnvoll eingesetzt werden. Cheng (2008) zeigt in seinen Ausführungen, dass bei einem SAP-Netweaver-Java-basierten System (und somit bei einem SAP-Portalsystem) dieses Verhalten erwartet werden kann. Wie im weiteren Verlauf dieser Arbeit noch dargestellt wird, bestätigen die durchgeführten Messungen diese Annahmen größtenteils. Erst im Volllastbereich treten äußere Systemeinflüsse auf (hauptsächlich durch die Speichermanagement-Mechanismen der Java-Virtual-Machine), die von dem Warteschlangenmodell nicht erfasst bzw. nur statisch parametrisiert werden können. Die CPU-Nutzlast kann bei vollem Hauptspeicher aufgrund der Speichermanagement-Aktivitäten keine vollständige Auslastung erreichen.

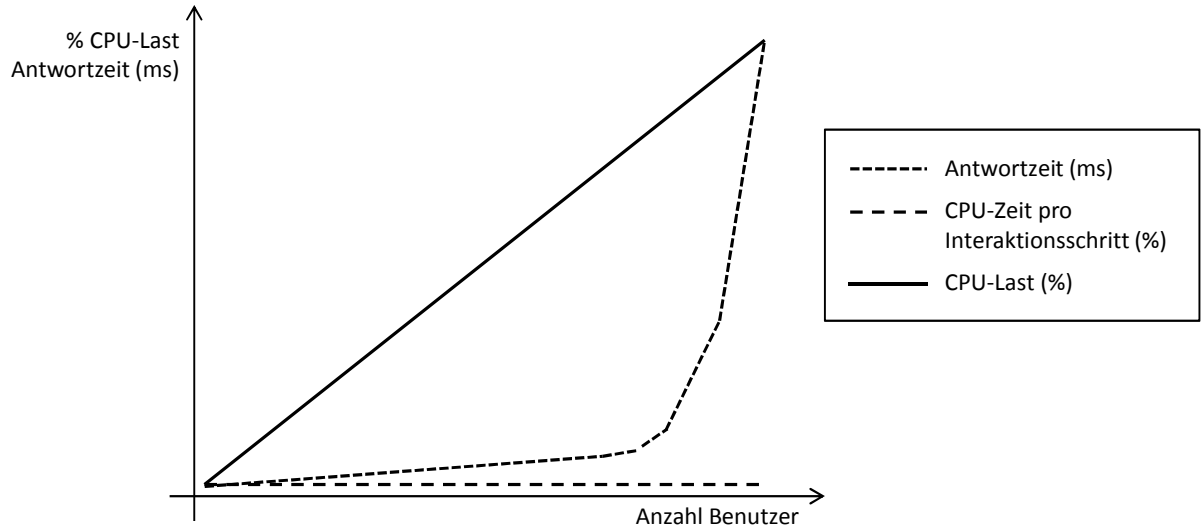


Abbildung 4-2: Erwartetes Systemverhalten nach der Warteschlangentheorie
Quelle: eigene Darstellung

4.1.2 Einflussgrößen

Die Frage der Einflussgrößen sowie Anforderungen an das LQN-Modell geht mit der Frage nach den zu modellierenden Komponenten einher. Da bei einem komplexen Portalsystem nicht sämtliche Einflussgrößen auf die Antwortzeiten von Benutzeranfragen erfasst und abgebildet werden können, wird zur Modellbildung die Auswahl sowie Granularität der Komponenten nach folgenden Hauptkriterien getroffen:

1. Ist die Architektur der Komponente bekannt?
2. Sind Performance-Daten für die zu modellierende Komponente verfügbar?
3. Wie groß ist der Einfluss auf die Gesamtleistung des Systems?

Zur Beantwortung der Fragen wurde die Literatur (siehe Kapitel 2) sowie die Systemdokumentation (siehe Kapitel 3) eingehend analysiert. Die Identifikation der Objekte sowie der Einflussfaktoren dient als Grundlage für die Modellierung und Parametrisierung der Systemkomponenten. Tabelle 4-1 zeigt eine Übersicht der Auswertung.

Objekt	**Architektur (Kapitel)**	**Monitor (Kapitel)**	**Einflussfaktoren**
Client	(3.1.1)	n/a	- Art der Anfragen - Anzahl - Denkzeit
Lastschritt (Task)	3.1.1 3.1.2	3.2.2 3.2.3 (3.3)	- Dispatching-Zeit - Wartezeit - Bearbeitungszeit - Abbrüche (durch Timeouts) - Datenbankaufrufe (falls zutreffend)
Datenbank (-Aufrufe)	wird als Black-Box betrachtet	3.2.4 (3.3)	- Bearbeitungszeit - Sperrwartezeiten (falls zutreffend) - Tabellenpuffer (falls zutreffend)
Tabellenpuffer	3.1.4	3.1.4	- Verdrängungen - Invalidierungen
Sperrobjekte	3.1.3	3.1.3	- Wartezeit beim Enqueue-Server - Wartezeit bei blockierten Elementen
Garbage-Collector (GC)	3.1.5	3.2.6 3.3	- GC-Dauer - GC-Intervall

Tabelle 4-1: Einflussgrößen auf die Antwortzeit in einem SAP-Netweaver-Portal-System
Quelle: eigene Darstellung

4.2 Modellierung und Parametrisierung der Systemkomponenten

Nachdem die theoretischen Grundlagen erarbeitet und die Dokumentation zur Systemarchitektur sowie verfügbare Monitore analysiert wurden, konnten die einzelnen Komponenten, die das Verhalten des Gesamtsystems beschreiben, modelliert werden. In einer ersten Iteration wurden folgende Elemente identifiziert:

- Benutzertyp (abhängig von den durchgeführten Anfragen sowie der Denkzeit)
- Lastschritt (somit die Anfrage bzw. der Task selbst)
- Sperr-Objekte (Sperrtabellen-Verwaltung im Enqueue-Server)
- Tabellenpufferung (Pufferung von Daten im Hauptspeicher)
- JDBC-Call (Durchführung eines Datenbankzugriffs)

- Datenbank (wird als Black-Box betrachtet)

In einer zweiten Iteration wurde das Verhalten des Garbage-Collectors mittels einer logischen Garbage-Collector-Komponente modelliert. Wie in Kapitel 6 noch gezeigt wird, nimmt dieser im Hochlastbereich einen erheblichen Einfluss auf das Leistungsverhalten des Systems. Durch das Hinzunehmen der GC-Parameter im Modell konnte der Einfluss erhöhter GC-Aktivitäten auf die Antwortzeit getestet werden.

Nachstehend werden die einzelnen Komponenten des LQN-Modells vorgestellt und dabei auf die Anforderungen, Umsetzung und Parametrisierung eingegangen.

4.2.1 Benutzer

Wie bereits in der Einleitung zu Kapitel 4 erwähnt, basiert der in dieser Arbeit verwendete Workload auf einer Portalfallstudie des SAP UCC der TU München (SAP UCC TUM 2012). Die Portalfallstudie wird von Dozenten der angeschlossenen Institutionen verwendet, um den Studenten den Umgang mit dem Portal näher zu bringen. Dabei führt jeder Student eine Reihe von Aktionen aus, deren Art und Umfang stets identisch ist. Somit wird die Lastintensität in der Fallstudie lediglich durch die Anzahl der Benutzer gesteuert.

Da den Benutzern allerdings verschiedene Denkzeiten zugewiesen werden sollen, damit sich der Ausführungszeitpunkt der einzelnen Lastschritte unterscheidet, muss ein generischer Ansatz gewählt werden, bei dem verschiedene Benutzertypen definiert werden können. Dadurch können neben einer variierenden Denkzeit auch unterschiedliche Interaktionsschritte für jeden Benutzertyp definiert werden, wodurch das Modellierungskonzept den Ansprüchen der Verallgemeinerbarkeit genügt.

Lastintensität

Die Lastintensität im allgemeinen Fall lässt sich wie folgt beschreiben: Ein Benutzertyp BT kann durch folgendes 3-Tupel dargestellt werden:

$$BT = (I, V, D)$$

Folgende Aussagen gelten:

- I ist eine endliche, nichtleere Menge von Interaktionsschritten.
- Die Flussrelation V definiert die durchgeführten Interaktionsschritte mit $V \subseteq (I \times I)$.
- Die Denkzeiten D (in Sekunden) sind Funktionen $D: I \rightarrow \mathbb{N}_0$

Einem Benutzertyp wird zudem bei der Modellierung eines Workload-Szenarios eine Multiplizität m zugewiesen, die die Anzahl der Benutzer pro Benutzertyp festlegt.

Die Summe aller Benutzertypen $B_1 - B_n$ unter Berücksichtigung ihrer Multiplizität m repräsentiert die modellierte Lastintensität L:

$$L = \sum_{i=1}^{n} BT_i \cdot m$$

LQN-Modellierung

Bei den LQNs werden Benutzer als Task dargestellt. Da ein Benutzertask jedoch immer einen Ausgangspunkt darstellt (und somit keine Anfrage erhalten, sondern lediglich senden kann), wird von einem Referenz-Task gesprochen. In geschlossenen Warteschlangensystemen werden von den Referenz-Tasks synchrone Anfragen gesendet und somit die Last am System erzeugt. Bei offenen Warteschlangensystemen werden asynchrone Anfragen gesendet, deren Ankunftsrate über einen entsprechenden Parameter definiert wird. Da Referenz-Tasks in LQN-Modellen Benutzer darstellen sollen, definiert ein eigens dafür eingerichtete Parameter die Denkzeit des Benutzers.

Umsetzung

Wie bereits erwähnt, führt ein Benutzer eine Reihe von Interaktionsschritten aus, deren Abfolge durch Denkzeiten zwischen den einzelnen Schritten gebremst wird. Jeder durchgeführte Interaktionsschritt $i_1 - i_n$ entspricht dabei einer Aktivität. Die Anzahl der Benutzer, die diese Folge von Interaktionsschritten durchführt, wird über die Multiplizität des Referenz-Tasks angegeben. Diese Modellierungsweise reicht im Prinzip aus, um die in Kapitel 5 durchgeführte Fallstudie zu modellieren, da sämtliche Studenten (Benutzer) dieselben Aktionen durchführen und lediglich die Anzahl der Studenten über die Multiplizität des Referenz-Tasks parametrisiert werden muss. Allerdings sollen, wie bereits erwähnt, die Denkzeiten der einzelnen Interaktionsschritte keinen Absolutwert darstellen. Daher ist es notwendig, verschiedene Benutzertypen zu modellieren, indem mehrere Referenz-Tasks mit den entsprechenden Interaktionsschritten definiert werden. Die Denkzeiten der einzelnen Interaktionsschritte werden anschließend exponentialverteilt parametrisiert. Es sei an dieser Stelle vorgegriffen, dass aufgrund der ausgegebenen Simulator-Resultate der Verzicht auf eine Multiplizität größer eins und die Modellierung jedes Benutzers als eigenen Task von Vorteil ist.

In Bezug auf den Ressourcenbedarf konzentriert sich die in dieser Arbeit durchgeführte Performance-Analyse auf die Applikationsschicht, daher wird bei den Clients von ausreichend zur Verfügung stehenden Ressourcen ausgegangen und infolgedessen der CPU-Bedarf vernachlässigt. Dies wird durch die Modellierung einer Client-CPU-Ressource mit unendlicher Multiplizität erreicht.

In der Summe ergibt sich das in Abbildung 4-3 dargestellte Konzept der Benutzermodellierung.

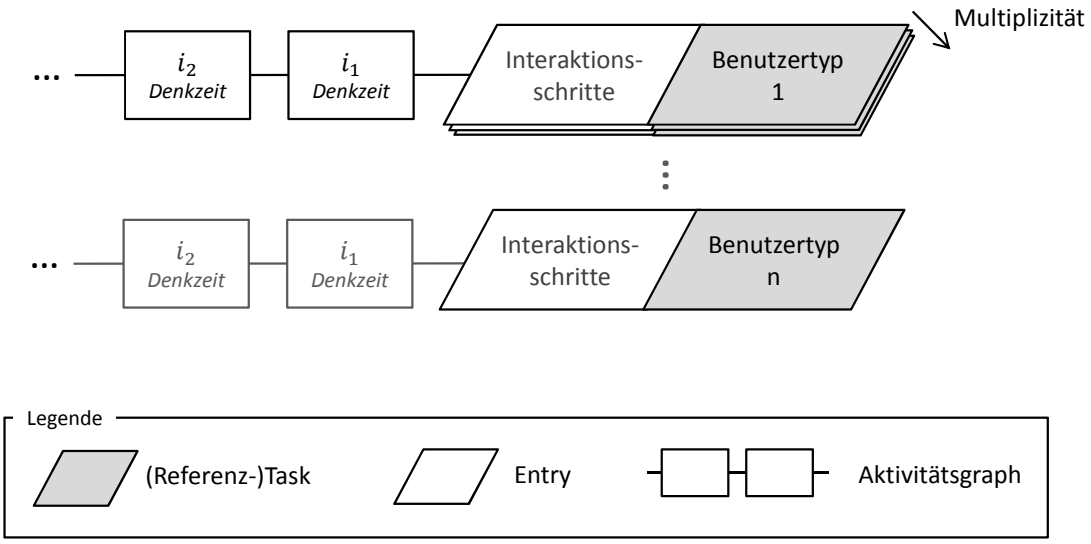


Abbildung 4-3: LQN-Modellierung der Benutzer
Quelle: eigene Darstellung

Parametrisierung

Wie bereits geschildert, werden die Benutzer als Referenz-Tasks und die einzelnen Benutzeranfragen (Interaktionsschritte) als Aktivitäten modelliert. Folgende Parameter, die sich auf die traditionelle Schreibweise der Eingabedatei für den LQNs bzw. LQsim beziehen (siehe Kapitel 2.5.4 sowie Franks et al. (2012)), definieren die Eigenschaften der Benutzer:

- Die Anzahl der Benutzer desselben Benutzertyps kann über die Multiplizität des Tasks, die über den Parameter <*multi_server_flag*> gesteuert wird, festgelegt werden.

- Die Denkzeit des Benutzers wird mit dem Parameter <*think_time*> definiert. Dieser wird für jeden Interaktionsschritt definiert, da unterschiedliche Benutzeraktionen unterschiedliche Denkzeiten erfordern. Im Normalfall sollten die Denkzeiten der einzelnen Interaktionsschritte exponentialverteilt werden.

- Da der Ressourcenbedarf bei den Benutzern keinen Einfluss nehmen soll, werden die gekoppelten CPU-Ressourcen mit einer unendlichen Multiplizität modelliert. Dazu wird der Parameter <*multi_server_flag*> des Prozessor-Tasks auf „infinite“ gesetzt.

- Die Service-Anfragen (Requests, siehe Kapitel 2.5.1) verbinden die Interaktionsschritte des Benutzers mit den entsprechenden Lastschritten am Server. Die durch den Benutzer getätigten Aufrufe der Lastschritte sind synchron, da bis zur Erledigung der Aufgabe gewartet und erst im Anschluss der nächste Interaktionsschritt eingeleitet wird. Die Anzahl der synchronen Anfragen wird über den Parameter <*rendezvous*> angegeben. Da im Normalfall jeder Lastschritt einmal ausgeführt wird, wird der Parameter auf den deterministischen Wert 1 gesetzt.

4.2.2 Lastschritt

Die Abarbeitung einer Benutzeranfrage (siehe Kapitel 3.1.1) soll aus der Sicht des Servers als Lastschritt bezeichnet werden. Während der Abarbeitung des Programmcodes kann ein Last-

schritt lesend oder schreibend auf die Datenbank zugreifen. Falls die Daten im Tabellenpuffer vorgehalten werden, kann ein vergleichsweise langsamer Zugriff auf die Datenbank umgangen werden (siehe Kapitel 3.1.4). Bei einem Datenbankzugriff können zudem Sperren gesetzt werden, die über den Enqueue-Server verwaltet werden (siehe Kapitel 3.1.3).

LQN-Modellierung

Ein LQN-Task repräsentiert, wie in Kapitel 2.5.1 dargelegt, Ressourcen eines Systems, beispielsweise Systemprozesse, Benutzer, Hardware-Einheiten oder Lastschritte.

Ein LQN-Task hat eine eigene Warteschlange für die eingehenden Anfragen und ist mit einem Prozessor-Task (der CPU-Ressource) verbunden. Die Abarbeitung der Anfragen kann nach den folgenden Scheduling-Mechanismen erfolgen:

- FIFO (*First In, First Out*): Der Standardmechanismus, bei dem die Anfragen in der Reihenfolge abgearbeitet werden, wie sie eingetroffen sind.

- PPR (*Priority, Preemptive Resume*): Ein prioritätsbasierter Mechanismus, bei dem Anfragen mit einer höheren Priorität sofort abgearbeitet werden, sobald sie bei dem Task ankommen. Prioritäten werden in den Entries mit einer Reichweichte zwischen 0 und $+\infty$ angegeben, wobei der Standardwert 0 beträgt.

- HOL (*Head-of-Line Priority*): Ebenfalls ein prioritätsbasierter Mechanismus, bei dem allerdings die aktuell ausgeführte Anfrage nicht unterbrochen wird und erst nach Beendigung die Anfrage mit erhöhter Priorität abgearbeitet wird.

Die einzelnen Entries des Tasks, die in Kapitel 2.5.1 beschrieben wurden, stellen die Operationen dar, die von dem Task durchgeführt werden können. Hinsichtlich der eingehenden und ausgehenden Aufrufe wird unterschieden zwischen synchronen, asynchronen und weitergeleiteten Aufrufen. Synchrone Anfragen werden auch als *Rendezvous* bezeichnet. Zudem senden Entries die Antworten bei einem eingegangenen, synchronen Aufruf. Die Parameter einer Entry können entweder lediglich über Phasen oder über zusätzliche Aktivitäten definiert werden.

Es wird zwischen drei Phasen unterschieden, wobei die erste Phase die Service-Phase darstellt, die mit dem bereitgestellten Dienst einer Bedienstation in einem Warteschlangennetz verglichen werden kann. Phase 2 und 3 sind autonome Phasen, die nach dem Senden der Antwort durchlaufen werden.

Aktivitäten werden dann eingesetzt, wenn das interne Verhalten einer Entry ein komplexeres Muster aufweist, beispielsweise die bei dem Benutzertask vorgestellte Sequenz von Interaktionsschritten. Sie formen einen gerichteten Graphen, der Gabelungen, Vereinigungen sowie Schleifen beinhalten kann, und können somit komplexere Abarbeitungsschemas abbilden. Sie können ebenso Anfragen und Antworten an andere Entries bzw. Aktivitäten senden.

Die Anzahl der Anfragen, die an weitere Server (niedrigerer Ebenen) gesendet werden, können mit einem stochastischen oder deterministischen Wert angegeben werden. Bei einer deterministischen Angabe werden genauso viele Anfragen gesendet, wie angegeben. Bei einer

stochastischen Angabe wird der angegebene Wert als Durchschnitt interpretiert und eine geometrische Verteilung angenommen.

Die Bedienzeit wird anhand des Mittelwertes und der Streuung angegeben. Dabei wird eine Anfrage an den verbundenen Prozessor-Task gesendet. Der quadrierte Variationskoeffizient (C_X^2) wird zur Beschreibung des Streuungsmaßes verwendet und folgt direkt aus dem Quadrieren des Variationskoeffizienten (siehe Kapitel 2.2.1.2):

$$C_X^2 = \frac{\sigma^2}{\mu^2}$$

Für die Generierung der Zufallsvariablen verwendet der Simulator folgende Verteilungen:

- $C_X^2 = 0$: deterministisch.
- $0 < C_X^2 < 1$: Gammaverteilung.
- $C_X^2 = 1$: Exponentialverteilung.

Mit der soeben beschriebenen Task-Komponente sowie ihren zugehörigen Entries bzw. Aktivitäten können die Lastschritte grundsätzlich modelliert werden.

Allerdings ist es in vielen Systemen so, dass mehrere Kopien eines Tasks vorhanden sind, die den bereitgestellten Dienst parallel abarbeiten können. Dazu werden beispielsweise in einem SAP-Netweaver-Portal-System mehrere Server-Instanzen konfiguriert, die jeweils wiederum mehrere Threads aufweisen. Bei der Modellierung dieser Parallelität stehen zwei Varianten zur Verfügung: Multiplizität und Replizierung.

Eine Multiplizität größer eins kann mit einem Server verglichen werden, der mehrere parallele Threads ausführt. Replizierung hingegen ist eine weitere Instanz des Servers. Der Hauptunterschied liegt dabei darin, dass bei einem Server-Task mit einer Multiplizität größer eins ein und dieselbe Warteschlange geteilt wird, wobei bei mehreren Server-Repliken jede Replik ihre eigene Warteschlange bereitstellt (siehe Abbildung 4-4).

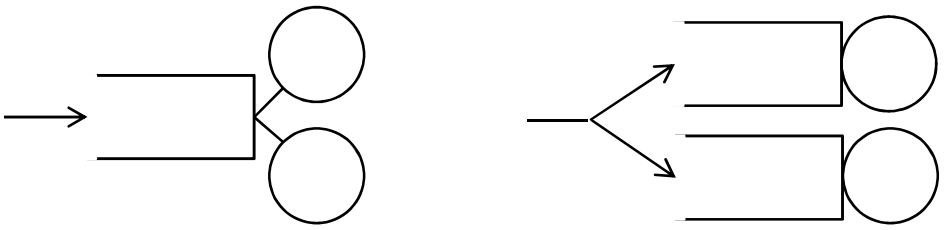


Abbildung 4-4: Multiplizität und Replizierung
Quelle: eigene Darstellung nach Franks et al. (2012)

Mit den Server-Repliken können somit mehrere Server-Instanzen, mit der Multiplizität multiple Threads abgebildet werden. Die Anwendung von Server-Repliken reicht jedoch nur dann aus, wenn die einzelnen Server-Instanzen echte Kopien sind: sowohl die unterstützten Opera-

tionen (Entries) als auch die Prozessoren müssen identisch sein. Sollten verschiedene Server-Instanzen Unterschiede aufweisen, müssen sie explizit modelliert werden und eine Dispatcher-Einheit vorgeschalten werden, die die Anfragen auf die jeweiligen Server-Instanzen verteilt.

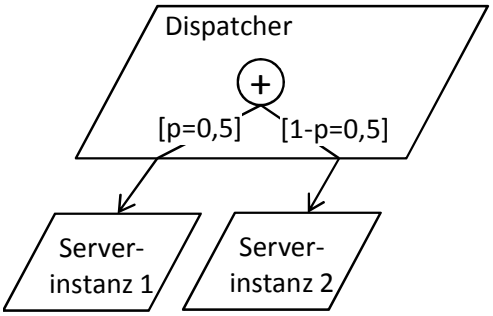


Abbildung 4-5: Vorgeschaltete Dispatcher-Einheit zum Verteilen der Anfragen
Quelle: eigene Darstellung

Umsetzung

Wie in Kapitel 3.1 beschrieben, besteht ein SAP-Netweaver-AS-Java aus einem Java-Dispatcher und ein oder mehreren Java-Server-Instanzen, die wiederum multiple Threads aufweisen. Standardmäßig sind für jeden Java-Server 40 Threads konfiguriert, die allerdings über einen entsprechenden Parameter konfiguriert werden können.

Nach Rolia et al. (2009) kann die explizite Modellierung des Dispatchers eines rein auf dem ABAP-Stack basierenden SAP-ERP-Systems vernachlässigt werden, da der Dispatcher keinen nennenswerten Einfluss auf die Systemperformance nimmt. Aufgrund der engen Architekturverwandtschaft des im Java-Stack eingesetzten Dispatchers wird auch in der vorliegenden Arbeit von dieser Annahme ausgegangen. Zudem ist die Abbildung des Verteilungsmechanismus nur über eine Annäherung möglich, da keine komplexen Verteilungsalgorithmen im LQN-Model integriert werden können.

Die multiplen Threads einer Java-Server-Instanz werden über die Multiplizität modelliert, da sie dieselbe Warteschlange innerhalb des Java-Server-Prozesses teilen. Mehrere Java-Server-Instanzen können über die Server-Repliken modelliert werden, solange sie dieselben Charakteristiken aufweisen. Allerdings ist auch in diesem Fall die Verteilungscharakteristik zu den einzelnen Warteschlangen der Repliken nur eine Annäherung an den effizienten Verteilungsalgorithmus des Java-Dispatchers.

Aufgrund der genannten Einschränkungen sollte eine explizite Modellierung des Dispatchers nur dann durchgeführt werden, wenn mehrere Hosts mit unterschiedlichen CPU-Ressourcen modelliert und anschließend einzeln analysiert werden sollen. Ansonsten eignet sich die Zusammenführung der einzelnen Server-Instanzen $SI_1 - SI_n$ zu einer einzigen logischen Server-Instanz, deren Multiplizität m die Gesamtsumme aller zur Verfügung stehenden Server-Threads darstellt:

$$m = \sum_{k=1}^{n} Anzahl\ Threads(SI_k)$$

Die einzelnen Lastschritte werden in den einzelnen Entries definiert. Dabei muss jede durchgeführte Operation des Workloads in jedem Task, sofern mehrere Tasks modelliert wurden, definiert sein. Damit die Service-Zeiten und Streuungsmaße zu den einzelnen Lastschritten angegeben werden können, muss der funktionale Trace (siehe Kapitel 3.2.3) ausgewertet werden. Zur Kapselung der Performancedaten zu den einzelnen Interaktionsschritten kann das Prinzip der Transaktionsverwaltung (siehe Kapitel 3.1.2) angewendet werden. Die vorgestellten Performance-Traces lassen eine Zuordnung der Leistungsdaten zu den jeweiligen Transaktions-IDs zu (siehe Angabe der Transaktions-IDs in den Traces in Kapitel 3.2).

Die in Tabelle 4-1 definierten Einflussfaktoren der Lastschritte beinhalten neben den behandelten Themen zudem noch potentielle Datenbankaufrufe sowie Abbrüche durch Timeouts. Datenbankaufrufe werden in den nachfolgenden Kapiteln zu den Sperrobjekten, der Tabellenpufferung sowie der Datenbank behandelt. Abbrüche durch Timeouts stellen sicher, dass ein Lastschritt nicht dauerhaft im System bleibt und den belegten Applikations-Thread nicht mehr freigibt. Eine derartige Situation könnte beispielsweise durch eine nicht (rechtzeitig) aufgelöste Sperrsituation eintreten. Daher ist es möglich, eine maximale Bearbeitungszeit für eine Entry anzugeben. In dem Simulationslog kann im Nachgang geprüft werden, mit welcher Wahrscheinlichkeit es zu Überschreitungen der maximalen Service-Zeit gekommen ist.

Abbildung 4-6 stellt die Modellierung der Lastschritte schematisch dar.

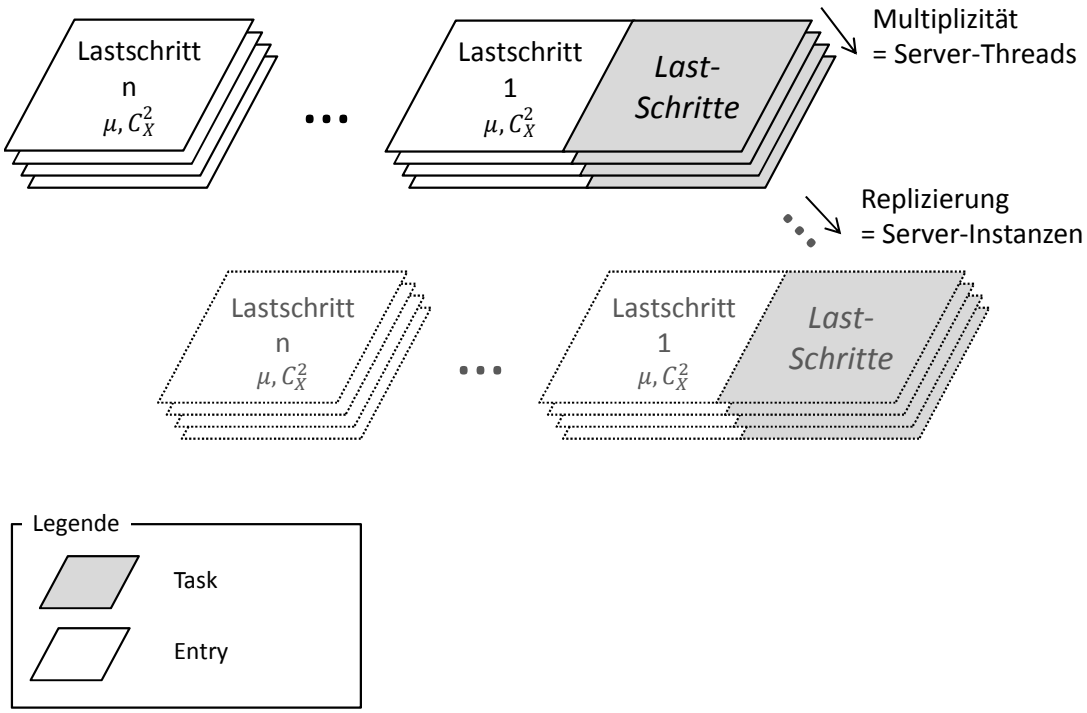


Abbildung 4-6: LQN-Modellierung der Lastschritte
Quelle: eigene Darstellung

Parametrisierung

Im Folgenden werden die Parameter der einzelnen Lastschritte genannt. Dabei beschränkt sich die Aufzählung auf die Parameter für die Konfiguration der individuellen Eigenschaften, die im vorangegangenen Abschnitt diskutiert wurden:

- Die Anzahl der Server-Threads wird über die Multiplizität des LQN-Tasks angegeben und mit dem Parameter <*multi_server_flag*> definiert.

- Die Anzahl der Server-Instanzen kann in bestimmten Fällen über Repliken gesteuert und mit dem Parameter <*replication_flag*> spezifiziert werden.

- Soll ein Dispatcher modelliert werden, wird dieser in einem eigenen Task modelliert, der eine Aktivität beinhaltet, die über eine oder mehrere Oder-Gabelungen auf die verschiedenen Server-Instanzen verweist. In dieser Arbeit wird aus den im Abschnitt über die Umsetzung genannten Gründen auf die explizite Modellierung des Dispatchers verzichtet und ist somit implizit über die Multiplizität des Lastschritte-Tasks gegeben.

- Der Scheduling-Mechanismus wird über den Parameter <*task_sched_type*> bestimmt.

- Die Warteschlangengröße (nicht zu verwechseln mit der Multiplizität und somit der Anzahl an Bedienstationen des Tasks) wird in dieser Arbeit nicht definiert, damit es im Simulationsmodell nicht zur unerwarteten Ablehnung von Anfragen kommt, die im realen System aufgrund der Warteschlange des Dispatchers und dessen Verteilungsalgorithmus nicht auftreten würden. Grundsätzlich kann diese aber über den Parameter <*queue_length*> spezifiziert werden.

- Ebenso werden keine unterschiedlichen Prioritäten (Parameter <*entry_priority*>) einzelner Anfragen, abgebildet durch die entsprechende Entry, gesetzt. Dies begründet sich direkt aus dem Verhalten des betrachteten Portalsystems.

- Die durchschnittliche Service-Zeit μ wird über den Parameter <*service_time*> angegeben.

- Die Streuung C_X^2 wird über den Parameter <*coeff_of_variation*> gesteuert.

- Der Parameter <*max_service_time*> spezifiziert die maximale Bearbeitungszeit.

- Die Anzahl der (synchronen) Anfragen definiert der Wert des Parameters <*rendezvous*>. Ob es sich bei diesem Wert um eine deterministische oder stochastische Angaben handelt, wird mit dem Schalter <*ph_type_flag*> für jede Phase festgelegt (0 = stochastisch, 1 = deterministsich) und somit auf 1 gesetzt.

4.2.3 Sperrmanagement

Wie bereits in Kapitel 3.1.3 dargestellt, können Datenbankaufrufe der einzelnen Interaktionsschritte Sperren erfordern. Diese werden über den Enqueue-Server verwaltet, der die einzelnen Sperranfragen sequentiell abarbeitet. Dabei kann es zu Wartezeiten kommen, bis die gewünschte Sperre gesetzt ist. Sollte das zu sperrende Objekt bereits gesperrt sein, erhöht sich die Wartezeit, bis das Objekt wieder freigegeben wird und die Sperre gesetzt werden kann. Bei den Sperranfragen wird zwischen Lese-, Schreib-, exklusiven Schreib- und optimistischen Sperren unterschieden.

LQN-Modellierung

Für die Abbildung eines Sperrmechanismus kann ein spezieller Task-Typ verwendet werden, ein sogenannter Semaphor-Task (Franks 2011). Ein Semaphor-Task beinhaltet zwei Entries: eine Entry „Signal“ und eine Entry „Wait“. Die Wait-Entry wird über eine synchrone Anfrage aufgerufen. Daraufhin werden so lange keine Anfragen mehr bedient, bis eine Anfrage an die Signal-Entry gesendet wird.

Ein Semaphor-Task bildet grundsätzlich einen Binärsemaphor ab. Für einen Zählsemaphor kann ein Multi-Server (siehe Abbildung 4-4) verwendet werden. Dabei gibt die Multiplizität die Anzahl der „Token“ an. Da ein regulärer LQN-Task mit einer Multiplizität > 1 (Multi-Server) eine einzige Warteschlange teilt, wurde dieses Verhalten bei den Semaphor-Tasks verändert, da bei einem Zählsemaphor jede Instanz des Multi-Servers eine eigene Warteschlange benötigt (siehe Abbildung 4-7).

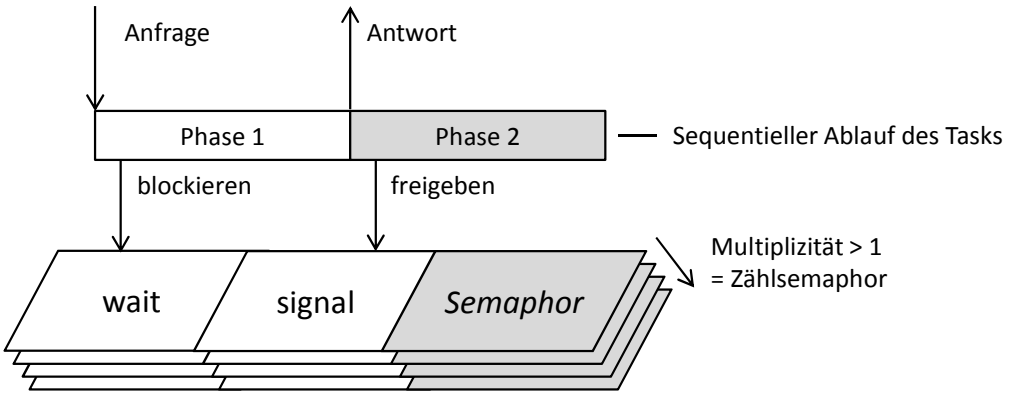


Abbildung 4-7: Funktionsprinzip eines Semaphor-Tasks
Quelle: eigene Darstellung

Für die Umsetzung der Sperr- und Löseanfragen ist es nicht zwingend notwendig, Aktivitäten zu modellieren, da diese über die verschiedenen Phasen einer Entry abgebildet werden können. Sobald eine Anfrage eine Entry erreicht, wird die Service-Phase (Phase 1) gestartet. Aufgrund des in dieser Phase notwendigen Datenbankaufrufs wird eine Sperre gesetzt, die über eine Anfrage an die Wait-Entry des Semaphors realisiert wird. Sobald die Anfrage von dem Server-Task abgearbeitet wurde, wird eine Antwort an den aufrufenden Task gesendet. Die anschließend initiierte zweite Phase des Server-Tasks sendet daraufhin eine Anfrage an die Signal-Entry des Semaphors und gibt somit das Objekt, welches durch den Semaphor gesperrt wurde, wieder frei.

Komplexere Sperrmechanismen, wie die in dem Portalsystem verwendeten Lese- und Schreibsperrtypen, sind mittels LQN-Modellierung nicht eins zu eins umsetzbar. Dazu schreibt C. Murray Woodside:

> „A full lock system, with many separate locks, and read and write locks, requires special treatment. The queueing disciplines are somewhat arcane, and there are too many locks to represent each one separately. This is a subject of current research.” (Woodside 2002, 18)

Ebenso formuliert Greg Franks in seiner kürzlich erschienenen Arbeit über passive Ressourcen in LQNs:

> "In the future, the semaphore task described here will be extended to handle read-write locks. A multi-entry, multi-queue task will be necessary with separate queues for the read acquire, write acquire and release operation." (Franks 2011, 14)

Zur Veranschaulichung des Problems sei ein Beispiel dargestellt: Ein erster intuitiver Ansatz sähe einen Lese- und einen Schreibsemaphor vor, die jeweils den Zugriff für Lese- und Schreiboperationen regeln. Der Lesesemaphor weist eine unendliche Multiplizität auf (da mehrere Leseoperationen gleichzeitig aktiv sein können), der Schreibsemaphor die Multiplizität von 1. Da Leseoperationen den Schreibzugriff sperren müssen, wird von der Leseoperation eine Schreibsperre gesetzt. Eine zweite, simultane Leseoperation würde zwar von dem Lese-Semaphor sofort bedient werden, der Aufruf der Wait-Entry des Schreibsemaphors (zum Schutz vor Schreibzugriffen) würde jedoch so lange in der Warteschlange verweilen, bis die erste Leseoperation ihren Zugriff beendet hat und über einen Aufruf der Signal-Entry die Schreibsperre aufhebt.

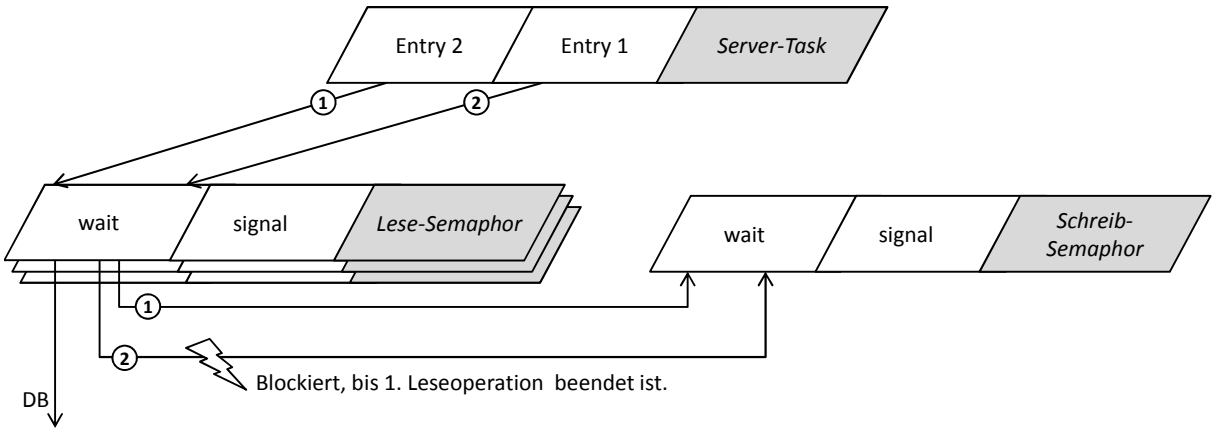


Abbildung 4-8: Beispiel zweier sich blockierender Leseoperationen
Quelle: eigene Darstellung

Umsetzung

In einem SAP-Netweaver-Portal-System kommen Lese- und Schreibsperren zum Einsatz. Da, wie soeben erwähnt, in LQN-Modellen ein Lese-Schreib-Sperrmechanismus nicht nativ umgesetzt werden kann, soll folgende Annäherung das Sperrverhalten beschreiben:

Im LQN-Modell wird jede Datenbankoperation, die im Workload auftritt, modelliert, damit die einzelnen Service-Zeiten entsprechend vergeben werden können. Datenbankoperationen, die sich gegenseitig ausschließen, fallen in einen Sperrbereich. Für jeden Sperrbereich wird eine Sperrverwaltung, wie in Abbildung 4-9 dargestellt, modelliert.

Eine Sperrverwaltung für einen Sperrbereich besteht aus drei Komponenten:

1. *Einem regulären LQN-Task für die Lesesperre des Sperrbereichs*: Jede Entry des LQN-Tasks stellt die (lesende) Datenbankoperation dar, die im Workload auftritt und in diesen Sperrbereich fällt. Die Multiplizität wird auf unendlich gesetzt, da mehrere

Leseoperationen gleichzeitig durchgeführt werden können. Jede Entry wird über einen Aktivitätsgraphen beschrieben, der noch näher erläutert wird.

2. *Einem regulären LQN-Task für die Schreibsperre des Sperrbereichs*: Jede Entry des LQN-Tasks stellt die (schreibende) Datenbankoperation dar, die im Workload auftritt und in diesen Sperrbereich fällt. Zudem wird eine Entry für die Abfrage, ob zurzeit eine Schreiboperation ansteht oder gerade ausgeführt wird, eingerichtet. Die Multiplizität wird auf 1 gesetzt, da nicht mehrere Schreiboperationen gleichzeitig durchgeführt werden können. Jede Entry, bis auf die zusätzliche Abfrage-Entry, wird über einen Aktivitätsgraphen beschrieben.

3. *Einen Semaphor-Task für die Sperren zwischen Lese- und Schreiboperationen*: Die Entries *Wait* und *Signal* stellen dabei die Operationen für das Sperren und Freigeben des Semaphors dar. Die Multiplizität des Semaphors wird auf 1 gesetzt und stellt somit einen Binärsemaphor dar.

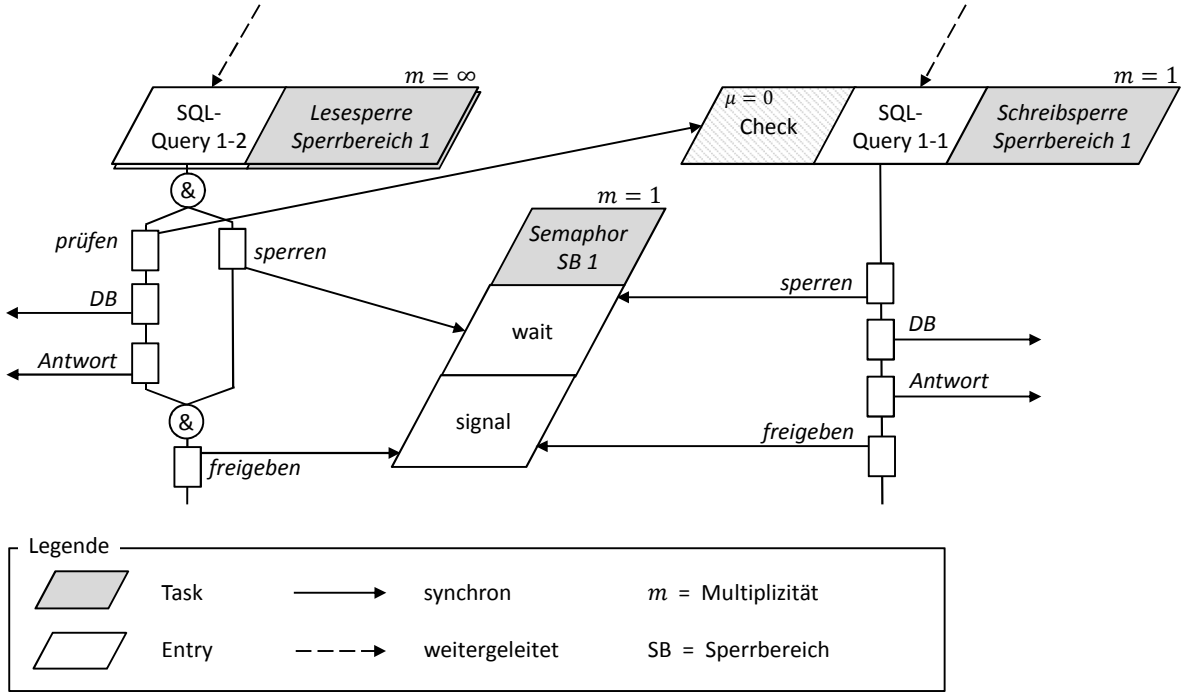


Abbildung 4-9: Aktivitäten der Sperrobjekte
Quelle: eigene Darstellung

Der Aktivitätsgraph in der Lese- bzw. Schreibsperre wird wie folgt beschrieben:

1. Die Anfrage einer Lesesperre erreicht den Lesesperren-Task des betreffenden Datenbankbereichs. Die Aktivität teilt sich im Anschluss über eine Und-Gabelung zu zwei nebenläufigen Strängen. Der erste Strang versucht über einen synchronen Aufruf (Rendezvous), eine Sperre bei dem Semaphor zu setzen. Der zweite Strang sendet zuerst eine synchrone Anfrage an die Check-Entry der Schreibsperre des betreffenden Datenbankbereichs. Solange eine Schreiboperation ansteht oder derzeit durchgeführt wird, wartet dieser Aktivitätsstrang auf eine Antwort. Sobald der Schreibsperren-Task

die Anfrage bedienen kann (Service-Zeit 0), erhält der Aktivitätsstrang eine Antwort und fährt mit der nächsten Aktivität fort: dem Aufruf der Datenbankoperation. Sobald diese abgeschlossen ist, wird die Antwort an das aufrufende Objekt (der Lastschritt-Entry) gesendet. Sobald beide Aktivitätsstränge ihre Aufgabe erledigt haben, wird über einen Aufruf der Signal-Entry der Semaphor freigegeben. Da diese Aufgabe lediglich eine verwaltende Tätigkeit der LQN-Simulation ist, wird ihr kein Ressourcenverbrauch zugeschrieben. Sollte eine Lesesperre gerade mit der Datenbankoperation beschäftigt sein und eine zweite Leseoperation ankommen, wird diese nicht blockiert, da Anfrage an den Schreibsperren-Task sofort abgeschlossen werden kann und der Datenbankaufruf sowie die Antwort an den aufrufenden Lastschritt erfolgen kann. Lediglich der zweite Aktivitätsstrang (die Sperranfrage an den Semaphor) bleibt solange blockiert, bis die erste Leseoperation den Vorgang komplett abgeschlossen und der Semaphor wieder freigegeben hat, sodass auch die zweite Leseoperation die Sperre setzen und sofort wieder freigeben kann. Dies stellt allerdings kein Problem dar, da dieser Verwaltungsakt keine Ressourcen verbraucht und lediglich die Aktivität abschließt.

2. Bei einer Schreibsperrenanfrage (die aufgrund der Multiplizität von 1 erst bedient werden kann, sobald alle früher angekommenen Schreibsperrenanfrage desselben Sperrbereichs abgearbeitet wurden) durchläuft die Entry eine Sequenz von Aktivitäten im Aktivitätsgraph. Die erste Aktivität sendet eine synchrone Anfrage an den Semaphor und bleibt solange blockiert, bis die Sperre gesetzt werden konnte. Dies stellt sicher, dass keine (früher eingetroffenen) Leseoperationen derzeit durchgeführt werden oder anstehen. Sobald die Sperre gesetzt werden konnte, kann in den folgenden Aktivitäten die Datenbankoperation durchgeführt, die Antwort an die Lastschritt-Entry gesendet und die Sperre wieder freigegeben werden. Solange eine Schreibsperrenanfrage durch den Schreibsperren-Task bedient wird, können keine später ankommenden Leseanfragen Datenbankoperationen durchführen, da der vorhin beschriebene Aufruf der Check-Entry nicht bedient werden kann.

Neben dem genannten Sperrprinzip von Sperrbereichen wird der Enqueue-Server im Modell abgebildet (siehe Abbildung 4-10). Der Enqueue-Server nimmt Sperranfragen der Lastschritte entgegen. Dazu werden die einzelnen Datenbankoperationen als Entry des Enqueue-Servers abgebildet. Da der Enqueue-Server die Anfragen sequentiell abarbeitet, wird ihm eine Multiplizität von 1 zugewiesen. Dadurch wird verhindert, dass zwei Sperranfragen zeitgleich bei den Sperranfrage-Tasks eintreffen und für eine ungewollte Nebenläufigkeit sorgen. Die eingehenden Sperranfragen werden dann von der jeweiligen Entry über eine weitergeleitete Anfrage an den Sperr-Task des betreffenden Sperrbereichs weitergeleitet. Sperrbereiche kapseln die Menge an Datenbankoperationen, die sich gegenseitig ausschließen (müssen).

Die Zeit, die für die Weiterleitung der Sperranfrage beansprucht wird, setzt sich aus der Wartezeit am Enqueue-Server und der Bedienzeit zusammen. Die Bedienzeit wird über die bekannten Maße μ (durchschnittliche Service-Zeit) und C_X^2 (quadrierter Variationskoeffizient) spezifiziert. Dies kann aus den Einträgen des Enqueue-Server-Traces berechnet werden.

Das vorgestellte Sperrkonzept führt anschließend die Sperrung des Sperrbereichs durch. Die Wartezeiten, die aufgrund einer bestehenden Sperre entstehen, entsprechen den Wartezeiten, die auch im realen System auftreten.

Der Aufruf der Datenbankoperation innerhalb des Aktivitätsgraphs sendet eine synchrone Anfrage an die entsprechende Entry des Datenbank-Tasks. Die Multiplizität des Datenbank-Tasks wird dabei auf die Anzahl der Datenbankverbindungen gesetzt, die die Java-Instanz(en) am Datenbankserver aufbauen. Da die Datenbank nicht im Detail modelliert wird und als Black-Box betrachtet wird, werden die durchschnittlichen Service-Zeiten der einzelnen Datenbankoperationen spezifiziert. Die Datenbankkomponente wird noch im Detail in Kapitel 4.2.5 erläutert.

Sobald die Datenbankoperation durchgeführt worden ist und die Antwort die Entry am Lese- bzw. Schreibsperren-Task erreicht, wird die Antwort an die aufrufende Lastschritt-Entry gesendet. Dies erfolgt aufgrund des weitergeleiteten Aufrufs der Anfrage vom Enqueue-Server an den Lese-/Schreibsperren-Task.

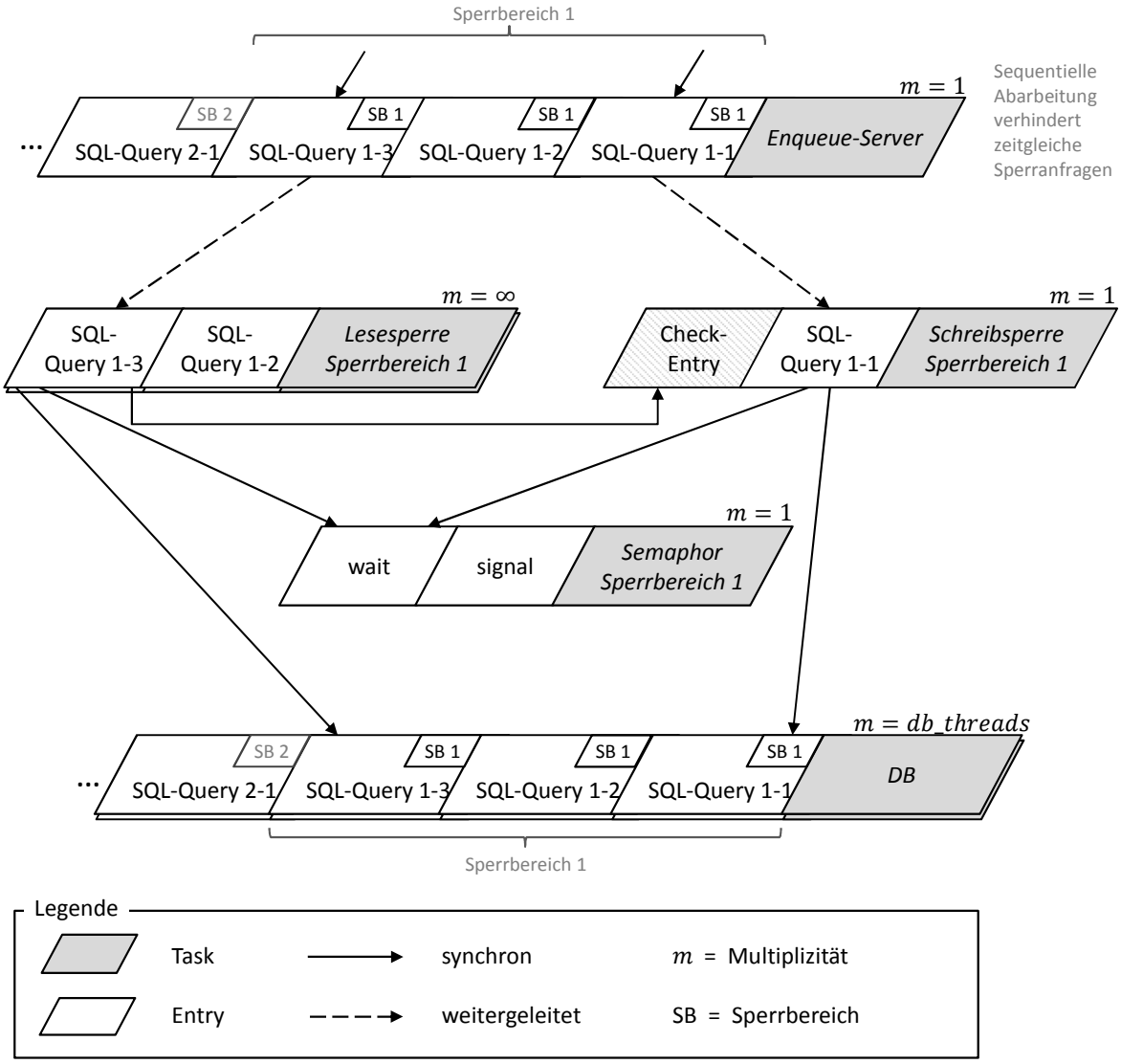


Abbildung 4-10: LQN-Modellierung der Sperrverwaltung
Quelle: eigene Darstellung

Optimistische Sperren werden erst dann von einer Lese- zu einer Schreibsperre, wenn es aufgrund einer durchgeführten Änderung des Benutzers notwendig ist. Da der Workload und somit die Benutzeraktionen bekannt sind, können optimistische Sperren als entsprechende Lese- bzw. Schreibsperren modelliert werden.

Kumulative Schreibsperren können nur dann gesetzt werden, wenn die Schreibsperre von demselben Eigentümer angefordert wird. Daher können nach der Analyse der Sperranforderungen in der Log-Datei (siehe Kapitel 3.1.3) kumulative Sperren eines Lastschrittes zu einer (1) Schreibsperre zusammengefasst werden.

Die Ermittlung der Sperrbereiche erfolgt über die Analyse der Sperranforderung-Log-Dateien sowie der SQL-Traces, bei der die einzelnen Datenbankoperationen der im Workload durchgeführten Lastschritte aufgezeichnet werden. Jede Datenbankoperation besteht neben der eigentlichen SQL-Anfrage aus weiteren Schritten (connect(), commit(), close(), usw.), die im SQL-Trace einzeln angegeben sind. Diese Schritte müssen zunächst zu einer Datenbankoperation zusammengefasst und die durchschnittliche Service-Zeit sowie die Streuung berechnet werden. Die einzelnen Datenbankoperationen, die sich gegenseitig ausschließen, werden wiederum zu einer Gruppe zusammengefasst und stellen einen Sperrbereich dar. Nicht alle in den Lastschritten durchgeführten Datenbankaufrufe fordern eine Sperre an (dies liegt am Entwickler des Java-Programms). Daher wird über die Enqueue-Log-Datei geprüft, welche Sperranforderungen gesendet wurden und anschließend werden nur jene modelliert, die auch tatsächlich eine Sperrung durchführen.

Parametrisierung

Im Folgenden wird die Parametrisierung der Sperrverwaltung dargestellt. Auch hier soll sich die Aufzählung auf die diskutierten Aspekte beschränken:

- Die zwei Entries Wait und Signal werden über die Entry-Definitons-Option „*P*“ für die Signal-Entry und Option „*V*“ für die Wait-Entry spezifiziert. Für jeden Sperrbereich wird ein eigener Semaphor eingerichtet.

- Die Aktivitäten in den Lese- bzw. Schreibsperr-Tasks werden über die Aktivitäts-Notation realisiert. Aufgrund der komplexen Notation soll hier auf die einzelnen Objekte nicht näher eingegangen, sondern auf das Benutzerhandbuch (Franks et al. 2012) verwiesen werden.

- Durchschnittliche Service-Zeiten, maximale Service-Zeiten und quadrierte Variationskoeffizienten werden für jede einzelne Aktivität im Aktivitätsgraph festgelegt. Ebenso wird der Ressourcenverbrauch (durch Koppelung an einen Prozessor-Task) für jede Aktivität einzeln definiert. Allerdings betreffen die CPU-Zeiten der einzelnen Lastschritte, die im funktionalen Trace (siehe Kapitel 3.2.3) ausgelesen werden können, den gesamten Lastschritt. Feingranularere CPU-Zeiten der einzelnen Aktivitäten des Lastschrittes sind nicht vorhanden und können somit nicht modelliert werden. Daher wird lediglich die Lastschritt-Entry an einen Prozessor-Task gekoppelt. Darunterliegende Komponenten sind somit implizit (mittels Durchschnittswerten) mit inbegriffen und werden nicht mehr an einen Prozessor-Task gekoppelt.

- In einem Aktivitätsgraphen, der synchrone Anfragen annimmt, muss genau eine Antwort-Aktivität definiert sein. Das Erreichen dieser Aktivität muss bei jedem Ablaufmuster garantiert sein. Dies ist beim Lese- und Schreibsperren-Task der Fall und wird direkt nach der Datenbankoperation ausgeführt.

- Die Anfragen werden vom Enqueue-Server über einen weitergeleiteten Aufruf durchgereicht. Die Funktionsweise von weitergeleiteten Aufrufen wurde bereits in Kapitel 2.5.1 beschrieben.

- Die Multiplizität der diskutierten Komponenten wird über den Parameter <*multi_server_flag*> spezifiziert. Die Multiplizität des Lesesperren-Tasks wird auf infinit gesetzt, da simultane Leseoperationen erlaubt sind. Die Multiplizität der Datenbankkomponente entspricht der Anzahl an Datenbankverbindungen, die vom gesamten System aufgebaut werden. Die restlichen Komponenten werden mit einer Multiplizität von 1 versehen.

4.2.4 Tabellenpufferung

Zur Erhöhung der Performance werden für Zugriffe auf bestimmte Datenbereiche Puffer verwendet, die im Hauptspeicher vorgehalten werden. Ein Zugriff auf den Hauptspeicher erfolgt wesentlich schneller als der vergleichsweise langsame Datenbankzugriff. Die zwei wesentlichen Merkmale bei der Tabellenpufferung, die als Einflussgrößen gewertet werden können, sind Verdrängungen und Invalidierungen. Verdrängungen werden nach dem Least-Recently-Used-Prinzip (LRU-Prinzip) durchgeführt. Sollte ein Tabellenbereich neu gepuffert werden, der Tabellenpuffer allerdings nicht mehr genügend freien Speicher aufweisen, müssen ältere Pufferobjekte gelöscht werden. Dabei werden die Objekte zuerst aus dem Tabellenpuffer gelöscht, die am längsten nicht mehr benötigt wurden.

Invalidierungen treten immer dann ein, wenn ein Datensatz verändert wird und somit der Inhalt des Puffers veraltet ist. Die Verdrängung bzw. Invalidierung des gepufferten Tabellenbereiches an sich verursacht keinen nennenswerten Ressourcenverbrauch, der bei der nächsten Anfrage notwendig gewordene Datenbankzugriff allerdings bedeutet eine höhere Antwortzeit und muss somit vom LQN-Modell berücksichtigt werden. Damit ein permanentes Laden einer Tabelle, die ständig invalidiert wird, verhindert wird, kann eine Tabelle nach dem Invalidieren erst nach Ablauf einer Wartezeit wieder in den Puffer geladen werden.

Es werden nur dann die Daten aus dem Puffer gelesen, wenn das Open-SQL-Statement eine Select-Anweisung ist, die alle Schlüsselfelder beinhaltet, die bei der Definition des Pufferobjektes verwendet wurden (siehe Tabellenpuffer-Arten in Kapitel 3.1.4). Verschachtelte SQL-Select-Statements werden nicht vom Puffer bedient, sondern direkt an das Datenbanksystem übergeben, da diese dort effizienter verarbeitet werden. Zusätzlich kann der Zugriff auf den Tabellenpuffer explizit übergangen werden. Dazu wird dem SQL-Statement eine Bypass-Klausel hinzugefügt.

LQN-Modellierung

In der Literatur zur LQN-Modellierung konnten keine Ansätze zur Modellierung von Puffern, die der Funktionsweise der Tabellenpuffer entsprechen, gefunden werden. Lediglich die Arbeit von Franks (2011) beschäftigt sich mit passiven Ressourcen, bei der die bereits vorge-

stellten Semaphor-Tasks eingesetzt werden. Es wird gezeigt, dass Semaphore zur Modellierung des Puffer-Pools für die Videoaufzeichnung eines Sicherheitssystems eingesetzt werden können. Der Einsatz des Semaphors dient dabei der Sperrung des Pufferbereichs, bis die Videodaten persistent in die Datenbank geschrieben wurden. Dieses Puffer-Prinzip entspricht somit nicht der Funktionsweise der in einem SAP-Netweaver-Portal-System verwendeten Tabellenpuffer, bei dem Daten aus der Datenbank gelesen und in den Tabellenpuffer geschrieben werden können, damit zukünftige Zugriffe auf den entsprechenden Tabellenbereich einen effizienteren Weg über den Puffer gehen können.

Stochastisch betrachtet kann man die Frage, ob ein Objekt aus dem Tabellenpuffer oder der Datenbank gelesen wird, mit einer Weiterleitungswahrscheinlichkeit vom Puffer zur Datenbank beantworten. In der LQN-Modellierung wird die Weiterleitung einer Anfrage von einem zum nächsten Task über eine weitergeleiteten Aufruf (siehe Kapitel 2.5.1) modelliert. Diesem kann eine Weiterleitungswahrscheinlichkeit zugewiesen werden, die angibt, mit welcher Wahrscheinlichkeit die Anfrage nicht von dem Task selbst bearbeitet wird. Übertragen auf die Tabellenpuffer gibt die Weiterleitungswahrscheinlichkeit an, mit welcher Wahrscheinlichkeit das Objekt nicht vom Puffer, sondern von der Datenbank gelesen wird.

Umsetzung

Wie soeben beschrieben, wird das Verhältnis zwischen dem Lesen vom Puffer und dem Lesen von der Datenbank über die Weiterleitungswahrscheinlichkeit angegeben. Die Weiterleitungswahrscheinlichkeit hängt von verschiedenen Größen ab, die im Folgenden analysiert werden sollen. Da lediglich die Puffergröße sowie die gepufferten Objekte bekannt, die Zugriffssequenzen und folglich die Verdrängungen und Invalidierungen aber abhängig von dem tatsächlichen Verlauf sind, kann die Bestimmung der Weiterleitungswahrscheinlichkeit nur unter vereinfachenden Annahmen („Poor Man's Assumption") erfolgen.

Verdrängungen löschen das Objekt im Puffer, sodass es bei der nächsten Leseanfrage von der Datenbank gelesen werden muss. Die Verdrängungshäufigkeit hängt von zwei Faktoren ab:

- Von der Größe des Tabellenpuffers im Verhältnis zur Gesamtgröße aller Objekte, die im Workload gepuffert werden.
- Von der Zugriffshäufigkeit auf das Objekt (LRU-Prinzip).

Unter der Annahme, dass alle Objekte gleich groß sind und dieselbe, gleichverteilte Zugriffshäufigkeit z besitzen, also

$$z = \frac{Gesamt{-}Zugriffe}{Anzahl\ der\ Objekte} ,$$

ergibt sich die Wahrscheinlichkeit, dass sich ein Objekt im Puffer befindet, aus dem Verhältnis der Puffergröße zur Gesamtgröße aller Pufferobjekte:

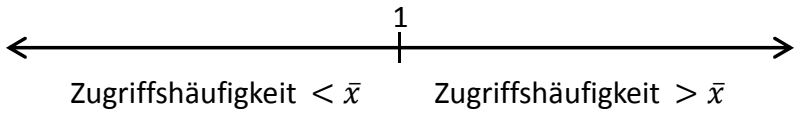
$$P_{Puffer} = \frac{Puffergröße}{Gesamtgröße\ aller\ Pufferobjekte}$$

Wie bereits erwähnt, nimmt neben der Puffergröße die Zugriffshäufigkeit Einfluss auf die Wahrscheinlichkeit, ob ein Objekt im Puffer enthalten ist. Nimmt man nun vereinfachend an,

dass die Zugriffsintervalle dieselbe Dauer aufweisen, lässt sich eine Gewichtung w der Ausgangswahrscheinlichkeit P_{Puffer} ermitteln, und zwar:

$$w = \frac{Anzahl\ Zugriffe\ auf\ Objekt}{Gesamtanzahl\ Zugriffe} \cdot Anzahl\ Objekte$$

Ist $w = 1$, entspricht die Zugriffshäufigkeit dem arithmetischen Mittel $\bar{x}$ und die Grundwahrscheinlichkeit wird nicht verändert.



Das Intervall dieser Gewichtung liegt allerdings bei $[0, Anzahl\ Objekte]$ und muss daher, in Abhängigkeit von der Grundwahrscheinlichkeit P_{Puffer}, auf das Intervall $[0, \frac{1}{P_{Puffer}}]$ getrimmt werden. Die daraus resultierende Gewichtung w' ergibt sich somit über die Fallunterscheidung:

$$w' = \begin{cases} \dfrac{w}{P_{Puffer} \cdot Anzahl\ Objekte} & f\ddot{u}r\ w > 1 \\ w & f\ddot{u}r\ w \leq 1 \end{cases}$$

In Summe ergibt sich die Wahrscheinlichkeit, mit der sich ein Objekt bei einem Zugriff im Puffer befindet mittels:

$$P_{im\_Puffer} = P_{Puffer} \cdot w'$$

Bei Invalidierungen verhält sich das Prinzip ähnlich. Je öfter ein Objekt invalidiert wurde (dies ist aus dem Tabellenpuffer-Trace ersichtlich), desto höher wird die Wahrscheinlichkeit, dass das Objekt aus der Datenbank gelesen wurde. Auch hier müssen wieder vereinfachende Annahmen erfolgen, da der genaue Ablauf zwischen Zugriffen, Verdrängungen und Invalidierungen nicht bekannt ist. Hinzu kommt das bereits beschriebene Systemverhalten, dass ein Objekt, nachdem es invalidiert wurde, für eine gewisse Zeit nicht mehr in den Puffer geladen wird, selbst wenn ein Lesezugriff darauf erfolgt. Die Gewichtung des Invalidierungseinflusses i' kann somit nur geschätzt werden und muss über mehrere Iterationszyklen und Rekalibrierungen angepasst werden. Will man nun die Wahrscheinlichkeit berechnen, mit der sich ein Objekt bei einem Zugriff nicht im Puffer befindet und die Anfrage somit an die Datenbank weitergeleitet wird, ergibt sich:

$$P_{Weiterleitung} = P_{fwd} = 1 - \left(P_{Puffer} * w' * i'\right)$$

Sobald die Weiterleitungswahrscheinlichkeit ermittelt wurde, kann das Pufferobjekt spezifiziert werden (siehe Abbildung 4-11). Für den Tabellenpuffer wird ein Task modelliert, der für jede Entry die durchgeführte SQL-Select-Abfrage abbildet. Die Multiplizität wird auf infinit gesetzt, da die Anzahl an simultanen Lesezugriffen nicht beschränkt ist.

Da die Bearbeitungszeit des Lesezugriffs auf den Puffer unerheblich ist und infolgedessen nicht in den Trace-Dateien protokolliert wird, wird die Service-Zeit der Tabellenpuffer-

Entries auf 0 gesetzt. Dadurch wirkt sich die Anfrage an den Tabellenpuffer-Task nicht auf die Antwortzeit aus, es sei denn sie wird vom Tabellenpuffertask mit der Wahrscheinlichkeit P_{fwd} an den Datenbank-Task weitergeleitet. Neben der Bearbeitungszeit der Datenbank-Entry können auch Wartezeiten auftreten, da die Multiplizität des Datenbank-Tasks der Anzahl an Datenbank-Verbindungen entspricht, wogegen die Multiplizität des Tabellenpuffers unbegrenzt ist.

Ein Pufferobjekt deckt oft verschiedene SQL-Select-Statements ab. Allerdings haben verschiedene SQL-Abfragen unterschiedliche Bearbeitungszeiten in der Datenbank, sodass sie einzeln modelliert werden müssen. Daher wird ein Pufferobjekt implizit über die Summe der SQL-Statements abgebildet, die beim Aufruf auf den Tabellenbereich des Pufferobjektes zugreifen. Demzufolge haben sämtliche SQL-Abfragen, die von demselben Pufferobjekt gekapselt werden, dieselbe Weiterleitungswahrscheinlichkeit.

Bei weitergeleiteten Aufrufen wird die Anfrage von dem Task beantwortet, der die Anfrage bearbeitet hat. Somit wird im Falle einer Weiterleitung die Antwort direkt von der Entry des Datenbank-Tasks an die Entry des Lastschrittes gesendet.

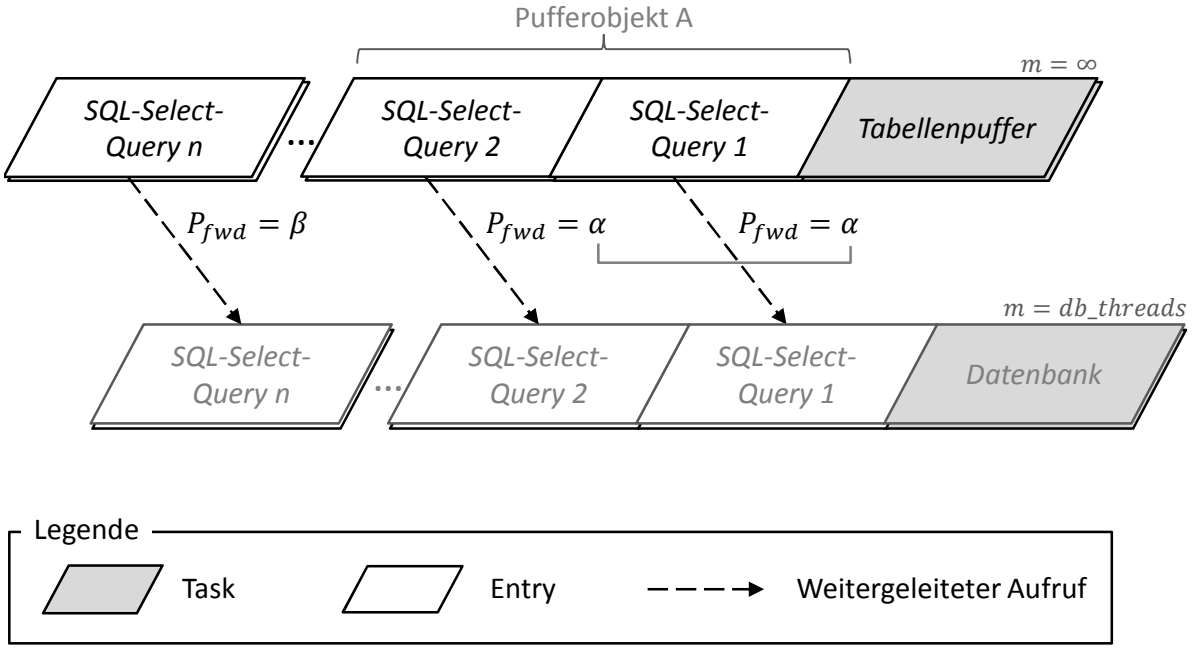


Abbildung 4-11: LQN-Modellierung der Tabellenpuffer
Quelle: eigene Darstellung

Parametrisierung

Die Punkte zur Parametrisierung beziehen sich auch hier wieder auf die diskutierten Aspekte der Tabellenpufferung:

- Die Anfragen an den Tabellenpuffer werden mit einer gewissen Wahrscheinlichkeit an die Datenbank weitergeleitet. Dazu wird die Entry-Notation mit der Option „*F*" verwendet.

- Die Weiterleitungswahrscheinlichkeit, die sich über das beschriebene Verfahren abschätzen lässt, wird als Fließkommazahl angegeben.

- Die Service-Zeit der Tabellenpuffer-Entries beträgt 0, da die Zugriffszeiten auf den Puffer zum einen unbedeutend gering sind, zum anderen nicht in den Trace-Dateien aufgezeichnet werden. In Bezug auf den Einfluss auf die Antwortzeit wird somit nur der Effekt der Verdrängungen und Invalidierungen modelliert.

4.2.5 Datenbank

Das Datenbanksystem wird, wie bereits in den vorangegangenen Kapiteln erwähnt, als Black-box betrachtet. Dies hat auf der einen Seite zwar den Nachteil, dass datenbankinterne Abläufe (wie zum Beispiel interne Puffer des Datenbanksystems) nicht explizit modelliert werden können, auf der anderen Seite ist dadurch die Modellierung der Datenbankkomponente herstellerunabhängig. Zudem wird der Schwerpunkt auf die Applikationsschicht des in dieser Arbeit betrachteten SAP-Netweaver-Portal-System gewahrt, da die feingranulare Modellierung eines Datenbanksystems eine ebenso hohe Komplexität aufweisen würde.

Datenbankaufrufe können Sperren erfordern (siehe Kapitel 4.2.3), vom Puffer bedient werden (siehe Kapitel 4.2.4) oder direkt, also ohne vorherige Sperranfragen, an das Datenbanksystem gerichtet sein.

Die einzelnen Service-Zeiten der im Workload durchgeführten Aufrufe sowie die Wartezeiten auf eine freie Datenbankverbindung (der Verbindungspool des JDBC-Services ist standardmäßig auf 10 pro Java-Server-Instanz begrenzt) werden im funktionalen Trace (siehe Kapitel 3.2.3) wiedergegeben.

LQN-Modellierung

Zur Darstellung des Datenbanksystems als Black-Box ist es ausreichend, einen LQN-Task zu modellieren. Die Multiplizität gibt dabei die Anzahl der Datenbankverbindungen an, die vom JDBC-Service im Verbindungs-Pool vorgehalten werden. Die einzelnen Entries des Datenbank-Tasks spiegeln die im Workload durchgeführten Datenbankabfragen wider. Die durchschnittliche Service-Zeit sowie das Streuungsmaß werden über die gesammelten Daten des funktionalen Traces ermittelt.

Umsetzung

Wie bereits in Kapitel 4.2.2 dargestellt, ist es ausreichend, die Multiplizität des Lastschritte-Tasks über die Summe der zur Verfügung stehenden Threads in den Java-Server-Instanzen zu spezifizieren:

$$m = \sum_{k=1}^{n} Anzahl\ Threads(ServerInstanz_k)$$

Die SQL-Abfragen werden über die Entries des Datenbank-Tasks abgebildet, wobei die Service-Zeit dem Wert entspricht, der im funktionalen Trace als „Datenbank-Zeit“ gekennzeichnet ist. Die Wartezeit, bis eine Datenbankverbindung hergestellt werden kann, ist darin nicht enthalten. Unabhängig davon ist die angegebene Service-Zeit ein Bruttowert, da diese Zeit nicht zwingend aus der reinen Bearbeitungszeit innerhalb der Datenbank bestehen muss. Wartezeiten innerhalb des Datenbanksystems würden ebenso in diesem Wert enthalten sein. Somit

spiegelt diese Angabe lediglich die Zeit wieder, die für die Abarbeitung außerhalb der Java-Server-Instanz benötigt wurde.

Betrachtet man die eingehenden Anfragen zum Datenbanksystem, sind verschiedene Wege möglich:

1. Der einfachste Fall ist ein direkter Datenbankaufruf, bei dem keine Tabellenpufferung erfolgt und keine vorangegangenen Sperranfragen gesendet wurden.

2. Im zweiten Fall können die Anfragen ebenfalls nicht vom Tabellenpuffer bedient werden. Allerdings wurde im Vorfeld von der Java-Applikation eine Sperranfrage an den Enqueue-Server gesendet.

3. Im dritten Fall werden keine Lesesperren gesetzt und die Select-Statements vom Puffer bedient.

4. Im vierten Fall werden sowohl Lesesperren gesetzt als auch Select-Statements vom Puffer bedient.

All die vier möglichen Fallkonstellationen müssen vom LQN-Modell abgedeckt werden. Betrachtet man die Modellierung der Tabellenpufferung sowie der Sperr-Objekte in den vorangegangenen Kapiteln, ergeben sich folgende Wege vom Lastschritte-Task zum Datenbank-Task:

1. Lastschritt → Datenbank

2. Lastschritt → Sperrverwaltung → Datenbank

3. Lastschritt → Tabellenpuffer (→ ggf. Datenbank)

4. Lastschritt → Sperrverwaltung (Lesesperre) → Tabellenpuffer (→ ggf. Datenbank)

Da die im Workload auftretenden SQL-Abfragen einzeln als Entry in den jeweiligen Komponenten modelliert werden, ist der jeweilige Weg vom Lastschritt zur Datenquelle eindeutig abbildbar. Lediglich die Frage, ob vom Tabellenpuffer oder von der Datenbank gelesen wird, wird über eine Weiterleitungswahrscheinlichkeit festgelegt. Abbildung 4-12 stellt die geschilderten Szenarien anhand beispielhafter Pfade grafisch dar:

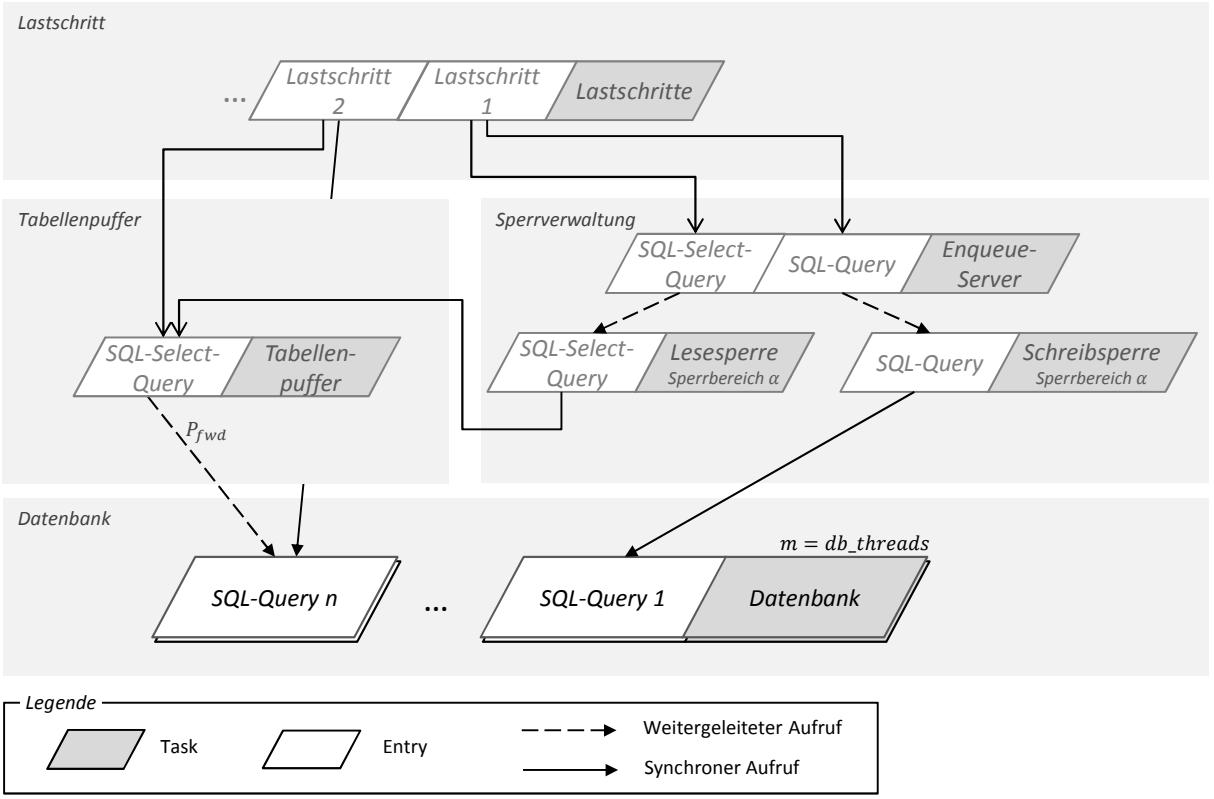


Abbildung 4-12: LQN-Modellierung der Datenbank
Quelle: eigene Darstellung

Parametrisierung

Die Parametrisierung des Datenbank-Tasks ergibt sich über die im vorangegangenen Abschnitt diskutierten Aspekte der Anzahl an Datenbankverbindungen im JDBC-Service und der aufgezeichneten Service-Zeit im funktionalen Trace:

- Die Anzahl an verfügbaren Datenbankverbindungen im Verbindungspool wird über die Multiplizität des Datenbank-Tasks abgebildet. Analog zur Aufsummierung der Server-Threads der n Java-Server-Instanzen (siehe Kapitel 4.2.2) wird auch hier die Summe der Datenbankverbindungen der n Verbindungspools gebildet:

$$m = \sum_{k=1}^{n} AnzahlVerbindungen(Verbindungspool_k)$$

- Die durchschnittliche Service-Zeit ermittelt sich aus den Messdaten der Datenbankzeit im funktionalen Trace. Obwohl diese Zeitangabe einen Bruttowert darstellt, der im LQN-Modell der CPU-Zeit entspricht, wird auf eine Abschätzung der tatsächlichen CPU-Zeit verzichtet, da diese, wie im vorherigen Abschnitt erwähnt, nicht über die verfügbaren Monitore bzw. Traces zu ermitteln ist.

4.2.6 Garbage-Collector

Der Garbage-Collector nimmt einen wesentlichen Einfluss auf die System-Performance und somit die resultierenden Antwortzeiten (siehe zum Beispiel Libic/Tuma/Buley (2009) oder Ufimtsev (2006)). Dieser Einfluss kann abhängig von der verwendeten JVM sowie GC-

Implementierung variieren (Libic/Tuma/Buley 2009). Beispielsweise unterscheiden sich die Algorithmen bei der Verwaltung der Young- und der Tenured-Generation (siehe auch Kapitel 3.1.5).

Unabhängig von der verwendeten Implementierung werden bei einem GC-Lauf die Applikations-Threads gestoppt. Dieser Zeitabschnitt wird auch als Stop-the-World-Pause bezeichnet. Daher kann eine übermäßige Aktivität des Garbage-Collectors zu hohen Antwortzeiten, ineffizienter CPU-Nutzung und Speicherauslagerung („Paging“) führen. Daraus resultieren zwei Leistungsindikatoren, die die Aktivitätsintensität des Garbage-Collectors beschreiben: zum einen die durchschnittliche Dauer, in der die Applikations-Threads blockiert sind, zum anderen das durchschnittliche Intervall von einem bis zum nächsten GC-Lauf (vgl. Cheng/Morrison 2007).

Die aufgezeichneten Messwerte des funktionalen Traces im SAP-System geben keinen Aufschluss über die GC-Zeiten. Daher ist in den angegebenen Zeiten implizit die Zeit enthalten, in der der Applikations-Thread aufgrund eines GC-Laufes nicht fortgesetzt werden konnte. Diese Tatsache führt bei der Modellentwicklung dazu, dass in der ersten Iteration auf eine explizite Modellierung des Garbage-Collectors verzichtet wird.

Zudem ist es mit den Mitteln der LQNs nicht möglich, das Verhalten des Garbage-Collectors identisch nachzubilden. Dies liegt zum einen an der stochastischen Natur von LQNs, zum anderen an den fehlenden Detailkenntnissen der GC-Implementierung (Ufimtsev 2006, 92). Die zwei Indikatoren zur Aktivitätsintensität (GC-Dauer und GC-Intervall) lassen sich jedoch über den GC-Trace und die bereitgestellten Analysetools, die in Kapitel 3.2.6 dargestellt wurden, ermitteln.

Bei der Auswertung der Simulations- und Messergebnisse, die in Kapitel 5.7 beschrieben wird, kann im Hochlastbereich ein starker Anstieg der Antwortzeiten festgestellt werden. Die Ursachenanalyse ergibt eine sehr hohe Aktivitätsintensität des Garbage-Collectors, die den übermäßigen Anstieg der Antwortzeiten erklären kann. Damit die Auswirkungen auf die Antwortzeit im Modell getestet werden können, wird in einer zweiten Iteration eine GC-Komponente eingebaut (siehe Abbildung 4-13), die die Angabe der beiden Aktivitätsindizes GC-Dauer und GC-Intervall als Parameter erlaubt.

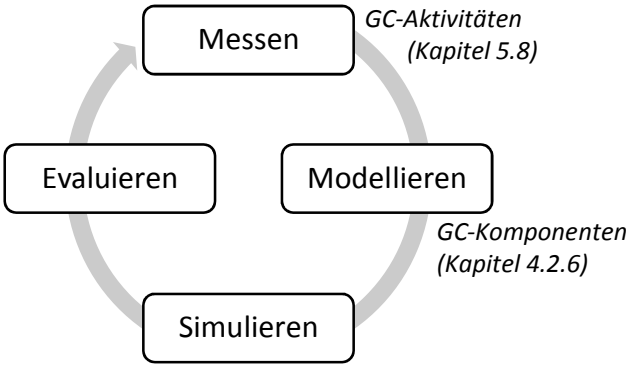


Abbildung 4-13: Zweiter Zyklus der Modellbildung
Quelle: eigene Darstellung

LQN-Modellierung

In der LQN-Literatur konnten keine Ansätze zur expliziten Modellierung des Garbage-Collectors bei Java-basierten Anwendungen identifiziert werden. Die GC-Zeiten waren stets implizit in den Bearbeitungszeiten der Komponente bzw. der Komponenten enthalten (siehe zum Beispiel Franks/Lau/Hrischuk (2011), Ufimtsev (2006), Ufimtsev/Murphy (2006) oder Xu et al. (2005)). Die genannten Gründe, die es schwer oder sogar unmöglich machen, ein akkurates Model zu erstellen, beziehen sich auf die Komplexität der JVM, die fehlende Quellcode-Offenheit der GC-Implementierungen sowie fehlende Messdaten.

Ein großer Nachteil der impliziten Parametrisierung (d.h. die Service-Zeit entspricht der Ausführungszeit inklusive GC-Anteil) ist der konstante GC-Anteil. Es ist allerdings gerade im Hochlastbereich eine überproportionale Aktivitätszunahme des Garbage-Collectors zu erwarten, da der Heap-Speicher an seine Grenzen gelangt und immer häufiger Speicherbereiche von nicht mehr benötigten Objekten freigegeben werden müssen. Nach Libic/Tuma/Buley (2009) kann der GC-Overhead sogar um ein Vielfaches, bis auf einen zweistelligen Prozentbereich, ansteigen.

Wie bereits erwähnt, lassen auch die erhobenen Messwerte, die noch in Kapitel 5.5 dargestellt werden, eine ähnliche Schlussfolgerung zu. Damit das LQN-Modell in der Lage ist, verschiedene GC-Aktivitätsmuster und deren Auswirkung auf die Antwortzeit zu testen, muss also der Garbage-Collector (bzw. die vorgestellten GC-Parameter Ausführungsdauer und Intervall) in das Modell integriert werden. Dies dient der in der zweiten Iteration durchgeführten Analyse des Hochlastbereichs.

Umsetzung

Damit die zwei Parameter GC-Dauer und GC-Intervall umgesetzt werden können, werden zwei logische Komponenten in das Modell integriert:

- *GC-Client*: Der GC-Client ist mit einem Benutzertask vergleichbar, der eine synchrone Anfrage alle α Zeiteinheiten nach dem Erhalt der Antwort der letzten Anfrage sendet. Dies wird über die Denkzeit des GC-Client-Tasks realisiert und stellt das durchschnittliche GC-Intervall zwischen den GC-Läufen dar. Die Aufrufe müssen die gesamte Zeitspanne der Simulation durchgeführt werden, also

  $$(GC_{Intervall} + GC_{Dauer}) * Aufrufanzahl \geq Simulationsdauer.$$

  Es ist ausreichend, die Anzahl der benötigten GC-Aufrufe unter dieser Bedingung abzuschätzen, da in der Simulationsausgabe unter anderem die gesuchten Antwortzeiten der Benutzertasks wiedergegeben werden und somit überschüssige GC-Aufrufe keinen Einfluss auf die Ergebnisse nehmen.

- *GC-Sim:* Die Komponente GC-Sim ahmt einen GC-Lauf nach, indem die eingehenden Anfragen vom GC-Client mit einer Bearbeitungszeit von μ bearbeitet werden. Die Bearbeitungszeit μ entspricht somit der durchschnittlichen GC-Dauer. Eine weitere Entry des GC-Sim-Tasks hat eine Bearbeitungszeit von 0 und nimmt eingehende Anfragen von den Lastschritten entgegen. Die Multiplizität der GC-Sim-Komponente beträgt 1.

Zudem wird bei jedem Lastschritt ein Aktivitätsgraph modelliert, der aus einer Sequenz von mehreren Aktivitäten besteht:

1. Zunächst wird eine synchrone Anfrage an den GC-Sim-Task gesendet. Sollte der GC-Sim-Task gerade nicht belegt sein, wird die Antwort sofort zurückgesendet. Ansonsten ist der GC-Sim-Task gerade aktiv, das im übertragenen Sinne einem GC-Lauf entspricht. Dadurch entstehen Wartezeiten, die den GC-Einfluss nachahmen.

2. Nach Erhalt der Antwort wird mit den eigentlichen Lastschrittaufgaben in Aktivität 2 fortgesetzt.

3. Nachdem die modellierten Lastschrittvorgänge durchgeführt wurden, wird schließlich die Antwort an die aufrufende Entry gesendet.

Mit der Einführung dieser logischen Garbage-Collector-Komponenten können somit 2 Parameter angegeben werden: das Intervall zwischen den GC-Client-Anfragen über die Denkzeiten der Entries und die Dauer der Bearbeitung über die Service-Zeit der GC-Sim-Entry. Natürlich sind auch diese Parameter nur Annäherungen, da zum Beispiel Lastschritte, die gerade ausgeführt werden, nicht von dem GC-Sim-Task gestoppt werden. Eine Modellierung der Stop-the-World-Pause, bei der laufende Lastschritt-Aktivitäten gestoppt werden, ist über die LQN-Modellierung nicht ohne weiteres möglich. Dennoch sei an dieser Stelle betont, dass diese Annäherung ausreicht, da die Möglichkeit geschaffen wird, über zwei Parameter den GC-Einfluss auf die Antwortzeit zu justieren. Dies ermöglicht Analysen des Antwortzeitverhaltens bei erhöhter bzw. übermäßiger GC-Aktivität, die bei impliziter Darstellung der GC-Zeiten als konstanter Faktor der Service-Zeit nicht einfach möglich gewesen wäre, da sämtliche Service-Zeiten der Lastschritte angepasst werden müssten.

Das Funktionsprinzip der logischen GC-Komponenten ist in Abbildung 4-14 noch einmal grafisch dargestellt.

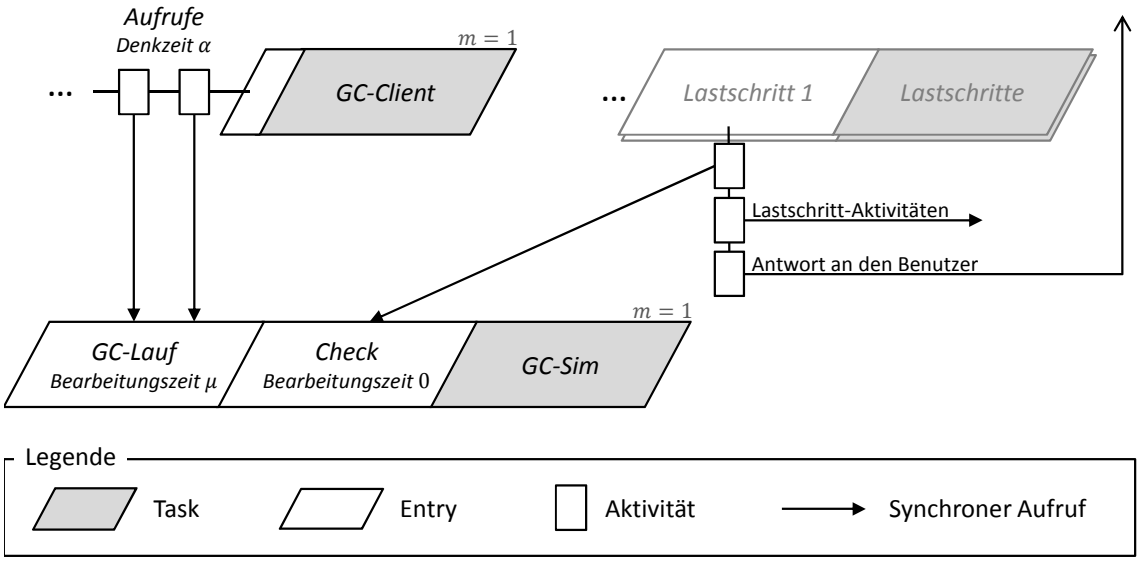


Abbildung 4-14: LQN-Modellierung der Garbage-Collector-Parametrisierung
Quelle: eigene Darstellung

Mit der expliziten Parametrisierung der GC-Aktivitätsintensität müssen die vom funktionalen Trace aufgezeichneten Ausführungszeiten der Java-Server-Tasks, also die Bearbeitungszeiten der Lastschritt-Entries, von den GC-Läufen bereinigt werden. Dies ist ein aufwändiger Prozess, da die einzelnen Lastschrittaufrufe mit den Ausführungszeitpunkten der GC-Läufe verglichen werden und die Stichproben verworfen werden müssen, die in den Zeitraum eines GC-Laufs fallen.

Beispielhaft sei die Stichprobe der Bearbeitungszeiten eines Lastschritts (Aufruf der Portalinformationen) in Abbildung 4-15 dargestellt. Hierbei wurde ein einzelner Lastschritt über den Lastgenerator 3.000 Mal automatisiert durchgeführt (nach fünf vorangegangenen Einschwingdurchläufen) und die Bearbeitungszeiten über den funktionalen Trace aufgezeichnet. Wie man unschwer erkennen kann, sind nach periodischen Abständen immer wieder einige Ausreißer mit stark erhöhten Werten zu erkennen, die sich mit den vom System durchgeführten GC-Zyklen erklären lassen.

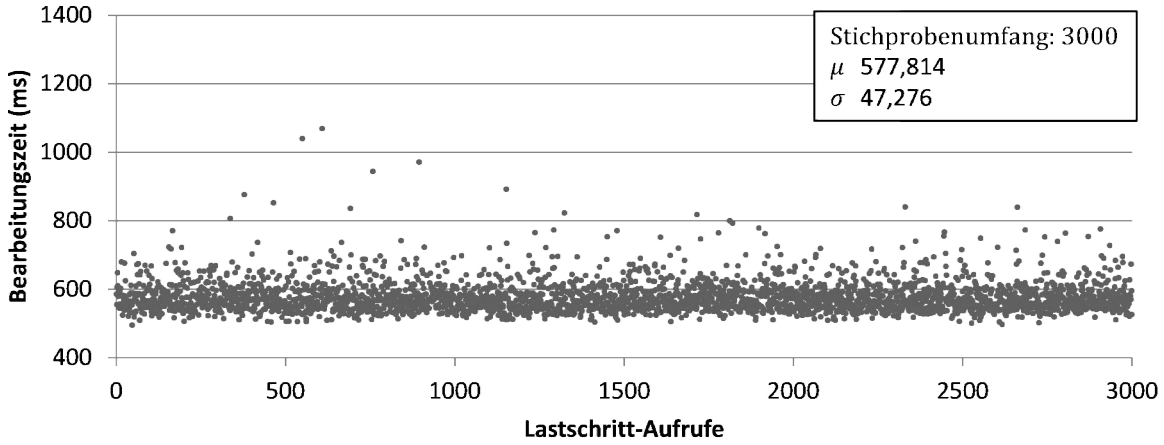


Abbildung 4-15: Bearbeitungszeiten eines Lastschritts
Quelle: eigene Darstellung

Parametrisierung

Die Parametrisierung der logischen GC-Komponenten wurde bereits angesprochen und hat folgende Merkmale:

- Das durchschnittliche GC-Intervall wird über eine Denkzeit α der GC-Client-Entries spezifiziert.
- Die durchschnittliche GC-Dauer wird über die mittlere Bearbeitungszeit μ der GC-Sim-Entry angegeben.

4.3 Gesamtbetrachtung des Modells

Die Modellierung des SAP-Netweaver-Portal-Systems beruht grundsätzlich auf der 3-Schicht-Architektur des SAP-Netweaver-Stacks (siehe Kapitel 2.1.1.1) und entspricht somit dem Modellierungsprinzip der LQNs. Die einzelnen Modellkomponenten können den Architekturschichten wie folgt zugeordnet werden:

- Präsentationsschicht: Benutzer
- Applikationsschicht: Java-Server, Tabellenpuffer, Sperrverwaltung, GC
- Persistenzschicht: Datenbank

Aufgrund der geschichteten Modellierungsweise von LQNs existieren verschiedene Ebenen, in der die Tasks platziert werden können. Somit entstehen Nachrichtenpfade, die von der Ausgangsebene bis zur untersten Ebene vordringen können und dabei nicht streng hierarchisch verlaufen müssen. Die Dauer vom Senden einer synchronen Anfrage bis hin zum Empfang der Antwort wird als Antwortzeit interpretiert.

Aufbauend auf die grundsätzliche Modellierung der einzelnen Schichten (Präsentationsschicht mittels Benutzer-Task, Applikationsschicht mittels Lastschritte-Task und Persistenzschicht mittels Datenbank-Task) wurden für die Datenbankanfragen die Sperrverwaltung sowie die applikationsseitige Tabellenpufferung explizit modelliert. Dies hat den Vorteil, dass zusätzliche Wartezeiten aufgrund des wechselseitigen Zugriffs auf die Datenbasis sowie der Leistungsgewinn mittels zwischengespeicherter Daten im Tabellenpuffer nicht nur implizit in der Service-Zeit der Datenbankkomponente enthalten ist.

Betrachtet man nun das Schichtprinzip des LQN-Modells, werden Anfragen von dem Benutzer-Task initiiert und an die darunterliegende Ebene, dem Lastschritte-Task, gesendet. Die Lastschritte wiederum können Datenbankaufrufe erfordern, die, abhängig von der optional verwendeten Sperrverwaltung und der potentiellen Verwendung des Tabellenpuffers, über verschiedene Wege realisiert werden und in Kapitel 4.2.5 dargestellt wurden.

Zusätzlich wurde zur Analyse der Auswirkungen des Garbage-Collectors die Möglichkeit geschaffen, die Aktivitätsintensität über die Näherungswerte der beiden Parameter „durchschnittliche GC-Dauer“ und „durchschnittliches GC-Intervall“ zu justieren. Die logischen GC-Komponenten des Modells betreffen das Speichermanagement der Applikationsschicht.

Abbildung 4-16 zeigt eine schematische Darstellung des LQN-Modells. Aus Gründen der Übersichtlichkeit werden die CPU-Ressourcen, die an die Servertasks gekoppelt sind, nicht dargestellt. Der Benutzer-CPU werden unbegrenzte Kapazitäten zugewiesen, um einen Einfluss auf die Benutzertasks zu vermeiden, da lediglich das Serververhalten betrachtet werden soll. Die Applikations- und Persistenzschicht verfügen jeweils über eine eigene CPU-Ressource, da die Messungen an einer dreistufigen Systeminstallation durchgeführt werden (vgl. Kapitel 2.1.1.1). Zudem werden nur repräsentative Anfragen zwischen zwei Entries dargestellt, die die möglichen Pfade vom Benutzer-Interaktionsschritt bis hin zur Datenbankkomponente verdeutlichen sollen. Die abgebildeten Komponenten zur Steuerung der GC-Aktivität werden erst im zweiten Schritt, bei der Analyse des GC-Verhaltens, eingesetzt.

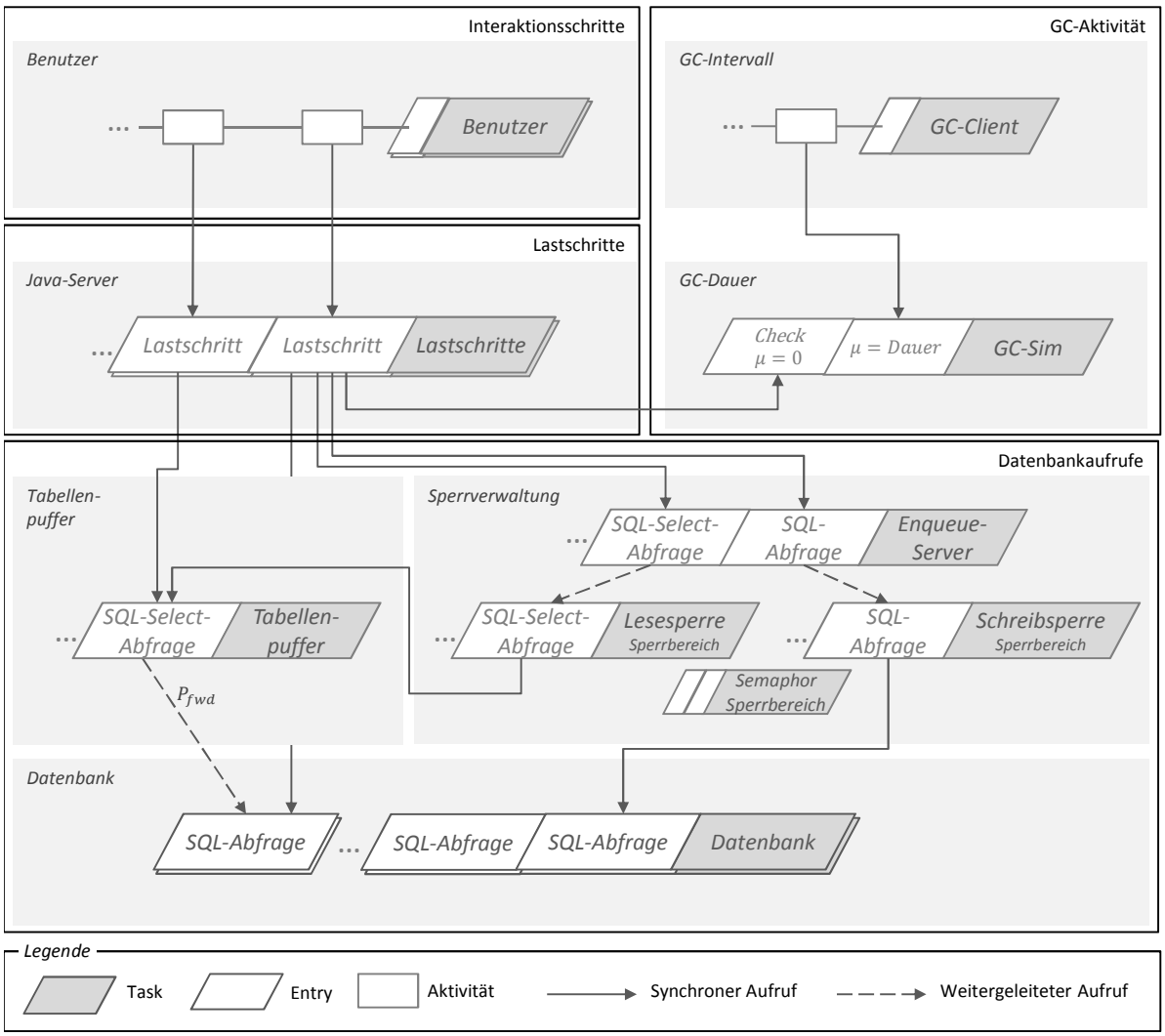


Abbildung 4-16: Schematische Darstellung des LQN-Modells
Quelle: eigene Darstellung

Mit dem vorgestellten Modell ist es möglich, SAP-Netweaver-Portal-Systeme mittels LQN zu modellieren. Die Komponenten der Sperrverwaltung, des Tabellenpuffers und die bei der Analyse des Garbage-Collectors eingeführten GC-Komponenten ermöglichen eine explizite Parametrisierung der identifizierten Einflussfaktoren im System (siehe Kapitel 4.1.2).

Der Workload wird über die modellierten Lastschritte charakterisiert und von den Benutzer-Tasks über die darin definierten Interaktionsschritte aufgerufen. Die Definition unterschiedlicher Benutzertypen ermöglicht die Charakterisierung verschiedener Aktivitäten und Denkzeiten. Da der betrachtete Workload inhärenter Bestandteil des LQN-Modells ist, muss er bei jeder Änderung entsprechend angepasst werden. Das vorgestellte Modellierungskonzept kann jedoch für eine Vielzahl von unterschiedlichen Lastmustern eines SAP-Netweaver-Portal-Systems angewendet werden und genügt somit den Anforderungen der Verallgemeinerbarkeit.

Zusammenfassend konnte mit den aus der Warteschlangentheorie abgeleiteten Grundannahmen (Kapitel 4.1.1) sowie der Identifikation und Einordnung der Einflussfaktoren (Kapitel 4.1.2) der Rahmen für das in diesem Kapitel erstellte Artefakt festgelegt und anschließend erstellt werden. Dieses Kapitel beantwortet somit Forschungsfrage 2, das heißt die Frage, wie

die in Forschungsfrage 1 identifizierten Systemkomponenten zur Leistungsanalyse eines SAP-Netweaver-Portal-Systems mittels LQN modelliert und parametrisiert werden können.

5 Simulation und Evaluation

In diesem Kapitel soll das vorgestellte Konzept der LQN-Modellierung eines SAP-Netweaver-Portal-Systems anhand einer Fallstudie des SAP-University-Competence-Centers der Technischen Universität München evaluiert werden. Dazu wird zunächst die verwendete Fallstudie beschrieben und auf die Eigenschaften des Workloads eingegangen. Im Anschluss hieran wird der Lastgenerator beschrieben, der für die im darauf folgenden Abschnitt dargestellten Messungen verwendet wird. Auf die Schilderung der verschiedenen Messszenarien und -ergebnisse folgt sodann die Erläuterung der Simulationsdurchführung. Der anschließende Vergleich der Mess- und Simulationsresultate beschreibt das Antwortzeitverhalten sowie mögliche Ursachen für die Diskrepanz zwischen den gemessenen und den simulierten Werten im Hochlastbereich. Vertiefend hierzu wird schließlich eine Analyse des GC-Verhaltens präsentiert.

5.1 Fallstudie

Die im SAP-University-Alliances-Programm (SAP UA 2012) entwickelten Schulungsunterlagen (im Folgenden Fallstudie genannt) zum SAP-Netweaver-Portal umfassen acht Kapitel, die die einzelnen Schulungstage repräsentieren und wie folgt gegliedert sind (siehe Abbildung 5-1):

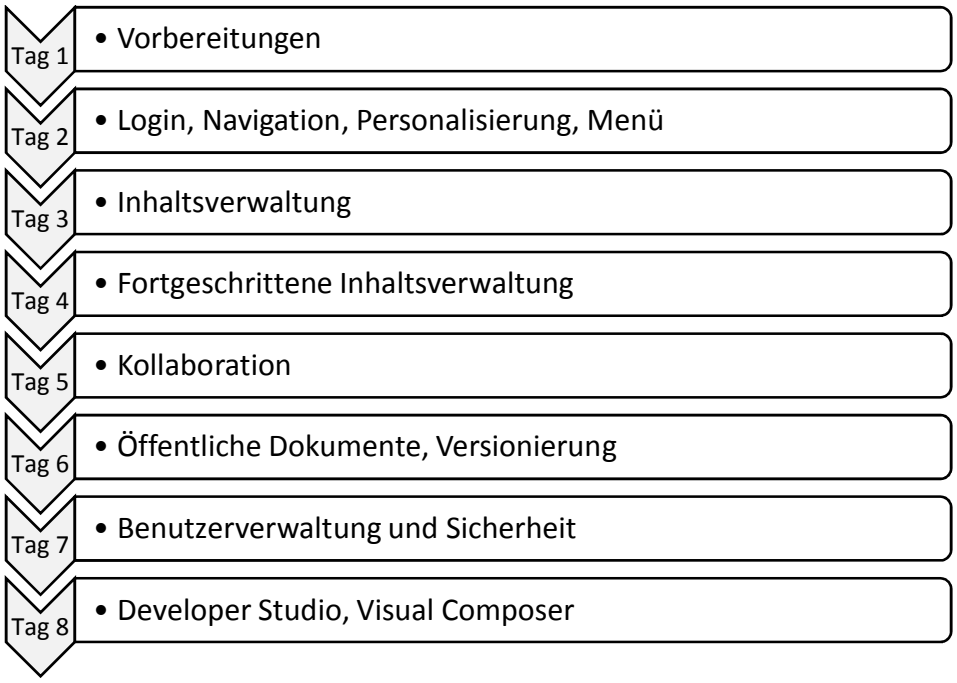


Abbildung 5-1: Schulungsinhalte der SAP-Netweaver-Portal-Fallstudie
Quelle: eigene Darstellung in Anlehnung an Jehle (2010, 63)

1. *Vorbereitungen*: Für die Durchführung der Übungen am Portalsystem ist es notwendig, diverse Vorbereitungen zu treffen. Dazu gehört im ersten Schritt das Anlegen einer eigenen Gruppe, der eine bestimmten Rolle zugewiesen wird, damit entsprechende Zugriffsrechte erlangt werden. Zudem wird ein Schulungsordner angelegt, in dem die Schulungsteilnehmer ihre erstellten Objekte ablegen können. Für die Ausführung bestimmter Experimente wird weiterhin ein virtueller Schulungsraum kreiert. Abschlie-

ßend benötigt jeder Schulungsteilnehmer ein eigenes Konto, dem ebenfalls eine entsprechende Rolle zugeordnet wird und mit einem initialen Password versehen wird. Die Vorbereitungen werden vom Dozenten durchgeführt und sind somit nicht Bestandteil der Übungen, jedoch notwendig für einen reibungslosen Ablauf der durchgeführten Aktionen am Portal.

2. *Login, Navigation, Personalisierung, Menü*: An Tag 2 loggen sich die Schulungsteilnehmer zum ersten Mal mit dem vergebenen Initialpasswort an. Daraufhin werden sie vom System aufgefordert, ein neues Benutzerpasswort zu bestimmen. Sobald dies durchgeführt wurde, werden erste Schritte am Portal durchgeführt. Dazu gehören die Änderung des Darstellungsschemas, das Eintragen detaillierter Benutzerinformationen, die Navigation innerhalb des Portals sowie weitere grundsätzliche Portaloperationen.

3. *Inhaltsverwaltung*: Am dritten Schulungstag legen die Schulungsteilnehmer eigene Unterordner im Schulungsordner an und legen in diesen ihre eigenen, erstellten i-Views (siehe Kapitel 2.1.2) ab. Auf Grundlage der erstellten iViews wird eine eigene Portal-Webseite zusammengestellt. Diese wird daraufhin einem eigens dafür erstellten Workset zugewiesen, dem damit generierten Objekt eine Rolle zugeordnet und somit die Konfiguration der Freigabeparameter durchgeführt. Über einen Aufruf der erstellten Portal-Website wird abschließend das Ergebnis überprüft. Die an diesem Schulungstag erstellten iViews beinhalten ausschließlich portaleigene Funktionen.

4. *Fortgeschrittene Inhaltsverwaltung*: Für die Integration externer Inhalte werden am vierten Schulungstag weitere iViews erstellt. Im Übrigen gleichen die durchgeführten Aktionen dem des vorherigen Schulungstages.

5. *Kollaboration*: Für die Kommunikation zwischen den Teilnehmern des Portalsystems werden Funktionen zur Zusammenarbeit zur Verfügung gestellt. Die in der Schulung durchgeführten Kollaborationsfunktionen betreffen das Erstellen eines Meeting-Rooms, die Nutzung der Instant-Message-Funktionalität sowie das Versenden von Emails. Zu diesen Kollaborationskomponenten werden zudem die benötigten Konfigurationsschritte sowie das Rechtemodell vorgestellt.

6. *Öffentliche Dokumente, Versionierung*: Zur Förderung des Austauschs von Informationen zwischen den Portal-Teilnehmern besteht die Möglichkeit, öffentliche Dokumente (beispielsweise Ordner) zu erstellen. Ein ausgeprägtes Rechtemodell sorgt für die genaue Definition, welche Teilnehmer berechtigt sind, darauf zuzugreifen. An diesem Schulungstag wird die Bereitstellung der öffentlichen Dokumente vorgestellt und durchgeführt.

7. *Benutzerverwaltung und Sicherheit*: Am vorletzten Schulungstag werden die Portaloperationen vorgestellt, die der Dozent am ersten Tag für die Schulungsvorbereitung durchgeführt hat, und zwar das Anlegen und Konfigurieren von Benutzern und Rollen.

8. *Developer Studio, Visual Composer*: Am letzten Schulungstag werden tiefergehende Werkzeuge zur Konfiguration und Entwicklung von Portalinhalten und deren Abhän-

gigkeiten vorgestellt, wobei in den Übungen lediglich auf den Visual Composer eingegangen wird.

5.2 Workload

Der modellierte Workload W_m wird, wie in Kapitel 2.3.2.1 beschrieben, über ein vereinfachtes Abbild des realen Workloads W_r gebildet. Da es für die Betrachtung komplexer Software-Systeme, zu denen das zu untersuchende Portalsystem zählt, nicht reicht, statische Eigenschaften zu betrachten (Jain 1991), werden portaleigene Funktionen verwendet, die wie in Abbildung 5-2 dargestellt, abgebildet werden.

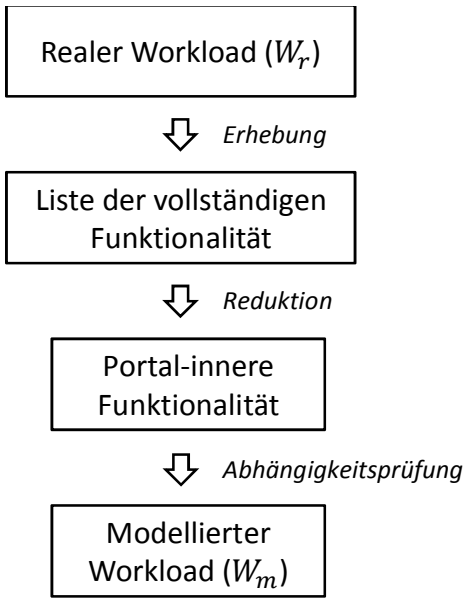


Abbildung 5-2: Ermittlung des modellierten Workloads
Quelle: eigene Darstellung in Anlehnung an Jehle (2010, 58)

Als Grundlage für den modellierten Workload dient die im vorangegangenen Kapitel beschriebene Fallstudie, die im Rahmen des SAP-University-Alliances-Programms zu Schulungszwecken eingesetzt wird. Die Betrachtung des Reaktionsverhaltens eines Portalsystems innerhalb einer Schulung zeigt ein erwartetes Muster mit erhöhten Lastspitzen (vgl. Jehle 2010, 62), wenngleich das Nutzungsprofil der Schulungssysteme im Allgemeinen keinen direkten Vergleich mit operativ betriebenen Systemen zulässt (Mohr/Simon/Krcmar 2005, 4f.). Für die Evaluation des in Kapitel 4 vorgestellten Modells mittels Simulation werden folgende besonderen Gegebenheiten des Workloads genutzt:

- Die Anzahl sowie Reihenfolge der durchgeführten Portaloperationen ist statisch und ergibt sich aus der Analyse der Schulungsunterlagen. Somit reduziert sich die Modellierung der Benutzer (siehe Kapitel 4.2.1) auf Benutzertypen mit unterschiedlichen Denkzeiten. Die Interaktionsschritte sowie deren Abfolge sind identisch.

- Die Schulungsinhalte beziehen sich, bis auf wenige Ausnahmen, auf portaleigene Funktionen. Somit bleibt trotz Reduktion ein Großteil des realen Workloads erhalten. Integrierende Funktionen, wie beispielsweise das Einbinden von ABAP-basierten ERP-Modulen, würden die Modellierung der externen Komponenten oder in der minimalen Ausprägung die implizite Angabe der Antwortzeit der externen Komponente

in der Bearbeitungszeit des Lastschrittes erfordern. Dies hätte zur Folge, dass die zusätzlichen (äußeren) Einflüsse einen weiteren, zu analysierenden Störfaktor für die Evaluation der Simulationsergebnisse darstellen würden.

Aus der Transkription der Schulungsinhalte in einzelne Operationen der Schulungsteilnehmer ergibt sich eine Liste von über 100 durchgeführten Funktionen (siehe Tabelle 7-1). Neben dem Ausschluss von integrierenden Funktionen müssen die Voraussetzungen für bestimmte Portaloperationen erfüllt sein, so besteht zum Beispiel eine Abhängigkeit zwischen dem Löschen und vorausgehenden Erstellen eines Benutzerkontos. Diese Abhängigkeit ergibt sich aus der Abfolge der in der Schulung durchgeführten Portaloperationen. Bei einzelnen Lastschritt-Messungen oder bei Portaloperationen, die von einer ausgeschlossenen Portalfunktion abhängen, muss sichergestellt sein, dass die Voraussetzungen erfüllt sind.

Zudem muss geprüft werden, ob die durchgeführten Portalfunktionen parallel ausgeführt werden können. Die Untersuchung dieser Eigenschaft erfolgt manuell, indem die betrachtete Funktion zeitgleich von zwei verschiedenen Benutzern ausgeführt wird (vgl. Jehle 2010, 66).

Nicht parallel durchführbare Funktionen spielen bei der Lasterzeugung eine Rolle, da ein entsprechender Automatismus dafür sorgen muss, dass ein paralleler Aufruf unterbleibt, damit der Lastschritt durchgeführt wird und die benötigten Daten aufgezeichnet werden können. Ebenfalls ist im SAP-Netweaver-Portal-System eine Mehrfachanmeldung eines einzelnen Benutzerkontos nicht möglich, sodass bei der Lasterzeugung unterschiedliche Konten verwendet werden müssen (Gootzit 2008).

Neben der Lasterzeugung muss auch bei der Simulation sichergestellt werden, dass das Verhalten dem realen System entspricht. Funktionen, die nicht parallel ausgeführt werden können, werden vom System so lange gesperrt, bis der Benutzer den Lastschritt durchgeführt hat. Sollte ein weiterer Benutzer versuchen, den Vorgang durchzuführen, würde eine Fehlermeldung darauf hinweisen, dass die Funktion derzeit gesperrt ist. Die Sperrung der Funktion kann im Simulationsmodell über einen Binärsemaphor abgebildet werden (siehe Abbildung 5-3). Dazu muss ein synchroner Aufruf der Wait-Entry des Binärsemaphors erfolgen und nach Beendigung über einen Aufruf der Signal-Entry in Phase 2 (also nach dem Senden der Antwort an den Benutzer) wieder freigegeben werden.

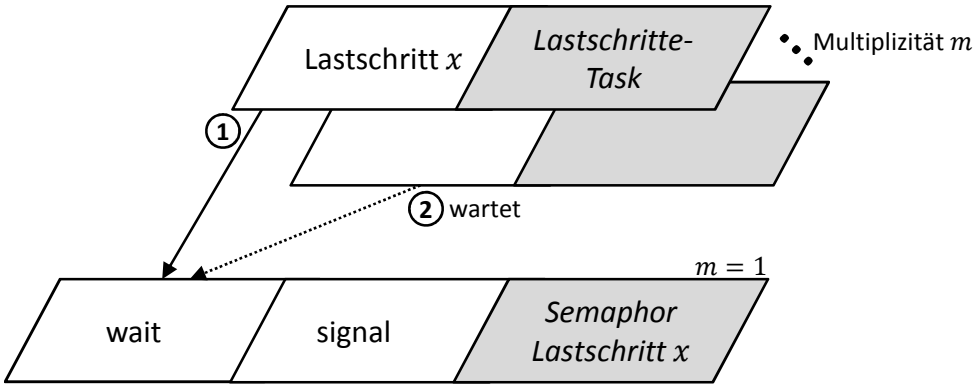


Abbildung 5-3: Sequentielle Abarbeitung bestimmter Lastschritte
Quelle: eigene Darstellung

Abschließend ist zu prüfen, welche Portaloperationen mehrfach durchgeführt werden, da auf eine doppelte Parametrisierung desselben Lastschrittes verzichtet werden kann (vgl. Jehle 2010, 65). Dementsprechend reduziert sich die vollständige Liste der Portaloperationen, unter Berücksichtigung der genannten Aspekte, auf die in Tabelle 5-1 aufgelisteten Funktionen. Mehrfach durchgeführte Operationen werden in dieser Darstellung nicht aufgeführt.

ID	**Funktion**	**Einzeln wiederholbar**	**Parallel nutzbar**
Tag 1 – Vorbereitungen			
1.0	Portal-Seite öffnen	X	X
1.1	Logon Admin		X
1.2	User anlegen	X	X
1.3	Gruppe anlegen	X	X
1.4	User zu Gruppen hinzufügen		X
1.5	Ordner anlegen	X	X
1.6	Rolle anlegen	X	X
1.7	Objekt (Rolle) öffnen		X
1.8	Einstiegspunkt aktivieren		X
1.9	Rolle zu Gruppe zuordnen		X
1.10	Rolle anpassen	X	X
1.11	Leserechte für Portal-Content vergeben	X	X
1.12	Schreibrechte für Ordner anlegen	X	X
1.13	Kurskonfiguration prüfen	X	X
1.14	Logout	X	X
Tag 2 – Login, Navigation, Personalisierung, Menü			
2.1	Initiales Passwort ändern	X	X
2.4	Zur Startseite wechseln		X
2.5	Personalisieren: Darstellungsschema ändern	X	X
2.6	Personalisieren: Darstellungsschema speichern		X
2.8	Detaillierte Benutzerinformationen eintragen	X	X
2.9	Top-Level-Navigation: Unterpunkte aufrufen	X	X
2.10	Startseite: Portal-Information abrufen	X	X
2.11	Content Management: Direct-Links zu Komponenten	X	X
2.12	Kollaboration: Grundlage für asynchrone Zusammenarbeit	X	X
2.13	Curriculum-Congress: Personalisierter Inhalt	X	X
2.15	History-Funktion: Wechseln zu Curriculum-Congress	X	X
2.17	Navigation: Überblick über „field areas“	X	X
Tag 3 - Inhaltsverwaltung			
3.1	Vorschau SAP-iViews	X	X
3.3	iView in neuen Ordner kopieren	X	X
3.4	Neues iView anlegen	X	X

3.5	Neues iView konfigurieren		X
3.6	Portal-Webseite erstellen	X	X
3.7	Portal-Webseite konfigurieren		X
3.8	neues Workset erstellen	X	X
3.9	neues Workset konfigurieren		X
3.10	Webseite mit Workset verlinken (Delta-Link)		X
3.12	Workset mit Rolle verlinken (Delta-Link)		
3.13	Workset schließen		
3.14	Rolle „Class role“ aktualisieren	X	X
Tag 5 – Kollaboration			
5.1	SAP-Meeting-Room erstellen	X	X
5.2	SAP-Meeting-Room konfigurieren		X
5.3	Benutzer/Gruppen zum SAP-Meeting-Room hinzufügen		X
5.4	Änderungen abspeichern		X
5.5	„Room Directory“ aufrufen	X	X
5.6	Eigenen Meeting-Room aufrufen	X	X
5.7	Eigenen Task anlegen (Single-Step)	X	X
5.8	Collaboration-Launch-Pad (CLP) aufrufen	X	X
5.9	User zum CLP hinzufügen		X
Tag 6 – Öffentliche Dokumente, Versionierung			
6.2	Öffentliches Dokument erstellen	X	X
6.3	Inhalt für öffentliches Dokument eingeben	X	X
6.4	Öffentliches Dokument versionieren		X
6.5	Öffentlichen Ordner öffnen	X	X
Tag 7 – Benutzerverwaltung und Sicherheit			
7.10	Benutzerpasswort ändern	X	X
X – Nachbereitungen			
X.1	Benutzer löschen		X
X.2	Gruppen löschen		X
X.3	Ordner löschen		X

Tabelle 5-1: Konsolidierte Liste der modellierten Portalfunktionen
Quelle: eigene Darstellung in Anlehnung an Jehle (2010, 66f.)

Die aufgeführten Portalfunktionen bilden den Zustand des Portalsystems im Schulungsbetrieb ausreichend ab (Jehle 2010, 68). Da sich die nicht parallel durchführbaren Operationen sich auf zwei Funktionen beschränken (ID 3.12 sowie 3.13), wurde bei der Lasterzeugung sowie der Workload-Modellierung auf diese Funktionen verzichtet, da zum einen eine mögliche zeitliche Verfälschung zwischen der sequentiellen Abarbeitung am System und der semaphorgesteuerten Abarbeitung bei der Simulation analysiert werden müsste, zum anderen die Integration dieser Funktionen keinen Mehrwert bei der Analyse des Antwortzeitverhaltens mit sich bringen würde.

5.3 Lasterzeugung

Für die Erhebung der Leistungsdaten ist es notwendig, die gewünschte Last am Portalsystem zu erzeugen. Durch den Einsatz eines Programmes zur automatischen Lastgenerierung kann zum einen eine deterministische Ausführung gewährleistet werden, zum anderen erleichtert die Automatisierung im Regelfall die Durchführung der Lastschritte. Die Herausforderung liegt dabei vor allem bei der Interaktion mit dem zu untersuchenden Objekt, da auf dynamisch generierte Inhalte entsprechend reagiert werden muss (Wassermann et al. 2008).

Bei der Durchführung der Messungen ist darauf zu achten, dass der Betrieb des Portalsystems durch zusätzliche Programme, beispielsweise zur Generierung der Last oder zum Aufzeichnen der Leistungsdaten, nicht beeinflusst bzw. nur durch ressourcenschonende Werkzeuge beobachtet wird (vgl. bspw. intrusives Monitoring auf Betriebssystemebene in Kapitel 3.3).

Da die in dieser Arbeit durchgeführte Interaktion mit dem Portalsystem ausschließlich über die HTTP-Schnittstelle erfolgt, kann die Applikation auf der Präsentations- bzw. Benutzerebene, d.h. der Web-Browser, durch ein Programm ersetzt werden, das die manuelle Benutzerinterkation mittels vorkonfigurierter Interaktionsschritte nachbildet. Dabei werden die Anfragen künstlich generiert und ein entsprechender http-Request an den Web-Server, in diesem Falle das Portalsystem, gesendet. Der technische Aufwand zur Implementierung eines automatischen Lastgenerators auf HTTP-Basis ist verhältnismäßig gering und führt daher zu einer nahezu unüberschaubaren Menge an angebotener Software (Hower 2009).

Die verschiedenen Werkzeuge unterscheiden sich jedoch in ihrer Zielsetzung, Funktionalität und Konfigurierbarkeit erheblich, sodass die Eignung eines Lastgenerators stark von dem zu untersuchenden Objekt abhängt (Jianyi/Zhengqiu 2008). Beispielsweise sind nicht alle Werkzeuge in der Lage, auf dynamische Rückmeldungen adäquat zu reagieren. Die Benutzerschnittstelle des SAP-Netweaver-Portals weist einige Charakteristika auf, die von einem Lastgenerator berücksichtigt werden müssen und daher im Folgenden vorgestellt werden. Im Anschluss wird auf den verwendeten Lastgenerator eingegangen.

5.3.1 Charakteristiken der Benutzerschnittstelle

Die Web-Schnittstelle des SAP-Netweaver-Portal-Systems bedient sich verschiedener Erweiterungen, die nicht in den Bereich der klassischen HTML-Spezifikation (W3C 2011) fallen. Die zusätzlich verwendeten Technologien, zu denen JavaScript, AJAX (Eichorn 2006) und dynamisches HTML (DHTML, Klein 2000) zählen, ermöglichen zum einen zusätzliche Funktionen der Benutzerschnittstelle, zum anderen wird die Datenmenge, die zwischen Benutzer und Server gesendet wird, reduziert. Allerdings führt die Verwendung dieser Technologien auch dazu, dass nicht alle Web-Browser vom SAP-Netweaver-Portal-System unterstützt werden.

Das prominenteste Beispiel der Zusatzfunktionen im SAP-Netweaver-Portal-System ist die Verwendung von Kontextmenüs, die über die rechte Maustaste aufgerufen werden. Die darin angebotenen Funktionen sind kontextspezifisch und ermöglichen eine von „Microsoft Windows“ bekannte Handhabung, durch einen Rechtsklick auf das Objekt kontextbezogene Aktionen angeboten zu bekommen. Die Schwierigkeit bei der Behandlung dieser nichtstandardisierten Schnittstellen ist vor allem der ordnungsgemäße Aufruf der Funktion, die über ein solches Element angesteuert wird.

Die Beschränkung auf einige wenige unterstützte Web-Browser aufgrund der Verwendung nicht-standardisierter Technologien wirkt sich auch auf die Lastgeneratoren aus. Diese müssen mit den dynamischen Elementen der Benutzerschnittstelle umgehen können. Von einer ausreichenden Kompatibilität kann jedoch nur ausgegangen werden, wenn der Hersteller das entsprechende Produkt freigibt. Ansonsten können auch nur kleine Veränderungen, die bei einem Versionssprung vorgenommen werden, zu einem Kompatibilitätsbruch des verwendeten Lastgenerators führen.

Da SAP als Benutzerschnittstelle für das Portal lediglich den Web-Browser nennt, liegt es nahe, den Browser als Vermittler zwischen Lastgenerator und Portalsystem zu verwenden. Dazu kann eine vom „Microsoft Internet Explorer“ (Microsoft 2011) oder auch von „Mozilla Firefox“ angebotene Schnittstelle (Feldt 2007) genutzt werden, die es erlaubt, automatisiert bzw. programmgesteuert Benutzerfunktionen aufzurufen.

5.3.2 Lastgenerator

Der in dieser Arbeit verwendete Lastgenerator basiert auf dem in einer vorangegangenen Dissertation (Jehle 2010) entwickelten, java-basierten Performance-Evaluation-Cockpit für ERP-Systeme (PEER). Für eine ausführliche Darstellung der Architektur sowie Validierung der funktionalen Aspekte sei auf die entsprechende Arbeit verwiesen. Zusammenfassend lassen sich die Eigenschaften von PEER wie folgt beschreiben:

- *Grafische Benutzeroberfläche*: PEER bietet eine grafische Benutzeroberfläche, von der aus die Tests geplant, konfiguriert und ausgeführt werden können.

- *Lasterzeugung*: PEER verwendet das von iOpus entwickelte Werkzeug iMacros (iOpus 2012) zur Wiedergabe von den im Web-Browser aufgezeichneten Benutzeraktionen.

- *Darstellung der Ergebnisse*: Neben der Lasterzeugung dient PEER auch der Aufzeichnung von Leistungsdaten. Dabei werden im Black-Box-Ansatz die Antwortzeiten der einzelnen Interaktionsschritte festgehalten und aggregiert dargestellt.

- *Einheitliche Schnittstellen*: Aufgrund der modularen Architektur und der Verwendung von einheitlichen Schnittstellen ist es problemlos möglich, eigene Lastmodule einzubinden.

Die modulare Architektur von PEER wird durch die Trennung der grafischen Oberfläche (GUI), der Ausführungseinheit und dem Konfigurator realisiert. Da die Aufzeichnung von Black-Box-Messdaten und die Darstellung der Ergebnisse für diese Arbeit nicht genutzt werden, sind lediglich die bereitgestellten Lastmodule relevant, auf denen aufbauend der für diese Arbeit verwendete Workload umgesetzt werden kann.

Nichtsdestotrotz soll der Testtreiber die Kompatibilität zur Verwaltungseinheit von PEER beibehalten. Daher wird das von PEER geforderte Interface „PerformanceDriver“ verwendet. Das im Listing 5-1 dargestellte Interface gibt dabei die abstrakte Definition der Methoden vor, die von dem Lasttreiber implementiert werden müssen:

```
1  public interface PerformanceDriver {
2     public String doInit();
3     public String doExec();
4     public String doCleanup();
5     public PerformanceStatus getStatus();
6     public String abortRun();
7  }
```

Listing 5-1: Implementiertes PEER-Interface
Quelle: Jehle (2010, 137)

Da verschiedene Lastmodule bzw. Testtreiber unterschiedliche Parameter erfordern (beispielsweise das zu setzende Passwort oder den Namen des iViews), werden in PEER sogenannte „Configurables“ eingesetzt, die beim Einbinden des Testtreibers durch den „Class Loader“ übernommen und nicht durch interne Bezeichnungen ersetzt werden. Dadurch ist es möglich, dass die Verwaltungseinheit von PEER die Variablennamen sowie –typen zur Laufzeit auswerten kann. Auch diese Architekturforderung wird übernommen, damit ein Einbinden in die PEER-Oberfläche möglich bleibt.

Der interne Aufbau des Testtreibers für die Portalfallstudie ist in Abbildung 5-4 dargestellt und wird im Folgenden kurz beschrieben:

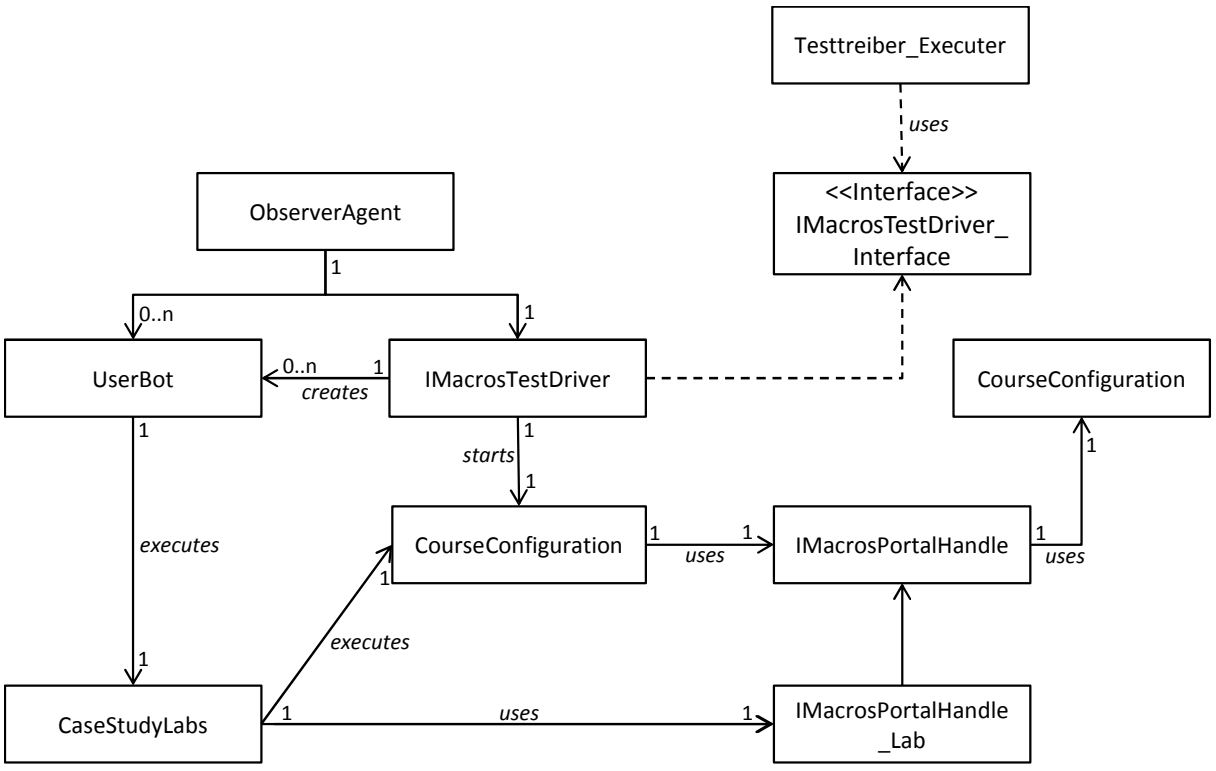


Abbildung 5-4: Schematische Darstellung des internen Testtreiber-Aufbaus
Quelle: Jehle (2010, 126)

– *Testtreiber_Executer*: Die Klasse „Testtreiber_Executer“ übergibt die benötigten Parameter des Testtreibers und steuert dessen Ausführung.

- *iMacrosTestDrive*r: Diese Klasse ist ein „Singleton“ (Gamma et al. 1995) und stellt die zentrale Komponente des Testtreibers dar. Sie steuert die Ausführung des Lastszenarios und erstellt die notwendige Anzahl an Benutzern mittels Instantiierung der Klasse „UserBot“. Zudem wird sie über den Verlauf der durchgeführten Aktionen über die Klasse „ObserverAgent“ benachrichtigt und leitet gegebenenfalls die nachfolgenden Schritte ein.

- *CourseConfiguration*: Wie bereits bei der Beschreibung der Fallstudie in Kapitel 5.1 dargestellt, sind Vorbereitungen nötig, die der Dozent vor den eigentlichen Übungseinheiten vornimmt. Dazu zählen unter anderem die Erstellung der Schulungsordner sowie das Anlegen der Benutzer, die von den Schulungsteilnehmern verwendet werden. Die Klasse „CourseConfiguration“ übernimmt die Initialisierung der benötigten Objekte für die Durchführung der Übungseinheiten und verwendet für die Kommunikation mit dem Portalsystem die ausgelagerte Klasse „iMacrosPortalHandle“.

- *iMacrosPortalHandle*: In dieser Klasse sind alle grundlegenden Interaktionen mit dem Portal implementiert. Die Wrapper-Klasse „iMacrosWrapper“ ermöglicht dabei die Kommunikation mit dem iMacros-Browser. Für die Ausführung der Interaktionsschritte sind hier unterstützende Methoden implementiert, die die Navigation durch den Portal-Content-Katalog, die Steuerung der Kontextmenüs sowie das Erkennen von Textausgaben und Bildern realisieren. Zudem wird hier der iMacros-Browser initialisiert und beendet.

- *UserBot*: Die Klasse „UserBot“ implementiert das Interface „Runable“ der Klasse „Thread“. Jede Instanz dieser Klasse repräsentiert einen Portal-Benutzer bzw. Schulungsteilnehmer.

- *ObserverAgent*: Auch diese Klasse ist ein Singleton und Bestandteil des Observer-Patterns (Gamma et al. 1995). Sie beobachtet ein „UserBot“-Objekt und benachrichtigt den „iMacrosTestDriver“, damit dieser die nachfolgenden Schritte einleiten kann.

- *CaseStudyLabs, iMacrosPortalHandle_Labs*: Als Case-Study-Labs werden die einzelnen Tage der Portal-Schulung bezeichnet. Jede Übungseinheit entspricht einem „Lab“. In der Klasse „CaseStudyLabs“ sind Funktionen für die Initialisierung und Ausführung der Interaktionsschritte implementiert. Die Klasse „iMacrosPortalHandle_Labs“ erweitert die Superklasse „CaseStudyLabs“ und beinhaltet die Implementierung der Makroinitialisierung sowie -ausführung.

Das Interface „iMacrosTestDriver_Interface“ definiert die benötigten Methoden bzw. Parameterübergabe zur Umsetzung der Portalfallstudie. Der Aufruf der Methoden erfolgt über den übergeordneten „Testdriver_Executer“, der aufgrund des Verzichts der PEER-GUI auch die Einstiegskomponente darstellt. Die umzusetzenden Methoden sind in Abbildung 5-5 dargestellt und werden im Folgenden kurz erläutert:

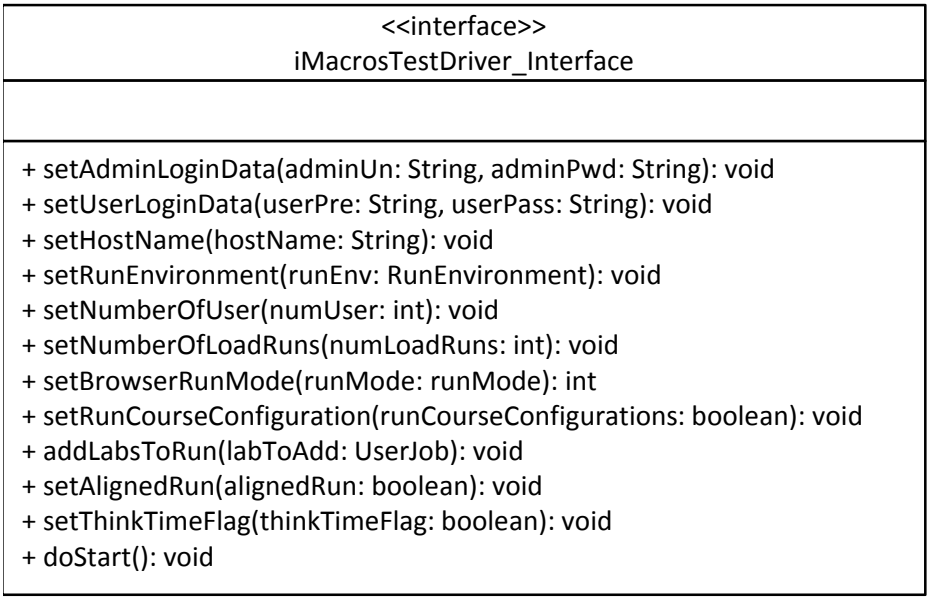


Abbildung 5-5: Testtreiber-Interface für die Portalfallstudie
Quelle: eigene Darstellung in Anlehnung an Jehle (2010, 124)

- *setAdminLoginData*: Hiermit wird das Administratorkennwort übergeben, das den Testtreiber erlaubt, die nötigen administrativen Aufgaben durchzuführen, wie zum Beispiel die Erstellung von Fallstudien-Benutzern.

- *setUserLoginData*: Das mit dieser Methode übergebene Präfix wird allen Benutzern und erstellten Objekten vorangestellt. Dadurch ist es möglich, sämtlichen generierten Inhalt einem Durchlauf zuzuordnen und gegebenenfalls zu löschen.

- *setHostName*: Wie sich bereits aus dem Namen der Methode erschließt, wird hiermit die Adresse des Servers festgelegt.

- *setRunEnvironment*: Ein Testtreiber kann diverse Konfigurationen bereitstellen, die unterschiedliche Systemeigenschaften und Lastszenarien abdecken. Mit diesem Parameter wird die zu verwendende Konfiguration ausgewählt.

- *setNumberOfUser*: Damit wird die Anzahl der Benutzer festgelegt, die die definierten Lastschritte parallel durchführen.

- *setNumberOfLoadRuns*: Hintereinander geschaltete Durchläufe können mit diesem Parameter auch auf der Ebene des Testtreibers definiert werden. Dies ist vor allem dann sinnvoll, wenn ein vorausgehender Durchlauf das System einpegeln (siehe auch Einschwingverhalten in Kapitel 5.5.1) soll.

- *setBrowserRunMod*e: Die Ausführung des iMacros-Browser kann sichtbar oder im sogenannten Tray-Modus erfolgen. Da bei der Lasterzeugung der Browser nicht sichtbar sein soll, um Ressourcen zu schonen, wird über diesen Methodenaufruf der Tray-Modus aktiviert.

- *setRunCourseConfiguration:* Die bereits angesprochene Vorkonfiguration zur Durchführung der Fallstudie kann hiermit aktiviert werden.

- *addLabsToRun:* Aufgrund des modularen Aufbaus der Fallstudienumsetzung in einzelne Labs können hiermit die zu verwendenden Übungseinheiten definiert werden. Für die gesamte Abbildung der Portalfallstudie werden sämtliche Labs hinzugefügt.

- *setAlignedRun*: Es besteht die Möglichkeit, nach jedem Interaktionsschritt die Benutzer zu synchronisieren und damit den Verlauf einer Übungseinheit zu simulieren, bei der der Dozent nach jedem Schritt wartet, bis die Teilnehmer die Aufgaben durchgeführt haben. Für die in dieser Arbeit durchgeführten Messungen wird diese Funktion nicht verwendet und daher auf „false“ gesetzt.

- *setThinkTimeFlag*: PEER sieht für die Benutzerdenkzeiten auf der Ebene des Testtreibers keine entsprechende Konfigurationsmöglichkeit vor. Die vorgeschlagene Umsetzung erfolgt lediglich auf der Ebene der Testlaufkomposition, bei der ein Pseudo-Testtreiber zwischen den einzelnen Testtreibern die Denkzeit des Benutzers simuliert (vgl. Jehle 2010, 143). Daher musste der Portalfallstudien-Testtreiber um eine Denkzeitkomponente erweitert werden. Mit diesem Methodenaufruf kann die Verwendung von Denkzeiten zwischen den Interaktionsschritten aktiviert werden, die über die bereitgestellte Methode „Thread.sleep()“ realisiert wird.

- *doStart*: Sobald alle Parameter durch die soeben vorgestellten Methodenaufrufe gesetzt wurden, kann der Testlauf mit dieser Methode gestartet werden.

Mit den genannten Erweiterungen des Lastmoduls sowie den notwendigen Anpassungen für die verwendete Portal-Version kann das Portal mit dem vorgestellten Workload unter Last gesetzt werden. Dabei erfolgt die Aufzeichnung der Leistungsdaten (siehe Kapitel 3.2 und Kapitel 3.3), die für die Parametrisierung des LQN-Modells (siehe Kapitel 4.2) sowie den Vergleich von simulierten und gemessenen Werten notwendig sind.

5.4 Szenarien

Eine gängige Methode für die Performance-Analyse von Rechnersystemen ist der Vergleich von Alternativen (Lilja 2000, 61f.). Dieser Vergleich dient meist der direkten Vorher-Nachher-Analyse von unterschiedlichen Systemen bzw. Systemkonfigurationen und erfolgt aufgrund der inhärenten Fehlerquote in den Messungen über statistische Verfahren. In dieser Arbeit wird kein direkter Vergleich im Sinne einer Vorher-Nachher-Analyse angestrebt. Die beiden Szenarien verfolgen das Ziel, unterschiedliche Lastsituationen zu untersuchen. Im ersten Szenario wird nur das SAP-Netweaver-Portal-System unter eine hohe Last gesetzt, wo hingegen im zweiten Szenario auch die Systemressourcen (des Hosts) ausgereizt werden.

Im ersten Szenario wird somit die grundsätzliche Eignung des LQN-Modells bezüglich des Antwortzeitverhaltens des Portalsystems getestet. Im zweiten Szenario werden die Rahmenbedingungen verschärft und Systemressourcen sowie Tabellenpuffer reduziert. Beiden Szenarien gleich sind die verwendete Hardware-Architektur und das zugrunde liegende Datenbanksystem. Das Testsystem ist als virtueller Host auf einem IBM-Power-750-Server installiert. Dies ermöglicht eine dynamische Konfiguration der Systemressourcen zur Laufzeit.

Als Betriebssystem wird AIX verwendet, der Hauptspeicher ist mit einem AME-Faktor (vgl. Kapitel 3.3.2) von 1,3 konfiguriert. Das Datenbanksystem ist auf einem eigenen virtuellen

Host installiert, damit der Server der Applikationsschicht nicht beeinflusst wird. Die CPU-Ressource des Datenbanksystems ist somit völlig unabhängig vom betrachteten Applikationsserver. Da die Datenbank lediglich als Black-Box betrachtet und nicht näher analysiert wird, werden dem Datenbank-Host ausreichend Systemressourcen zur Verfügung gestellt.

5.4.1 Szenario 1 – Ausreichende Systemressourcen

Im ersten Szenario werden, wie bereits erwähnt, die Systemressourcen in ausreichendem Maße vergeben (siehe Abbildung 5-6). Zum einen werden die CPU-Ressourcen nicht gekappt, damit der zunehmende Bedarf bei steigender Benutzeranzahl beobachtet werden kann. Ebenso werden die Tabellenpuffer ausreichend dimensioniert, sodass es nicht zu Verdrängungen kommt. Dies reduziert die Weiterleitungswahrscheinlichkeit im Tabellenpuffer-Task und somit dessen Einfluss auf das Antwortzeitverhalten.

Das erste Szenario stellt folglich ein Basisszenario dar, in dem auf erhöhte Einflüsse der Ressourcenknappheit verzichtet wird. Primäres Ziel dieses Szenarios ist die Überprüfung des Systemverhaltens unter optimierten Bedingungen sowie der Vergleich der Daten mit den erzielten Simulationsergebnissen.

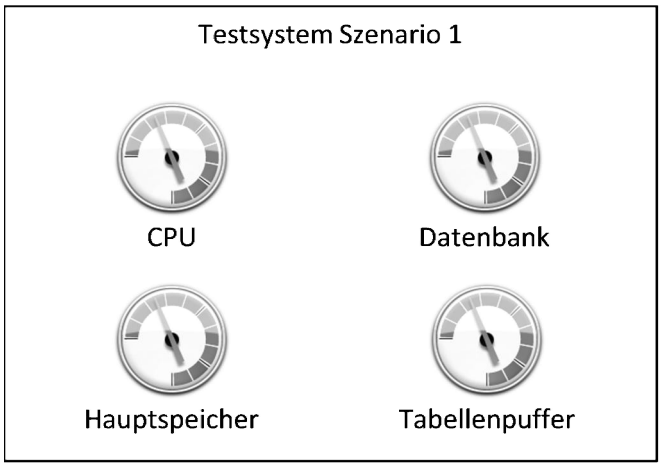


Abbildung 5-6: Ressourcenzuweisung im ersten Test-Szenario
Quelle: eigene Darstellung

5.4.2 Szenario 2 – Knappe Systemressourcen

Im zweiten Szenario werden gezielt Systemressourcen reduziert und somit zusätzliche Einflüsse auf die Antwortzeit geprüft (siehe Abbildung 5-7):

- Der Hauptspeicher ist ausreichend, da unzureichender Hauptspeicher zu einem Out-of-Memory-Fehler oder zumindest zur Speicherauslagerung und somit zu erheblichen Performance-Verlusten führt, die im Java-Umfeld grundsätzlich vermieden werden müssen. Allerdings erreicht die Hauptspeicherauslastung einen hohen Wert, sodass der Garbage-Collector in zunehmendem Maße für freie Speicherbereiche sorgen muss.

- Die Datenbank-Ressourcen werden auf einem hohen Niveau belassen, sodass keine zusätzlichen Nebeneffekte eintreten, da die Datenbank-Komponente im LQN-Modell lediglich als Black-Box betrachtet wird.

- Die CPU-Ressourcen werden unterhalb des im ersten Szenario gemessenen maximalen Verbrauchs gesetzt. Dadurch können die modellierten CPU-Ressourcen und deren Einfluss auf die Antwortzeit intensiver geprüft werden.

- Auch wird die Größe des Tabellenpuffers reduziert, sodass es zu Verdrängungen kommt, deren Einfluss mit den Ergebnissen der Simulation verglichen werden kann.

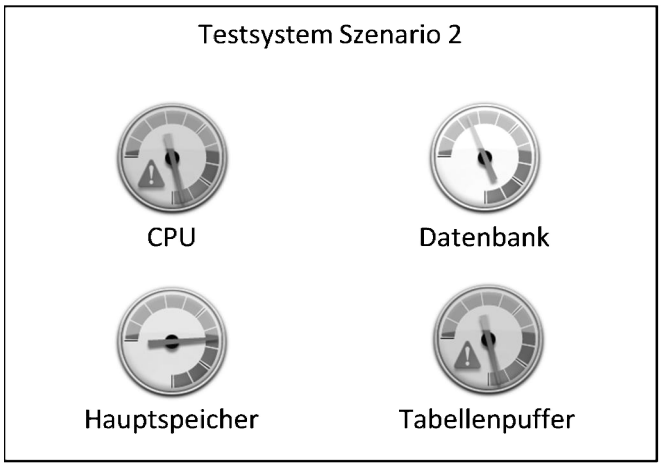


Abbildung 5-7: Ressourcenzuweisung im zweiten Test-Szenario
Quelle: eigene Darstellung

Das zweite Szenario ermöglicht somit die Untersuchung des Performance-Verhaltens im Hochlastbereich, bei dem sowohl das SAP-Portal-System, als auch die Systemressourcen (CPU) an ihre Grenzen gelangen.

Aus den beiden Szenarien ergeben sich folglich verschiedene Lastsituationen:

1. *Niedriglastbereich*: Sowohl das SAP-Portal-System, als auch die Systemressourcen verfügen über Reserven.

2. *Hohe Auslastung des SAP-Portal-Systems in Szenario 1*: Das Portalsystem wird über eine hohe Anzahl an simultan agierenden Benutzern und einer Einschränkung der Java-Server-Instanzen bzw. Applikations-Threads unter Last gesetzt.

3. *Hohe Auslastung des SAP-Portal-Systems und der Systemressourcen in Szenario 2*: Hierbei werden neben der Auslastung der Java-Server-Instanzen und der Verminderung der Tabellenpuffergröße, die zu einer zunehmenden Anzahl an Datenbankanfragen führt, auch die CPU-Ressourcen begrenzt und infolgedessen sowohl das SAP-Portal-System als auch der virtuelle Host unter Last gesetzt.

5.5 Messung

In diesem Kapitel werden zunächst das Einschwingverhalten sowie oszillierende Messwerte des Netweaver-Portal-Systems erläutert und anschließend die Messwiederholungen sowie Messergebnisse der beiden ersten Szenarien beschrieben.

5.5.1 Einschwingverhalten

Bei der Analyse von Ausreißern (siehe Kapitel 2.2.2.1) kann festgestellt werden, dass besonders bei den ersten Messreihen von wiederholt durchgeführten Lastsequenzen untypische Werte auftreten. Die Antwortzeiten nehmen in den ersten Durchläufen ab (siehe Abbildung 5-8) und stellen ein Einschwingverhalten dar, das unter anderem auf die Verwendung von Puffern zurückzuführen ist (Meyer/Guicking 1974). Das gewählte Beispiel entspricht der bei Jehle (2010, 167) verwendeten Benutzerinteraktion, bei der die Content-Administration aus der Top-Level-Navigation aufgerufen wird, und bestätigt die Minderung der Antwortzeit in den ersten 3 – 5 Durchläufen.

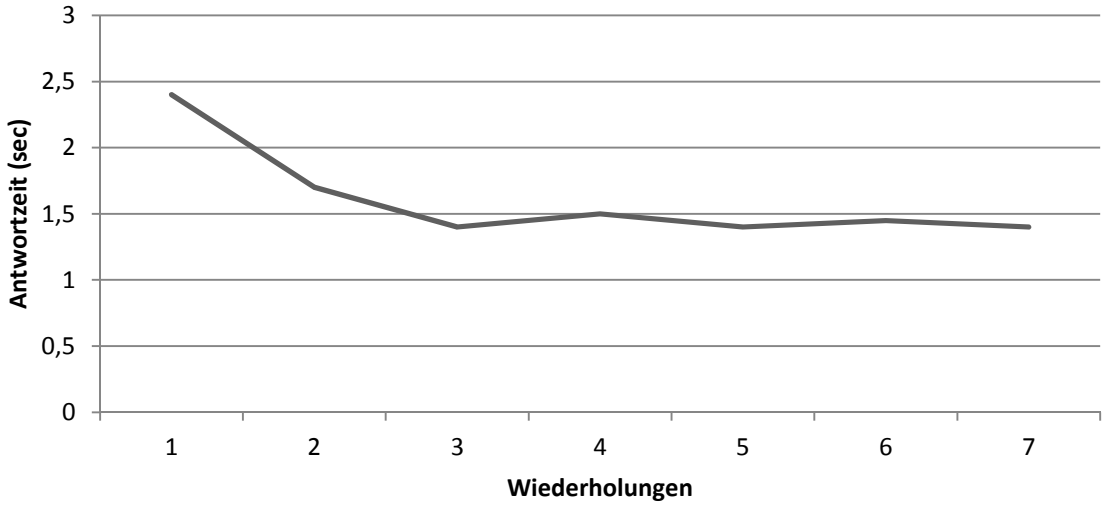


Abbildung 5-8: Einschwingverhalten am Beispiel einer Menü-Navigation
Quelle: eigene Darstellung

Die nötige Anzahl an Wiederholungen zur Stabilisierung der Antwortzeiten hängt von dem betrachteten System ab. Bei der Analyse von ERP-Systemen im Allgemeinen wird eine dreimalige Wiederholung der Lastsequenz empfohlen (Barham et al. 2003), bei einen SAP-Netweaver-Portal-System wurde ein Minimum von fünf Wiederholungen vorgeschlagen (Jehle 2010, 167). Auch die eigene Erhebung zeigt eine ähnliche Charakteristik und veranlasst somit eine Einschwingzeit von fünf Wiederholungen bei den in dieser Arbeit durchgeführten Performance-Messungen.

5.5.2 Oszillierende Messwerte

Bei der Messung von Leistungsdaten eines Portalsystems fällt auf, dass die Werte auch nach einer ausreichenden Einschwingzeit sowie etlichen Messwiederholungen keinen konstanten Wert erreichen (Jehle 2010, 168f.). In der Literatur werden hierfür verschiedene Gründe angegeben, beispielsweise „Page Faults“ (Parupudi/Winograd 1972) oder Multitasking-Mechanismen (Lilja 2000, 45). Zusammenfassend lassen sich die Ursachen mit nicht vorhersagbaren Einflüssen auf Systemebene beschreiben. Eine explizite Steuerung dieser Aktivitäten für die Leistungsmessung würde starke Veränderungen am System erfordern und in der Folge das untersuchende Objekt stark beeinflussen (Tanenbaum 2006). Diese Beeinflussung würde wiederum eine Übertragbarkeit der erhobenen Messwerte in Frage stellen (Baumgarten/Siegert 2007). Daher stehen keine nutzbaren Mittel zur Verfügung, die Effekte

der Oszillation der Messwerte zu verhindern. Nur durch die Anwendung statistischer Verfahren kann dieser Effekt gemindert werden.

Zur Veranschaulichung sei der Lastschrittaufruf aus Kapitel 4.2.6 näher betrachtet. Wie bereits dargestellt, wurde ein Lastschritt (Aufruf der Portalinformationen) nach 5 Einschwingdurchläufen automatisiert 3000 Mal aufgerufen. Bei dem Vergleich zwischen den ersten 30 und den letzten 30 Wiederholungen fällt auf, dass keine Stabilisierung der Bearbeitungszeit auftritt (siehe Abbildung 5-9). Folglich ist auch keine schwankungsärmere Antwortzeit möglich.

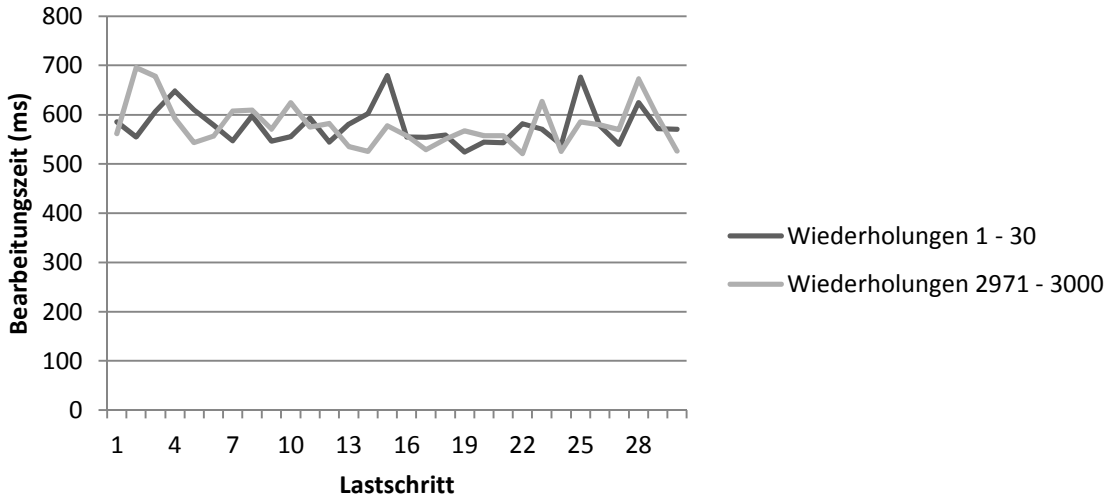


Abbildung 5-9: Oszillierende Messwerte
Quelle: eigene Darstellung

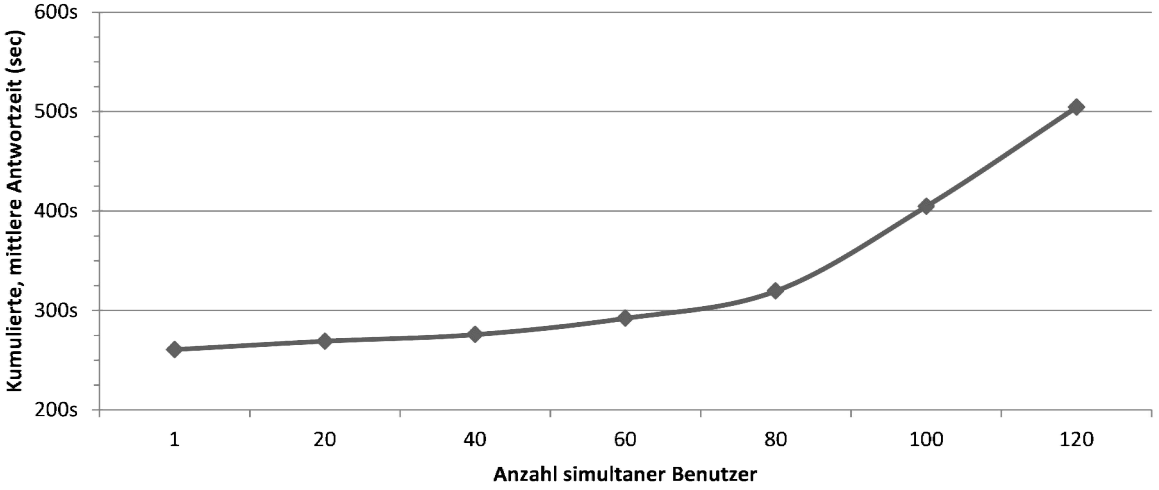
5.5.3 Messwiederholungen

Für die einzelnen Messreihen würde im Idealfall die Anzahl der Benutzer sukzessive um 1 erhöht werden. Zudem sollten aufgrund des zentralen Grenzwertsatzes mindestens 30 Wiederholungen erfolgen. Bei einer Grenzlast von 120 simultan agierenden Benutzern würde dies zu 120 mal 30 Durchläufen der (modellierten) Fallstudie führen. Bei einer geschätzten Dauer von ca. 200 bis 2000 Sekunden (je nach Anzahl der Benutzer) würde dies einer gesamten Ausführungszeit von 200 bis 2000 Stunden entsprechen. Daher wurde die Anzahl der Benutzer nicht jeweils um 1, sondern in 20er-Schritten erhöht. Bei 5 Wiederholungen zum Einschwingen und 30 gemessenen Wiederholungen führt dies zu 245 Durchläufen der Fallstudie und entspricht bei einer vorab geschätzten mittleren Ausführungszeit von 500 Sekunden einer Gesamtdauer von 34 Stunden.

5.5.4 Szenario 1 – Ausreichende Systemressourcen

Wie bereits dargestellt, werden im ersten Szenario die Systemressourcen in ausreichendem Maße bereitgestellt. Zu Beginn werden die kumulierten Antwortzeiten für die Durchführung der (modellierten) Portalfallstudie, in Abhängigkeit von einer unterschiedlichen Anzahl an simultan agierenden Benutzern, dargestellt. Im Anschluss werden verschiedene Komponenten sowie deren Einfluss (vgl. Kapitel 4.1.2) betrachtet.

5.5.4.1 Antwortzeiten



μ	260,725	269,180	275,775	292,156	319,540	404,587	504,311
σ	17,318	17,330	17,285	21,647	21,103	25,977	43,295
c_x	0,066	0,064	0,063	0,074	0,066	0,064	0,086
Median	254,590	262,999	269,860	284,487	312,620	395,384	488,974
Q1 (25%)	252,829	260,830	267,939	282,287	310,140	392,744	484,573
Q3 (75%)	262,573	270,900	277,504	294,466	322,556	407,359	508,932

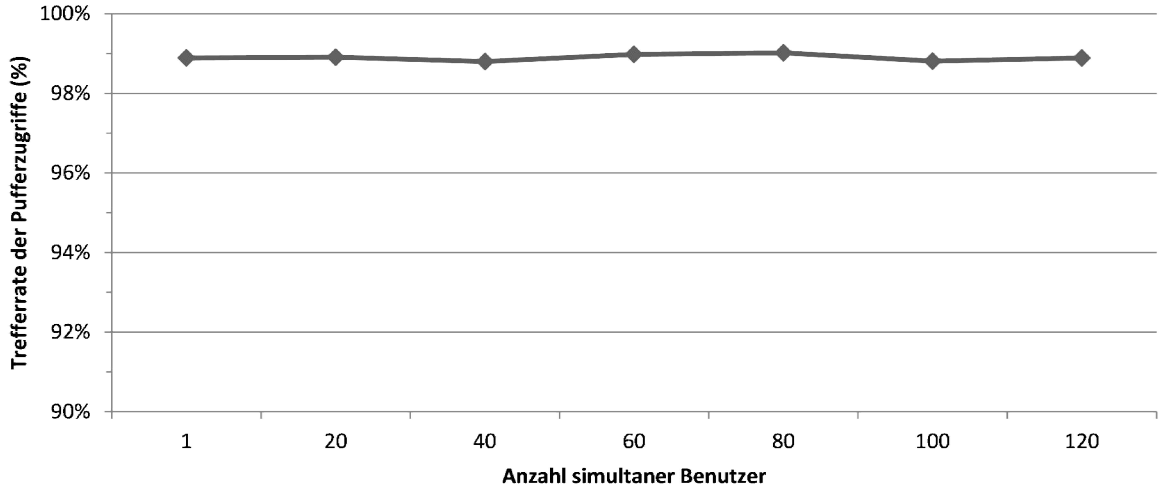
Abbildung 5-10: Kumulierte Antwortzeiten in Sekunden (Szenario 1)
Quelle: eigene Darstellung

In Abbildung 5-10 ist die Dauer, die für die Abarbeitung der Portalfallstudie benötigt wird, bei einer steigenden Anzahl von simultanen Benutzern dargestellt. Das aus den Grundannahmen abgeleitete Verhalten (vgl. Kapitel 4.1.1) lässt sich dabei deutlich erkennen:

- *Leichter Anstieg in der ersten Phase*: Die aggregierten Antwortzeiten steigen zunächst in geringem Maße an, bis eine Anzahl von ca. 80 gleichzeitig agierenden Benutzern erreicht wird. In dieser Phase sind noch genügend freie Applikations-Threads vorhanden. Der leichte Anstieg lässt sich mit zunehmenden Enqueue-Zeiten im Sperrmanagement erklären, da bei dem verwendeten Workload (mit identischen Interaktionsschritten der Schulungsteilnehmer) regelmäßig Sperrsituationen auftreten.

- *Starker Anstieg in der zweiten Phase*: Ab ca. 80 Benutzern ist ein deutlicher Anstieg der Antwortzeiten zu erkennen, der vor allem darauf zurückzuführen ist, dass die 80 Applikations-Threads (2 Java-Server-Instanzen zu je 40 Applikations-Threads) ausgelastet sind und zusätzliche Wartesituationen eintreten. Da die CPU-Ressourcen nicht gekappt sind, ist kein Einflusszuwachs des System-Overheads (vgl. Jehle 2010, 185f.) zu verzeichnen.

5.5.4.2 Pufferzugriffe



μ	98,89	98,91	98,80	98,98	99,02	98,81	98,89
σ	0,09	0,07	0,08	0,10	0,07	0,07	0,09
c_x	0,001	0,001	0,001	0,001	0,001	0,001	0,001
Median	98,90	98,91	98,81	98,98	99,02	98,81	98,90
Q1 (25%)	98,84	98,87	98,75	98,92	98,98	98,77	98,84
Q3 (75%)	98,94	98,95	98,84	99,03	99,05	98,85	98,94

Abbildung 5-11: Trefferrate der Pufferzugriffe in Prozent (Szenario 1)
Quelle: eigene Darstellung

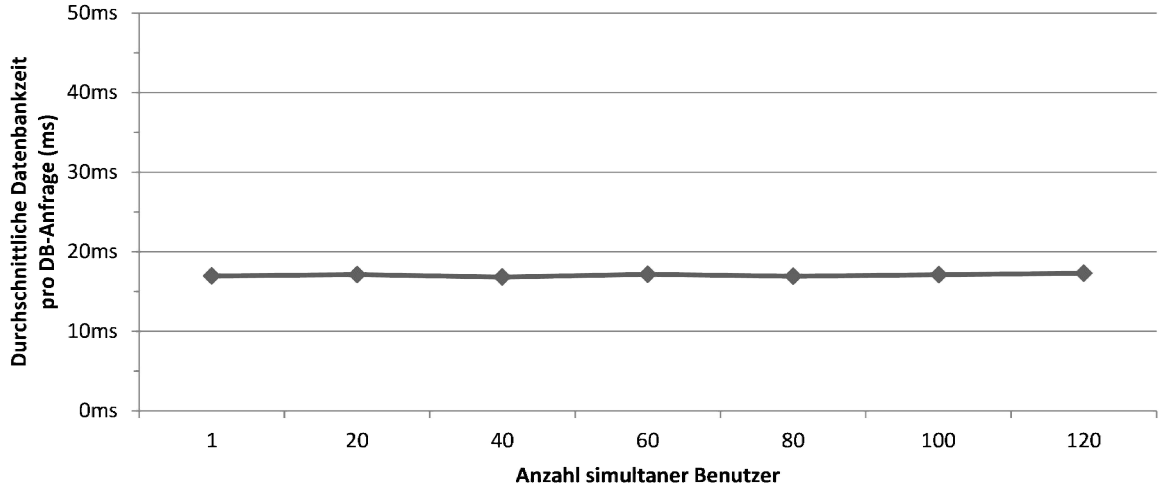
In Kapitel 4.1.2 konnten Invalidierungen und Verdrängungen der Tabellenpuffer-Inhalte als wesentliche Einflussgrößen der SAP-System-Performance identifiziert werden. Wie in Kapitel 3.1.4 dargestellt, lässt sich die Anzahl der Zugriffe auf den Puffer (*#Requests*) sowie die Anzahl der Treffer (*#Hits*) ablesen und folglich eine Trefferrate

$$Trefferrate = \frac{\#Hits}{\#Requests}$$

errechnen. In einem gut konfigurierten SAP-Netweaver-System gilt eine Trefferrate von 95% oder höher als befriedigend (Heiss/Veirich/Gratzl 2005, 355), oft werden in der Praxis sogar Werte um die 98 bis 99% erreicht.

Aufgrund der ausreichend dimensionierten Tabellenpuffergröße in diesem Szenario konnte eine Trefferrate von knapp 99% erreicht werden (siehe Abbildung 5-11) und folglich die Anzahl der nötigen Datenbankzugriffe auf ein Minimum reduziert werden. Es ist zudem ersichtlich, dass die Anzahl simultaner Benutzer keinen Einfluss auf die Trefferrate nimmt und es somit nicht zu einer erhöhten Anzahl an Invalidierungen und Verdrängungen kommt.

5.5.4.3 Datenbankanfragen



μ	16,962	17,138	16,822	17,159	16,930	17,114	17,307
σ	0,577	0,707	0,883	0,612	0,795	0,971	1,060
c_x	0,034	0,041	0,052	0,036	0,047	0,057	0,061
Median	16,974	17,171	16,863	17,216	16,967	17,160	17,356
Q1 (25%)	16,576	16,716	16,295	16,716	16,455	16,534	16,674
Q3 (75%)	17,265	17,503	17,279	17,503	17,341	17,616	17,854

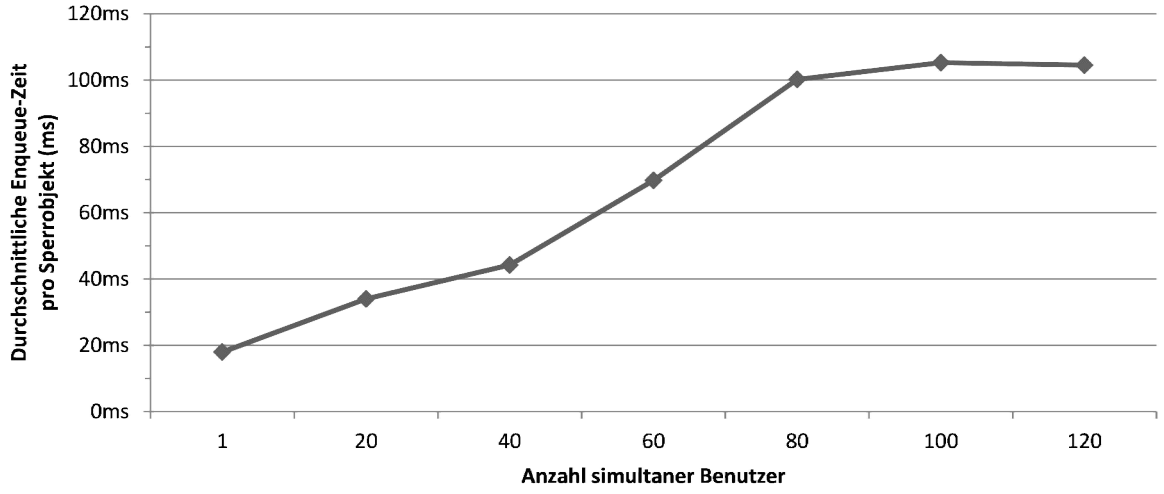
Abbildung 5-12: Durchschnittliche Datenbankzeit pro DB-Anfrage in Millisekunden (Szenario 1)
Quelle: eigene Darstellung

Neben den Pufferzugriffen soll geprüft werden, ob die für Datenbankanfragen benötigte Zeit – unabhängig von der Wartezeit auf eine potentielle Sperrfreigabe oder einer freien Datenbankverbindung – durch die Anzahl simultaner Benutzer beeinflusst wird. Dazu kann die Datenbankzeit, die im funktionalen Trace aufgezeichnet wird, ausgelesen werden. Diese stellt für das SAP-Netweaver-Portal-System eine Black-Box dar, da die Anfrage über den JDBC-Service an die Datenbank übergeben wird und die Zeit gemessen wird, bis die Antwort von der Datenbank zurückkommt. Die Datenbankzeit stellt somit nicht die reine Bearbeitungszeit am Datenbanksystem dar, sondern die zwischen dem Absender der Anfrage und dem Erhalt der Antwort verstrichene Zeit. Ebenso ist im LQN-Modell die Datenbank als Black-Box modelliert und aufgrund der eigenen CPU-Ressource unabhängig vom Portal-Server. Die Bearbeitungszeiten im LQN-Modell bilden die Datenbankzeiten des funktionalen Trace ab.

Es entsteht also die Notwendigkeit, dass dem Datenbanksystem ausreichende Systemressourcen zur Verfügung gestellt werden und das reale System der Annahme im LQN-Modell folgt, dass sich keine zusätzlichen, datenbankspezifischen Einflüsse auf die Antwortzeit auswirken. Dies ist in Bezug auf die Praxis eine offensichtlich idealisierte Betrachtungsweise, die sich direkt aus dem gewählten Black-Box-Ansatz ableitet.

In Abbildung 2-1 kann der konstante Verlauf der (durchschnittlichen) Datenbankzeiten bei einer steigenden Anzahl von Benutzern beobachtet werden. Die tatsächliche Zeit für eine (effektiv durchgeführte) Datenbankabfrage hängt somit lediglich von der potentiellen Wartezeit im Sperrmanagement und den zu Verfügung stehenden Datenbankverbindungen (10 pro Java-Server-Instanz, vgl. Kapitel 3.1.1) ab. Beide sind im LQN-Modell explizit über das Sperrmanagement sowie die Multiplizität des Datenbanktasks abgebildet.

5.5.4.4 Sperrmanagement



μ	17,984	34,004	44,218	69,735	100,217	105,279	104,539
σ	0,737	1,125	1,527	1,675	3,559	4,340	3,564
c_x	0,041	0,033	0,035	0,024	0,036	0,041	0,034
Median	17,956	34,139	43,983	69,651	100,223	105,550	104,713
Q1 (25%)	17,567	33,539	43,178	68,618	98,086	101,919	102,668
Q3 (75%)	18,439	34,664	45,315	71,023	102,425	108,105	106,736

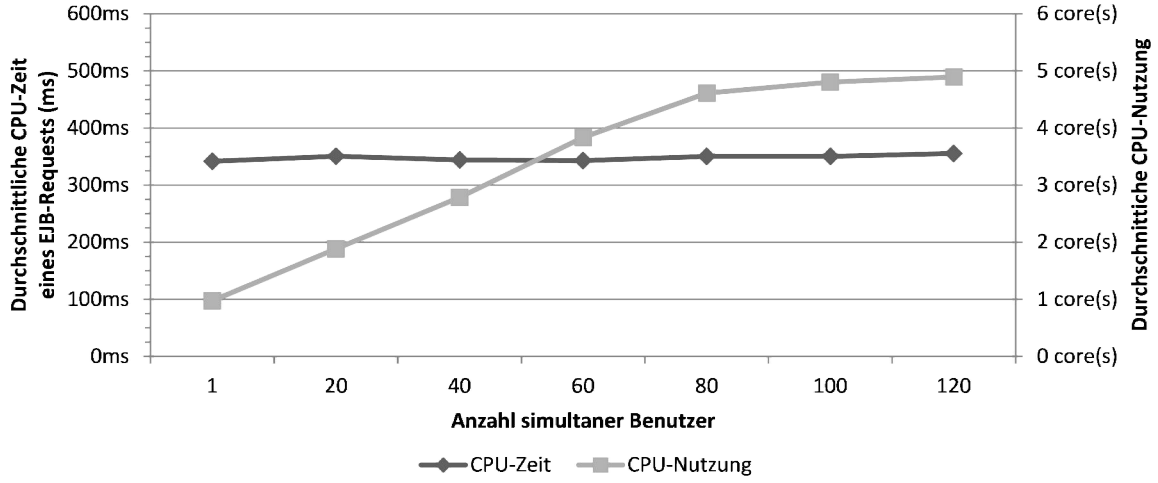
Abbildung 5-13: Durchschnittliche Enqueue-Zeit pro Sperrobjekt in Millisekunden (Szenario 1)
Quelle: eigene Darstellung

Die Zeit, in der auf das Setzen einer Sperre bzw. die Freigabe eines Objektes gewartet wird, erhöht sich erwartungsgemäß mit der steigenden Anzahl an simultanen Benutzern. In Abbildung 5-13 kann die durchschnittliche Zeit, die vom Enqueue-Trace für die einzelnen Sperranfragen aufgezeichnet wird (vgl. Kapitel 3.1.3), betrachtet werden. Die dargestellten Enqueue-Zeiten EZ beinhalten die Zeit der Datenbankabfrage, da sich die Werte aus der Zeitdifferenz zwischen dem Erstellen (t_{create}) und dem Lösen der Sperre (t_{delete}) berechnen:

$$EZ = t_{create} - t_{delete}$$

Auffallend ist hierbei der stetige Zuwachs bis zu einer Anzahl von 80 gleichzeitig agierenden Benutzern. Die annähernd gleichbleibende Enqueue-Zeit ab 80 Benutzern ergibt sich aus der maximalen Parallelität, über die das Portalsystem aufgrund der 80 konfigurierten Applikations-Threads verfügt.

5.5.4.5 CPU-Zeiten



CPU-Zeit	μ	341,573	350,497	344,091	342,906	350,312	350,289	355,537
	σ	24,962	26,963	25,574	24,111	26,851	32,837	30,426
	c_x	0,073	0,077	0,074	0,070	0,077	0,094	0,086
	Median	341,312	350,543	343,758	343,312	350,097	339,359	351,561
	Q1 (25%)	318,938	325,933	320,937	320,938	325,934	320,256	335,780
	Q3 (75%)	357,321	368,153	360,088	358,478	367,386	380,678	374,888
CPU-Nutzung	μ	0,974	1,885	2,784	3,834	4,612	4,804	4,894
	σ	0,018	0,028	0,052	0,064	0,099	0,109	0,109
	c_x	0,019	0,015	0,019	0,017	0,022	0,023	0,022
	Median	0,975	1,886	2,788	3,838	4,618	4,811	4,901
	Q1 (25%)	0,966	1,872	2,760	3,803	4,566	4,752	4,842
	Q3 (75%)	0,987	1,905	2,822	3,880	4,684	4,882	4,972

Abbildung 5-14: CPU-Zeit eines EJB-Requests sowie durchschnittliche CPU-Nutzung (Szenario 1)
Quelle: eigene Darstellung

Wie in Kapitel 4.1.1 dargestellt, leiten sich aus der Theorie der Warteschlangennetze zwei Annahmen bezüglich der CPU-Ressource ab. Zum einen ist die CPU-Zeit eines Lastschrittes unabhängig von der Anzahl simultan agierender Benutzer, zum anderen steigt die CPU-Nutzung linear zur Benutzeranzahl.

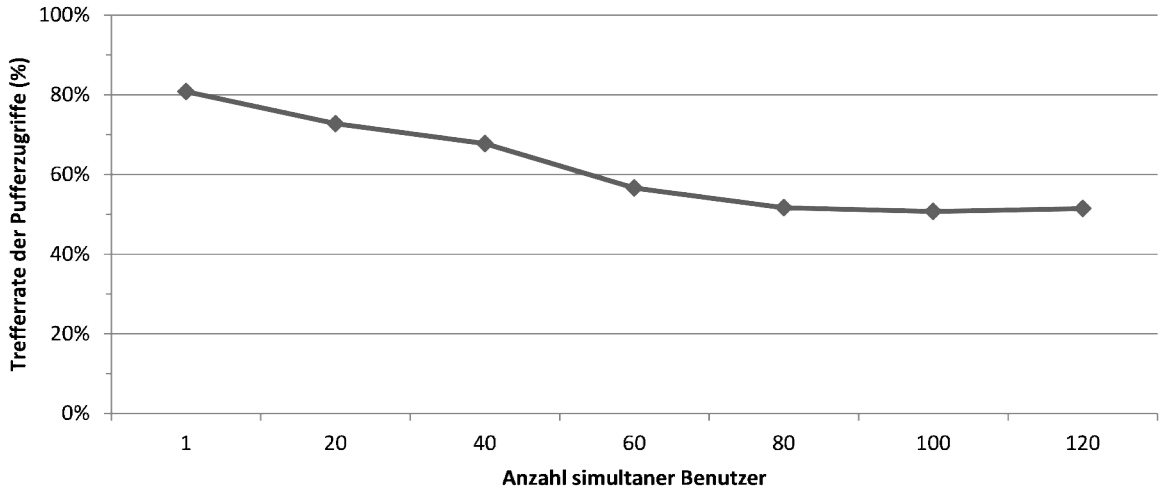
Zur Überprüfung der ersten Annahme (CPU-Zeit) soll als Beispiel der EJB-Request aus Kapitel 3.2.3 (Abbildung 3-20) herangezogen werden. Wie man in Abbildung 5-14 erkennen kann, bleiben die durchschnittlichen CPU-Zeiten bei einer steigenden Anzahl von Benutzern nahezu konstant, wenngleich bei 100 und 120 simultanen Benutzern eine leicht erhöhte Streuung aus den Messdaten abzulesen ist.

Die zweite Annahme (CPU-Nutzung) wird mittels verwendeter Kerne (*Cores*) überprüft. Die virtuellen Prozessoreinheiten werden der logischen Partition (LPAR) über die Managementkonsole des IBM-Power-Servers zugewiesen und ihre Nutzung über Betriebssystemmittel (vgl. Kapitel 3.3) aufgezeichnet. Auch hier ist bei einer Benutzerzahl größer 80 ein Einbruch des nahezu linearen Anstiegs zu erkennen, der ebenso auf die maximale Parallelität der Applikations-Threads zurückzuführen ist. Nichtsdestotrotz zeigt der lineare Anstieg deutlich, dass die CPU-Ressourcen abhängig von der Anzahl an Benutzern genutzt werden und der Ressourcenbedarf somit der zweiten Annahme entspricht.

5.5.5 Szenario 2 – Knappe Systemressourcen

Im zweiten Szenario wird, wie bereits geschildert, der Tabellenpuffer bewusst verkleinert, sodass nicht alle Objekte darin Platz finden und es folglich zu Verdrängungen kommt. Zudem werden die CPU-Ressourcen auf drei virtuelle Kerne gekappt, damit eine Volllast der CPU-Ressourcen geprüft werden kann.

5.5.5.1 Pufferzugriffe



μ	80,81	72,76	67,76	56,59	51,65	50,71	51,43
σ	3,15	3,25	3,25	3,59	3,47	3,36	3,66
c_x	0,039	0,045	0,048	0,063	0,067	0,066	0,071
Median	81,55	73,53	68,53	57,46	52,48	51,51	52,48
Q1 (25%)	79,49	71,41	66,41	55,17	50,26	49,34	50,72
Q3 (75%)	82,80	74,80	69,80	58,79	53,79	52,79	53,59

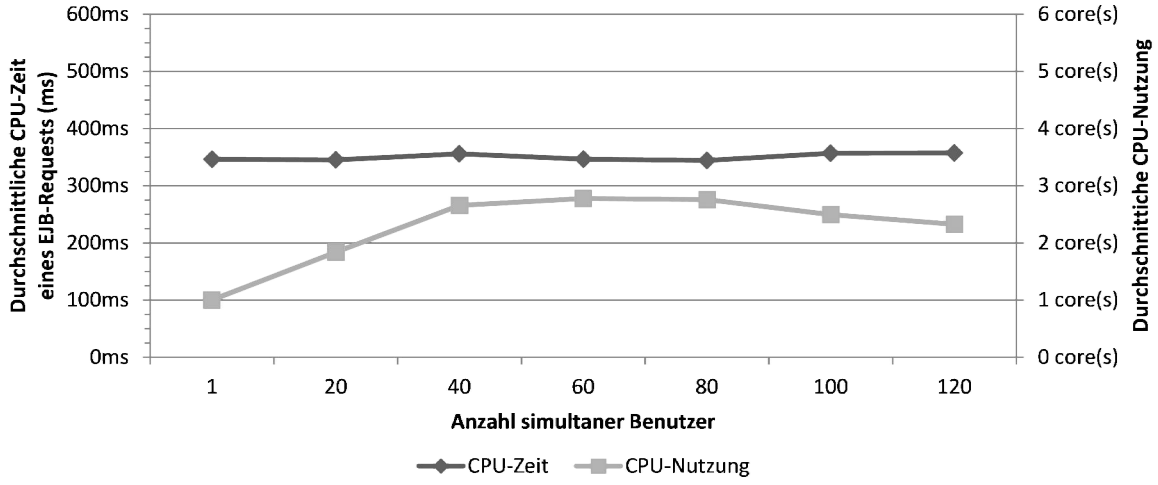
Abbildung 5-15: Trefferrate der Pufferzugriffe in Prozent (Szenario 2)
Quelle: eigene Darstellung

Aufgrund der geringen Tabellenpuffergröße ist eine verminderte Trefferquote zu erwarten, die mit einer steigenden Benutzeranzahl weiter verringert wird. Dies begründet sich durch eine zunehmende Anzahl an Verdrängungen und Invalidierungen.

Abbildung 5-15 zeigt die erhobenen Daten der Tabellenpufferaufzeichnung und bestätigt das erwartete Verhalten. Die in der Literatur für ein gut konfiguriertes System geforderte Trefferrate von 95% (Heiss/Veirich/Gratzl 2005, 355) wird nicht erreicht. Zudem ist im Vergleich zum ersten Szenario eine deutlich höhere Streuung zu verzeichnen, die hauptsächlich auf die unterschiedliche Anzahl an Verdrängungen zurückzuführen ist.

Die Weiterleitungswahrscheinlichkeit ist in diesem Szenario außerdem von der Anzahl simultaner Benutzer abhängig und muss entsprechend parametrisiert werden. Diese Abhängigkeit ist jedoch nicht in der in Kapitel 4.2.4 vorgestellten Berechnungsmethode abbildbar. Somit kann die Trefferrate nur über Erfahrungswerte geschätzt werden, falls keine Messdaten zur Verfügung stehen sollten. Die Schätzung stellt einen wesentlichen Ungenauigkeitsfaktor dar, sodass bei datenbankintensiven Workloads eine unzureichende Größe des Tabellenpuffers auch einen erheblichen Einfluss auf die Simulationsgenauigkeit nimmt.

5.5.5.2 CPU-Zeiten



CPU-Zeit	μ	346,350	345,398	355,888	346,398	344,402	356,896	357,386
	σ	25,031	26,039	28,592	26,039	25,017	32,734	30,857
	c_x	0,072	0,075	0,080	0,075	0,073	0,092	0,086
	Median	341,633	340,163	350,139	341,163	339,372	347,544	349,438
	Q1 (25%)	330,605	324,479	332,918	325,479	324,304	328,567	332,887
	Q3 (75%)	356,382	359,847	371,754	360,847	358,285	377,804	378,765
CPU-Nutzung	μ	1,001	1,840	2,657	2,776	2,759	2,494	2,326
	σ	0,017	0,018	0,038	0,030	0,034	0,038	0,023
	c_x	0,017	0,010	0,014	0,011	0,012	0,015	0,010
	Median	1,001	1,839	2,657	2,781	2,763	2,499	2,331
	Q1 (25%)	0,986	1,824	2,624	2,767	2,729	2,461	2,306
	Q3 (75%)	1,012	1,852	2,683	2,794	2,781	2,519	2,343

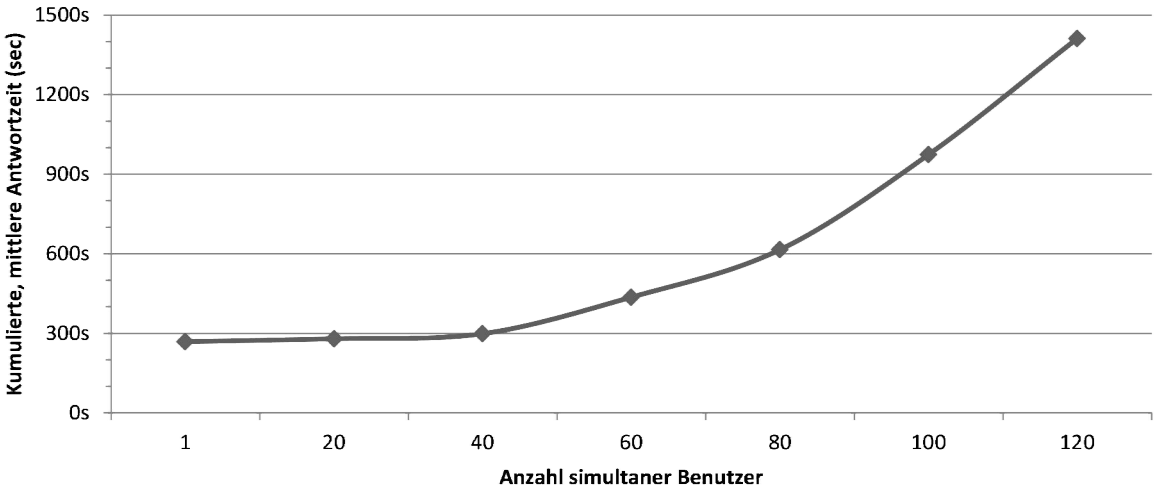
Abbildung 5-16: CPU-Zeit eines EJB-Requests sowie durchschnittliche CPU-Nutzung (Szenario 2)
Quelle: eigene Darstellung

In Abbildung 5-16 sind erneut die CPU-Zeiten des beispielhaften EJB-Requests sowie die aufgezeichnete CPU-Nutzung auf Betriebssystemebene ersichtlich.

Die Grenze von drei verfügbaren Kernen wird zwischen 40 und 60 simultanen Benutzern erreicht. Ab diesem Zeitpunkt sind die CPU-Ressourcen des Systems erschöpft. Ab 80 Benutzern ist dann ein Einbruch der CPU-Nutzung (System-CPU-Zeit plus Benutzer-CPU-Zeit) zu erkennen. Wie die Aufzeichnungen des Betriebssystem-Hilfsmittels „topas_nmon“ zeigen, bricht die CPU-Nutzung nach bestimmten Intervallen immer wieder ein und gilt als Indiz für eine Garbage-Collector-Aktivität in diesen Zeiträumen. Dieser Sachverhalt wird in Kapitel 5.8 näher untersucht.

Die Bearbeitungszeiten des EJB-Requests zeigen auch in diesem Szenario keine wesentlichen Veränderungen und bestätigen somit die Unabhängigkeit von Benutzeranzahl und CPU-Zeit sowie der Anzahl an zugewiesenen, virtuellen Kernen.

5.5.5.3 Antwortzeiten



μ	268,708	279,677	298,704	436,410	615,527	974,425	1412,108
σ	17,344	17,306	19,023	29,224	45,460	114,731	181,838
c_x	0,065	0,062	0,064	0,067	0,074	0,118	0,129
Median	262,590	273,654	292,399	426,057	599,423	933,781	1347,691
Q1 (25%)	260,603	271,829	290,012	423,087	594,802	922,119	1329,207
Q3 (75%)	270,573	280,985	300,730	439,529	620,379	986,671	1431,516

Abbildung 5-17: Kumulierte Antwortzeiten in Sekunden (Szenario 2)
Quelle: eigene Darstellung

Die Antwortzeiten verzeichnen in diesem Szenario (siehe Abbildung 5-17) einen deutlich höheren Anstieg als bei den Vergleichswerten in Szenario 1. Die Ursachen für den erhöhten Anstieg lassen sich nicht auf eine einzelne Einflussgröße beschränken, vielmehr können in diesem Bereich die Auswirkungen allgemeiner Ressourcen-Engpässe erkannt werden:

- Neben den bereits im ersten Szenario festgestellten Ursachen für den Anstieg der Antwortzeiten treten in diesem Szenario zunehmende Verdrängungen sowie Invalidierungen im Tabellenpuffer auf, die in einer erhöhten Anzahl von Datenbankabfragen resultieren.

- Ab ca. 40 bis 50 Benutzern sind die CPU-Ressourcen voll ausgelastet und müssen folglich von einer zunehmenden Anzahl an Benutzern geteilt werden. Dies führt, ähnlich zu der Begrenzung der Applikations-Threads im ersten Szenario, zu einem wesentlichen Anstieg der Antwortzeiten.

- Deutlich sichtbar ist ein weiterer Zuwachs ab 80 Benutzern, der mit einem zunehmenden System-Overhead zusammenhängt (vgl. Jehle 2010, 185f.). Aber auch der soeben erwähnte Anstieg der GC-Aktivitäten wirkt sich auf die Antwortzeiten aus und wird in Kapitel 5.8 weiter analysiert.

Zusammenfassend wird ein Hochlastbereich erreicht, in dem sowohl das SAP-Portal-System, als auch die Systemressourcen ausgelastet sind. Der Mangel an zusätzlichen CPU-Ressourcen bewirkt einen deutlichen Zuwachs der Antwortzeiten und verschärft sich durch erhöhte Systemaktivitäten außerhalb des SAP-Portal-Systems.

5.6 Simulation

Nachdem das Modell über die erhobenen Messdaten parametrisiert werden konnte, wird in diesem Kapitel die Durchführung der Simulation beschrieben. Dabei werden zuerst die verschiedenen Modellkomponenten in der Eingabedatei erläutert und im Anschluss die Informationen, die vom Simulator in die Ausgabedatei geschrieben werden, dargestellt.

Der verwendete Simulator LQsim (LQsim 2011), der bereits in Kapitel 2.5.4 vorgestellt wurde, basiert auf den ereignisgesteuerten PARASOL-Simulator (Neilson 1991). Für die Syntax der Eingabedatei stehen zwei verschiedene Formate zur Verfügung:

- Traditionelle Grammatik
- XML-Schema

Obwohl beide Formate gleich mächtig sind, ist ein Trend zum XML-Schema erkennbar. Dies ergibt sich aus der Benutzeranleitung (Franks et al. 2012), in der die traditionelle Grammatik seit der Aktualisierung vom Februar 2011[7] nur noch im letzten Kapitel beschrieben wird. Dennoch wird im folgenden Abschnitt lediglich auf die traditionelle Grammatik eingegangen, da die durchgeführten Untersuchungen dieser Arbeit zum Zeitpunkt dieser jüngsten Entwicklung zu weit fortgeschritten waren, als dass der Aufwand eines Wechsels zum XML-Schema gerechtfertigt gewesen wäre. Es sei jedoch angemerkt, dass für zukünftige Arbeiten mit LQsim das XML-Schema bevorzugt werden sollte.

Der Simulator wird auf Kommandozeilenebene ausgeführt und stellt verschiedene Optionen zur Verfügung. Die bei den Versuchen und Testläufen dieser Arbeit verwendeten Parameter seien an dieser Stelle kurz dargestellt:

- *-A run-time[,precision[,skip]]*: Die Dauer eines Simulationsdurchlaufs (Blöcke) wird über den Wert „run-time“ angegeben. Die Anzahl der Simulationsblocks wird automatisch ermittelt. Die Präzision, mit anderen Worten die Definition des Konfidenzintervalls, kann über den zweiten Wert angegeben werden. „Skip“ stellt eine Einschwingphase dar; das mit diesem Wert definierte Zeitintervall fließt nicht in das Ergebnis mit ein.

- *-B blocks[,runtime[,skip]]*: Alternativ kann die Anzahl der Blöcke manuell festgelegt werden. Die beiden anderen Werte sind analog zu Option „-A“.[8]

- *-R*: Mit diesem Parameter werden zusätzliche statistische Information in der Standardausgabe ausgegeben. Unter Verwendung von Option „-A“ oder „-B“ werden zusätzlich die Percentile (95% sowie 99%) ausgegeben.

[7] Revision 9242, 11. Februar 2011

[8] Der verwendete Simulator (Version 5.6) hatte unter Microsoft® Windows® Probleme bei der Ausführung der Simulationsläufe mit Option *-A* und *-B*. Daher wurde ein Linuxsystem für die Simulationsläufe eingesetzt.

- -*S seed*: Bestimmt den Saatwert, der für die Initialisierung des Zufallszahlengenerators verwendet wird.

5.6.1 Eingabedatei des Simulators

Die Eingabedatei des Simulators stellt eine editierbare Textdatei dar, die grundsätzlich in fünf Bereiche unterteilt wird:

1. *Allgemeine Informationen*: Neben einem String zur Beschreibung der Eingabedatei werden in diesem Bereich Simulator-Parameter übergeben, beispielsweise das Konvergenzkriterium, die Iterationsgrenze oder das Ausgabeintervall.

2. *Prozessordeklarationen*: Dieser Bereich deklariert die Prozessoren, deren Scheduling-Typ sowie die Anzahl ihrer Kerne.

3. *Task-Deklarationen*: In diesem Abschnitt werden die Tasks, deren zugehörigen Entries und task-spezifische Parameter deklariert.

4. *Entry-Deklarationen*: Der Entry-Bereich bestimmt die Parameter der Entries. Dies können zum Beispiel Service-Zeiten oder Aufrufe anderer Entries sein.

5. *Deklarationen der Aktivitäten:* Im letzten Abschnitt werden die Aktivitäten, sofern sie im Modell definiert sind, deklariert.

Das folgende exemplarische Modell (vgl. auch die schematische Darstellung des vollständigen Modells in Abbildung 4-16) beschreibt die in Kapitel 4.2 dargestellten Komponenten[9]. Aus Gründen der Übersichtlichkeit wird die jeweilige Anzahl der Komponenten reduziert:

- 1 Benutzertyp (mit 4 Benutzern).

- 2 Interaktionsschritte.

- 1 Java-Server-Instanz mit 2 korrespondierenden Lastschritten: Lastschritt 1 stellt eine Lese- und eine Schreibsperren-Anfrage, Lastschritt 2 eine lesende DB-Anfrage ohne Sperrobjekt.

- 2 SQL-Abfragen, eine lesend („Select“) und eine schreibend („Update“).

- 1 Sperrbereich für beide SQL-Abfragen.

- 1 Tabellenpuffereintrag der lesenden SQL-Abfrage.

Allgemeine Informationen

Für die Definition der Simulator-Parameter werden die Standardwerte aus der Dokumentation übernommen:

[9] Mit Ausnahme des explizit modellierten Garbage-Collectors.

```
## Beispielhafte Modellierung + Parametrisierung von 2 vereinfachten Interaktionsschritten
## ausgeführt mit: LQsim v5.6
G
    "Modellbeispiel"          # optionaler String für Informationen
    0.0001                    # Konvergenzkriterium
    0                         # Iterationsgrenze
    0                         # Ausgabeintervall
    0.9                       # Under-Relaxiation-Koeffizient
-1
```

Listing 5-2: Allgemeine Inforationen zur Simulator-Eingabedatei (Beispiel)
Quelle: eigene Darstellung

Prozessordeklarationen

An die drei Schichten (vgl. Kapitel 2.1.1.1) werden entsprechende CPU-Ressourcen vergeben. Die Prozessoren der Benutzer- und Persistenzschicht haben eine unendliche Kapazität, da lediglich die Kapazitätsengpässe der Applikationslogik betrachtet werden sollen.

```
## Prozessordeklarationen
P 3
    p Processor_User f i              # Prozessorressource Benutzer, unendliche Kapazität i
    p Processor_Server s 1.0 m 40     # Prozessorressource Server, Prozessor-Sharing
    p Processor_DB f i                # Prozessorressource DB, FCFS, infinite
-1
```

Listing 5-3: Prozessordeklarationen (Beispiel)
Quelle: eigene Darstellung

Task-Deklarationen

Im Bereich der Task-Deklarationen werden die Tasks der einzelnen Komponenten des Modells spezifiziert:

- *Benutzertyp*: Die Multiplizität $m = 20$ des Tasks gibt an, dass 20 Benutzer dieses Benutzertyps modelliert werden, *r* kennzeichnet den Task als Referenztask.
- *Lastschritte*: Der Lastschritte-Task hat eine Multiplizität von 40, da 40 Applikations-Threads für die Java-Server-Instanz konfiguriert werden.
- *Datenbank*: Die 10 Datenbankverbindungen der Java-Server-Instanz werden über die entsprechende Multiplizität des Datenbank-Tasks modelliert.
- *Tabellenpuffer*: Der Tabellenpuffer-Task hat eine unendliche Multiplizität.
- *Sperrmanagement*: Das Sperrmanagement besteht aus einem Task für den Enqueue-Server sowie drei Tasks für jeden Sperrbereich (Lesesperre, Schreibsperre, Semaphor). Zusätzlich wird ein „Pseudo-Task“ definiert, der für die Aktivitätsdefinition benötigt wird.

```
16 ## Task-Deklarationen   U=Usertype, W=Workloadstep, D=DB, P=Puffer, E=Enqueue – xxE=Entry
17 ##                      RS=Lesesperre, WS=Schreibsperre, S=Semaphor, WSP=WS-Pseudotask
18 T 8
19    t U1 r U1E1 -1 Processor_User m 20
20    t W1 f W1E1 W1E2 -1 Processor_Server m 40      # 40 Threads, 1 Java-Instanz
21    t D1 f D1E1 D1E2 -1 Processor_DB m 10          # 2 SQL-Queries, 10 DB-Verbindungen
22    t P1 f P1E1 -1 Processor_Server i              # Tabellenpuffer, unendliche Multiplizität
23    t E1 f E1E1 E1E2 -1 Processor_Server           # Enqueue-Server, 2 SQL-Queries
24    t RS1 f RS1E1 -1 Processor_Server              # Lesesperre, Sperrbereich 1
25    t WS1 f WS1CHK WS1E1 -1 Processor_Server       # Schreibsperre, Sperrbereich 1, inkl. Checkentry
26    t S1 S S1W S1S -1 Processor_Server            # Binärsemaphor, Wait- und Signal-Entries
27 -1
```

Listing 5-4: Task-Deklarationen (Beispiel)
Quelle: eigene Darstellung

Entry-Deklarationen

Im Abschnitt der Entry-Deklaration werden, wie bereits erwähnt, die Entries der einzelnen Tasks spezifiziert. Im vorgestellten Beispiel wurden die folgenden Deklarationen durchgeführt:

- Die Entry des Benutzertyp-Tasks verweist auf die Aktivität, die die einzelnen Interaktionsschritte durchläuft.
- Den Entries der Lastschritte-Tasks werden die Service-Zeiten und Varianzen zugewiesen.
- Die Anfragen, die von den Lastschritte-Tasks gesendet werden, werden spezifiziert. Im gewählten Beispiel werden im ersten Interaktionsschritt Lese- und Schreibsperren-Anfragen gesendet, im zweiten Lastschritt Anfragen an den Tabellenpuffer.
- Den Datenbank-Entries der SQL-Abfragen werden entsprechende Service-Zeiten und Varianzen zugewiesen.
- Bei der Tabellenpuffer-Entry der lesenden SQL-Abfrage wird eine Weiterleitungswahrscheinlichkeit zur entsprechenden Entry des Datenbank-Tasks definiert.
- Der Enqueue-Server leitet beide SQL-Abfragen an die entsprechenden Tasks des Sperrmanagements (Lese-/Schreibsperre) weiter.
- Sowohl der Lesesperren- als auch der Schreibsperren-Task wird über einen Aktivitätsgraphen definiert. Die entsprechenden Entries verweisen auf die jeweilige Start-Aktivität des Aktivitätsgraphen.
- Die Check-Entry des Schreibsperren-Tasks wird spezifiziert.
- Die Entries des Semaphors werden als Wait- bzw. Signal-Entry deklariert.

– Die Entry des Schreibsperren-Pseudotasks wird definiert, indem auf die entsprechende Aktivität verwiesen wird.

```
28 ## Entry-Deklarationen    A=Aktivität
29 E 23
30     A U1E1 U1A1                  # Interaktionsschritte sind über Aktivität U1A1 definiert
31     s W1E1 2.731 -1              # Lastschritt 1: Service-Zeit
32     c W1E1 0.00438 -1            # Lastschritt 1: quadrierter Variationskoeffizient
33     H W1E1 1 2.5 : 4.0           # Lastschritt 1: Abgrenzungen Histogramm
34     f W1E1 0 -1                  # Lastschritt 1: stochastische Aufrufe
35     y W1E1 E1E1 7.0 -1           # 1 Anfrage Lesesperre SQL-Query 1
36     y W1E1 E1E2 3.0 -1           # 1 Anfrage Schreibsperre SQL-Query 1
37     s W1E2 3.489 -1              # Lastschritt 2: Service-Zeit
38     c W1E2 0.00414 -1            # Lastschritt 2: quadrierter Variationskoeffizient
39     f W1E2 0 -1                  # Lastschritt 2: stochastische Aufrufe
40     y W1E2 P1E1 15.0 -1          # 7 Anfragen über Tabellenpuffer
41     s D1E1 0.017 -1              # SQL-Query 1: Service-Zeit  (Select-Statement)
42     c D1E1 0.00168 -1            # SQL-Query 1: quadrierter Variationskoeffizient
43     s D1E2 0.029 -1              # SQL-Query 2: Service-Zeit  (Update-Statement)
44     c D1E1 0.00168 -1            # SQL-Query 2: quadrierter Variationskoeffizient
45     s P1E1 0.0 -1                # Tabellenpuffer: Service-Zeit 0 (da unwesentlich)
46     F P1E1 D1E2 0.02 -1          # Tabellenpuffer: Weiterleitungswahrscheinlichkeit 2%
47     s E1E1 0.001 -1              # Enqueue-Server Select-Statement: Service-Zeit
48     c E1E1 0.0 -1                # quadrierter Variationskoeffizient 0, somit deterministisch
49     F E1E1 RS1E1 1.0 -1          # wird an die Lesesperre weitergeleitet
50     s E1E2 0.001 -1              # Enqueue-Server Update-Statement: Service-Zeit
51     c E1E2 0.0 -1                # quadrierter Variationskoeffizient 0, somit deterministisch
52     F E1E2 WS1E1 1.0 -1          # wird an die Schreibsperre weitergeleitet
53     A RS1E1 RSA1                 # Lesesperre über Aktivität definiert
54     A WS1E1 WSA1                 # Schreibsperre über Aktivität definiert
55     s WS1CHK 0.0 -1              # Schreibsperre: Check-Entry
56     s S1S 0.0 -1                 # Semaphor-Sperrbereich: Signal-Entry Service-Zeit
57     s S1W 0.0 -1                 # Semaphor-Sperrbereich: Wait-Entry Service-Zeit
58     P S1S                        # Semaphor: definiert S1S als Signal-Entry
59     V S1W                        # Semaphor: definiert S1W als Wait-Entry
60 -1
```

Listing 5-5: Entry-Deklarationen (Beispiel)
Quelle: eigene Darstellung

Deklaration der Aktivitäten

Sieht man von der expliziten GC-Modellierung ab, sieht das in Kapitel 4.2 vorgestellte Modell drei Entries mit Aktivitäten vor, namentlich die Aktivitätsgraphen der Benutzer-Interaktionen, der Lesesperre sowie der Schreibsperre.

Der Aktivitätsgraph des Benutzertyp-Tasks stellt eine Sequenz der einzelnen Interaktionsschritte dar. In Listing 5-6 sind die Aktivitätsdefinitionen der beiden beispielhaften Interakti-

onsschritte dargestellt. Jeder Interaktionsschritt wird genau ein Mal durchgeführt, dementsprechend ist die Anzahl der Aufrufe deterministisch.

Die Service-Zeit des Interaktionsschritts wird auf 0 gesetzt, da der Ressourcenverbrauch der Präsentationsebene nicht betrachtet wird.

Wie später noch gezeigt wird, werden die durchschnittlichen Antwortzeiten zu den einzelnen Aktivitäten in der Ausgabedatei des Simulators ausgegeben.

```
61 ## Aktivitäts-Deklarationen
62 A U1
63     s U1A1 0.0                  # 1. Interaktionsschritt, Service-Zeit  = 0
64     Z U1A1 0.0                  # optionale Denkzeit des Interaktionsschrittes
65     f U1A1 1                    # deterministische Anzahl an Aufrufen
66     y U1A1 W1E1 1.0             # Aufruf des 1. Lastschritts
67     s U1A2 0.0                  # 2. Interaktionsschritt
68     Z U1A2 0.0                  # optionale Denkzeit
69     f U1A2 1                    # deterministisch
70     y U1A2 W1E2 1.0             # Aufruf Lastschritt 2
71     :
72     U1A1 -> U1A2                # Aktivitätsgraph: Sequenz von A1 nach A2
73 -1
```

Listing 5-6: Aktivitätsdeklaration der Benutzer-Interaktionen (Beispiel)
Quelle: eigene Darstellung

In Listing 5-7 ist die Aktivitätsdeklaration der Lesesperre aufgelistet. Da es sich lediglich um einen logischen Mechanismus zur Realisierung der Lesesperre handelt, werden alle Aktivitäten mit einer Service-Zeit von 0 spezifiziert. Der Rechenaufwand für das Setzen der Sperre ist bereits in der Entry des Enqueue-Servers definiert.

Im Gegensatz zur schematischen Darstellung in Kapitel 4.2.3 muss eine zusätzliche Und-Gabelung eingeführt werden, die zum einen die Antwort an die aufrufende Entry sendet, zum anderen die nebenläufigen Threads wieder zusammenführt und die Anfrage zur Freigabe des Semaphors sendet. Dazu wird eine Pseudo-Aktivität verwendet, da der direkte Schritt von der Und-Gabelung zur Und-Vereinigung von LQsim nicht akzeptiert wird. Da keine Service-Zeiten für die Aktivitäten definiert sind, nimmt die zusätzliche Aktivität keinen Einfluss auf die Antwortzeiten.

Zudem musste aufgrund eines Initialisierungsproblems des Simulators die Aktivität zur Überprüfung, ob eine Schreibsperre besteht, vorgezogen werden und erst im Nachhinein die Und-Gabelung zur parallelen Ausführung modelliert werden. Dies nimmt jedoch keinen Einfluss auf die Sperrlogik bzw. die Antwortzeiten, da vereinfacht gesagt die Überprüfung, ob eine Schreibsperre vorliegt, in jedem Fall die Lesesperre-Anfrage blockiert. Somit ist auch dieser Ansatz valide.

```
74  ## Aktivitäts-Deklarationen
75  A RS1
76      s RSA1 0.0                  # Startaktivität Lesesperre – keine Service-Zeiten
77      s RScheck 0.0               # Aktivität: Prüfung ob Schreibsperre vorliegt
78      s RSlock 0.0                # Aktivität: Sperre am Semaphor setzen
79      s RSdb 0.0                  # Aktivität: Datenbankanfrage
80      s RSreply 0.0               # Aktivität: Senden der Antwort
81      s RSpseudo 0.0              # Pseudo-Aktivität für zusätzliche Und-Gabelung
82      s RSrelease 0.0             # Aktitivtät: Freigabe des Semaphors
83      y RScheck WS1CHK 1.0        # Rendezvous mit Check-Entry
84      y RSlock S1W 1.0            # Anfrage an Wait-Entry des Semaphors
85      y RSdb P1E1 1.0             # Anfrage an DB über Tabellenpuffer
86      y RSrelease S1S 1.0         # Anfrage an Signal-Entry des Semaphors
87      f RScheck 1                 # deterministische Anzahl an Anfragen
88      f RSlock 1                  # deterministische Anzahl an Anfragen
89      f RSdb 0                    # stochastische Anzahl an Anfragen
90      f RSpseudo 1                # deterministische Anzahl an Anfragen
91      f RSrelease 1               # deterministische Anzahl an Anfragen
92      :                           # Definition des Aktivitätsgraphen:
93      RSA1 -> RScheck;            # Sequenz
94      RScheck -> RSlock & RSdb;   # Und-Gabelung
95      RSdb -> RSpseudo & RSreply; # zusätzliche Und-Gabelung (Pseudo-Aktivität)
96      RSpseudo & RSlock -> RSrelease; # Und-Vereinigung
97      RSreply[RS1E1]              # Antwort
98  -1
```

Listing 5-7: Aktivitätsdeklaration der Lesesperre (Beispiel)
Quelle: eigene Darstellung

Die Aktivitätsdeklaration der Schreibsperre ist in Listing 5-8 dargestellt. Auch hier muss im Gegensatz zur schematischen Darstellung in Kapitel 4.2.3 eine Und-Gabelung eingeführt werden, die die Antwort an die aufrufende Entry sowie die Anfrage an die Signal-Entry des Semaphors parallel durchführt. Die initialen Definitionen verhalten sich analog zu den vorangegangenen Aktivitätsgraphen.

```
99  ## Aktivitäts-Deklarationen
100 A WS1
101     s WSA1 0.0                  # Startaktivität Schreibsperre – keine Service-Zeiten
102     s WSlock 0.0                # Aktivität WSlock sendet Anfrage an Wait-Entry
103     s WSdb 0.0                  # Aktivität WSdb sendet Anfrage an Datenbank-Entry
104     s WSreply 0.0               # Aktivität WSreply sendet die Antwort
105     s WSrelease 0.0             # Aktivität WSrelease sendet Anfrage an Signal-Entry
106     f WSlock 1                  # deterministsche Anzahl an Anfragen
107     f WSdb 0                    # stochastische Anzahl an Anfragen
108     f WSreply 1                 # deterministsche Anzahl an Anfragen
109     y WSlock S1W 1.0            # Anfrage an Wait-Entry des Semaphors
110     y WSdb D1E2 1.0             # Anfrage an Datenbank
111     y WSrelease S1S 1.0         # Anfrage an Signal-Entry des Semaphors
```

```
112     :
113     WSA1 -> WSlock;                 # Definition des Aktivitätsgraphen
114     WSlock -> WSdb;
115     WSdb -> WSrelease & WSreply;
116     WSreply[WS1E1]                  # Antwort
117  -1
```

Listing 5-8: Aktivitätsdeklaration der Schreibsperre (Beispiel)
Quelle: eigene Darstellung

5.6.2 Ausgabedatei des Simulators

Nachdem der Simulator mit der Eingabedatei gestartet und die Simulation durchlaufen wurde, werden die Ergebnisse in einer Textdatei präsentiert. Im ersten Abschnitt der Ausgabedatei werden allgemeine Informationen ausgegeben (siehe Listing 5-9):

- Version des Simulators sowie Urheberrechte
- Verwendeter Befehl zur Ausführung des Simulators
- Name der Eingabedatei
- Anzahl an durchgeführten Iterationen („Blöcken“) sowie Wert des Konvergenztests

```
1  Generated by: lqsim, version 5.6
2  Copyright the Real-Time and Distributed Systems Group,
3  Department of Systems and Computer Engineering
4  Carleton University, Ottawa, Ontario, Canada. K1S 5B6
5
6  Invoked as: lqsim -blocks=30 diss_sample.lqn
7  Input:  diss_sample.lqn
8
9  Convergence test value: 0
10 Number of iterations:  30
```

Listing 5-9: Allgemeine Informationen der Ausgabedatei (Beispiel)
Quelle: eigene Darstellung

Danach folgen verschiedene Abschnitte, die die Parametrisierung des Eingabemodells beschreiben:

- Prozessorinformationen sowie Scheduling-Algorithmen
- Task-Informationen (Name, Typ, Repliken, Prozessor, Priorität, Entries)
- Parametrisierte Service-Zeiten der Entries bzw. Aktivitäten
- Durchschnittliche Anzahl an Rendezvous zwischen Entries bzw. Aktivitäten
- Weiterleitungswahrscheinlichkeiten
- Phasentyp der Entries bzw. Aktivitäten (deterministisch oder stochastisch)

Daraufhin werden die Ergebnisse des Simulationslaufs dargestellt. Zuerst werden Angaben zu den durchschnittlichen Verzögerungen bei synchronen Anfragen und Vereinigungen wiedergegeben. Im Anschluss folgen, nach den einzelnen Phasen aufgeschlüsselt, die durchschnittlichen Service-Zeiten für die modellierten Entries bzw. Aktivitäten.

Unglücklicherweise wird der Begriff Service-Zeit ebenfalls bei der Parametrisierung des Modells (siehe Eingabedatei des Simulators, Kapitel 5.6.1) verwendet und stellt dort die reine Bearbeitungszeit, auch Bedienzeit genannt, dar. In der Ausgabedatei wird die durchschnittliche Service-Zeit als Antwortzeit verstanden, die sich aus folgenden Bestandteilen zusammensetzt:

- den Wartezeiten am Prozessor,
- den Wartezeiten bei synchronen Aufrufen weiterer Tasks,
- der eigenen Bedien- bzw. Prozessorzeit,
- den Bedienzeiten aufgerufener Tasks.

In dem in Kapitel 5.6.1 eingeführten Beispiel würde die Service-Zeit der Lastschritt-Entry des zweiten Interaktionsschrittes aus den Wartezeiten am Prozessor, den Aufrufen des Tabellenpuffers (die bei Eintreten der Weiterleitung aus Warte- und Bedienzeiten des Datenbank-Tasks bestehen können) und der eigenen Bearbeitungszeit zusammengesetzt werden. Die eigene Bearbeitungszeit (parametrisierte Service-Zeit) wird zwischen den Aufrufen aufgeteilt und entspricht dem Bedarf an Prozessorzeit. Zur Veranschaulichung ist der Sachverhalt in Abbildung 5-18 dargestellt, wobei zur Vereinfachung nur zwei Tabellenpufferaufrufe durchgeführt werden.

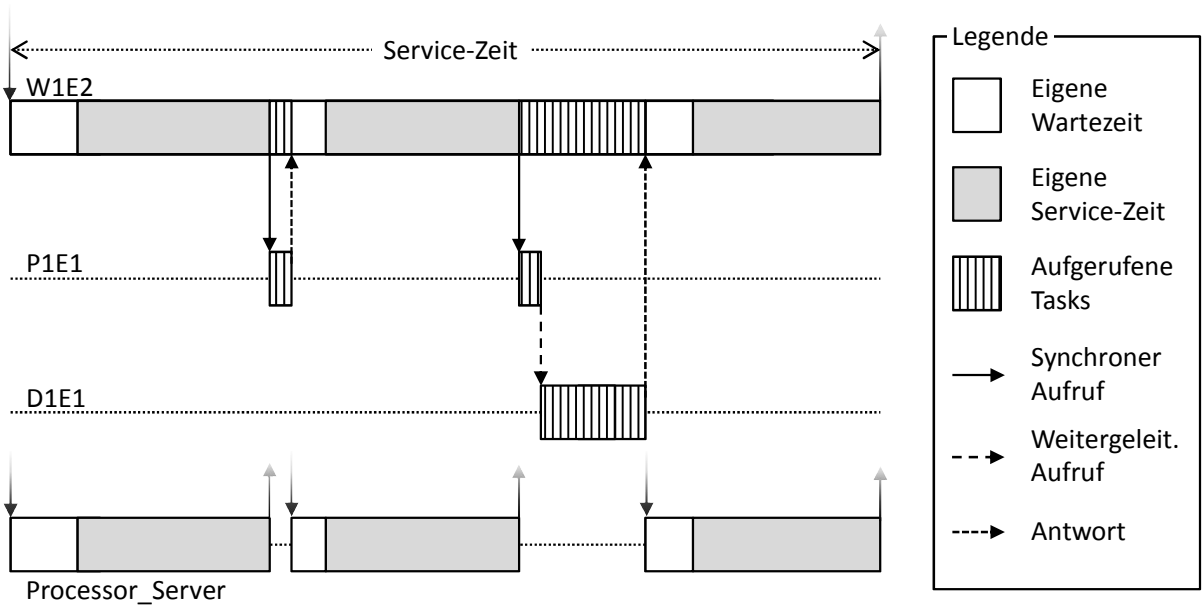


Abbildung 5-18: Zusammensetzung der Service-Zeit in der Ausgabedatei (Beispiel)
Quelle: eigene Darstellung

Die durchschnittlichen Wartezeiten für Task-Aufrufe sind in dem bereits genannten Abschnitt zu den durchschnittlichen Verzögerungen bei synchronen Aufrufen und Vereinigungen aufge-

führt. Die durchschnittlichen Wartezeiten am Prozessor werden in einem späteren Abschnitt über die Wartezeiten sowie Nutzung der Prozessoren pro Phase angegeben.

In Listing 5-10 sind die Service-Zeiten eines Simulationslaufes des Beispiels aus Kapitel 5.6.1 dargestellt. Neben den durchschnittlichen Service-Zeiten der beiden exemplarischen Interaktionsschritte (Aktivitäten U1A1 und U1A2) gibt die Service-Zeit der Entry U1E1, also der Entry des Benutzer-Tasks, die akkumulierten Service-Zeiten und somit die Gesamtantwortzeit des modellierten Workloads an, da die Service-Zeiten der Benutzeraktivitäten sowie die Denkzeiten 0 sind.

Im allgemeinen Fall errechnet sich die Antwortzeit der Interaktionsschritte aus:

$$\frac{\textit{Wiedergegebene Servicezeit der Aktvität des Interaktionsschrittes} - \textit{Denkzeit der Aktivität des Interaktionsschrittes}}{\textit{Antwortzeit des Interaktionsschrittes}}$$

Die Summe der Antwortzeiten ergibt folglich die Gesamtantwortzeit des modellierten Workloads.

```
  1  Service times:
  2
  3  Task Name      Entry Name     Phase 1     Phase 2
  4  U1             U1E1           6.39862     0
  5                 Activity Name
  6                 U1A1           2.97635
  7                 U1A2           3.42226
  8  W1             W1E1           2.97635
  9                 W1E2           3.42226
[…]
 15  RS1            RS1E1          0.0073017   0.0011371
 16                Activity Name
 17                RSA1           0
 18                RScheck        0.00672217
 19                RSdb           0.000579533
[…]
```

Listing 5-10: Service-Zeiten in der Ausgabedatei (Beispiel)
Quelle: eigene Darstellung

Im Lesesperrenbereich RS1 kann man die durchschnittlichen Sperrzeiten des Lesesperrenbereichs erkennen. Die Entry RS1E1 stellt die durchschnittliche Service-Zeit einer Sperranfrage (ohne Enqueue-Server-Zeit) dar. Den Mittelwert der reinen Datenbankzeit findet man bei der Aktivität *RSdb*.

Die Zeit am Enqueue-Server ist in dem Wert RS1E1 nicht enthalten und kann aufgrund des weitergeleiteten Aufrufs von E1E1 nach RS1E1 nicht direkt abgelesen werden. Allerdings ergibt sich die durchschnittliche Gesamtzeit (inkl. Datenbankzeit) der SQL-Abfrage E1E1 über die Addition der mittleren Service-Zeit von RS1E1, der mittleren Service-Zeit von E1E1 sowie der mittleren Verzögerung des Rendezvous zwischen dem Lastschritt und E1E1.

Nach der Auflistung der einzelnen Service-Zeiten, und somit der Antwortzeiten, werden Varianz und Variationskoeffizient zu den einzelnen Entries bzw. Aktivitäten festgehalten. Daraufhin folgt die Angabe zu den maximalen Service-Zeiten. Dabei wird die Wahrscheinlichkeit festgehalten, mit der eine definierte maximale Service-Zeit einer Entry oder Aktivität überschritten wurde. Im anschließenden Bereich werden die durchschnittliche Haltezeit bzw. Verweilzeit, die Varianz und die Nutzung des Semaphors zu den modellierten Semaphor-Tasks wiedergegeben. Die Werte des Sperrbereich-Semaphors haben aufgrund der Modellarchitektur keine besondere Aussagekraft. Allerdings lässt sich über die Nutzung des Semaphors ein Richtwert zur Häufigkeit von konkurrierenden Sperranfragen ablesen.

Falls in der Eingabedatei Histogramm-Abgrenzungen angegeben wurden, werden im darauffolgenden Abschnitt für die einzelnen Entries und Aktivitäten die Service-Zeit-Verteilungen angezeigt (siehe Listing 5-11). Für jede Service-Zeit-Verteilung wird zudem die Dichtefunktion graphisch dargestellt[10].

```
1  Service time distributions for entries and activities:
2
3  Histogram for entry W1E1, phase 1
4  <= bin          <          mean          +/- 95%
5  underflow                  0.000001      0.005876 |
6  2.5             2.575      0.000106      0.011018 |
7  2.575           2.65       0.003381      0.036669 |
8  2.65            2.725      0.028118      0.098527 |****
9  2.725           2.8        0.081842      0.126358 |***********
10 2.8             2.875      0.128808      0.147159 |*****************
11 2.875           2.95       0.147248      0.068986 |********************
12 2.95            3.025      0.142279      0.098897 |*******************
13 3.025           3.1        0.122559      0.107793 |*****************
14 3.1             3.175      0.098469      0.114087 |*************
15 3.175           3.25       0.075110      0.122356 |**********
16 3.25            3.325      0.055729      0.048624 |*******
17 3.325           3.4        0.039102      0.076470 |*****
18 3.4             3.475      0.026734      0.082474 |****
19 3.475           3.55       0.017975      0.078640 |**
20 3.55            3.625      0.011830      0.062569 |**
21 3.625           3.7        0.007924      0.033113 |*
22 3.7             3.775      0.004894      0.036135 |*
23 3.775           3.85       0.003016      0.035690 |
24 3.85            3.925      0.001948      0.013682 |
25 3.925           4          0.001192      0.041501 |
26 overflow                   0.001735      0.051717 |
```

Listing 5-11: Service-Zeit-Verteilung des ersten Lastschritts (Beispiel)
Quelle: eigene Darstellung

[10] Die Histogrammfunktion ist mit Version 5.6 des Simulators nicht mehr lauffähig. Daher musste für die Analyse der Service-Zeit-Verteilungen auf die Vorgängerversion 5.5 ausgewichen werden.

Mit einer Parametrisierung der Service-Zeit von 2,731 und der Verteilung von 0,00438 bei dem ersten Lastschritt des Beispielmodells ergibt sich die erwartete Gammaverteilung, die vom Simulator bei einem quadrierten Variationskoeffizienten von $0 \leq c_x^2 \leq 1$ angewendet wird (vgl. Kapitel 4.2.2).

Im vorletzten Abschnitt der Ausgabedatei werden Informationen zu dem Durchsatz und der Nutzung der einzelnen Entries und Aktivitäten wiedergegeben. Zusätzlich wird die Summe für die einzelnen Tasks errechnet.

Der Durchsatz spiegelt die Anzahl an Aufrufen pro Zeiteinheit wider. Der wiedergegebene Wert der Nutzung liegt im Bereich $0 \leq Nutzung_{Komponente} \leq m$, wobei m die Multiplizität der Komponente spezifiziert. Er stellt somit die durchschnittliche Anzahl an verwendeten Threads dar. Über die Nutzung der Komponenten können potentielle Flaschenhälse erkannt werden. Zeigt beispielsweise der Datenbanktask eine sehr hohe Nutzung der Threads bei einer durchschnittlichen Systemlast, kann eine Erhöhung der Datenbankverbindungen den Flaschenhals beseitigen. Ebenso können hohe Sperrzeiten einzelne Applikations-Threads blockieren. Sollten noch Systemressourcen vorhanden sein, jedoch keine freien Applikations-Threads mehr zur Verfügung stehen, liegen diese brach. Auch hier würde eine Erhöhung der Anzahl an Applikations-Threads dem Flaschenhals entgegenwirken.

Der letzte Abschnitt der Ausgabedatei widmet sich der Nutzung und den Wartezeiten der modellierten Prozessoren. Auch hier spiegelt die Nutzung die durchschnittlich verwendeten Kerne des Prozessors wider und entspricht somit einem Wert von $0 \leq Nutzung_{CPU} \leq m$, wobei m die Multiplizität des Prozessors darstellt. Liegt der Wert nahe m, kann von einem Engpass an CPU-Ressourcen ausgegangen werden.

Wartezeiten bei den Prozessorressourcen entstehen dann, wenn eine Anfrage einer Komponente von dem entsprechenden Prozessor-Task nicht sofort bedient werden kann, da alle Bedienstationen (Prozessor-Threads) beschäftigt sind. Als Beispiel seien die CPU-Ressourcen aus Szenario 2 betrachtet, die ab ca. 40 simultan agierenden Benutzern ausgereizt sind. Die Server-CPU im LQN-Modell wird entsprechend mit einer Multiplizität von 40 parametrisiert, da die CPU lediglich 40 Applikations-Threads (und somit 40 simultane Benutzer) zeitgleich bedienen kann.

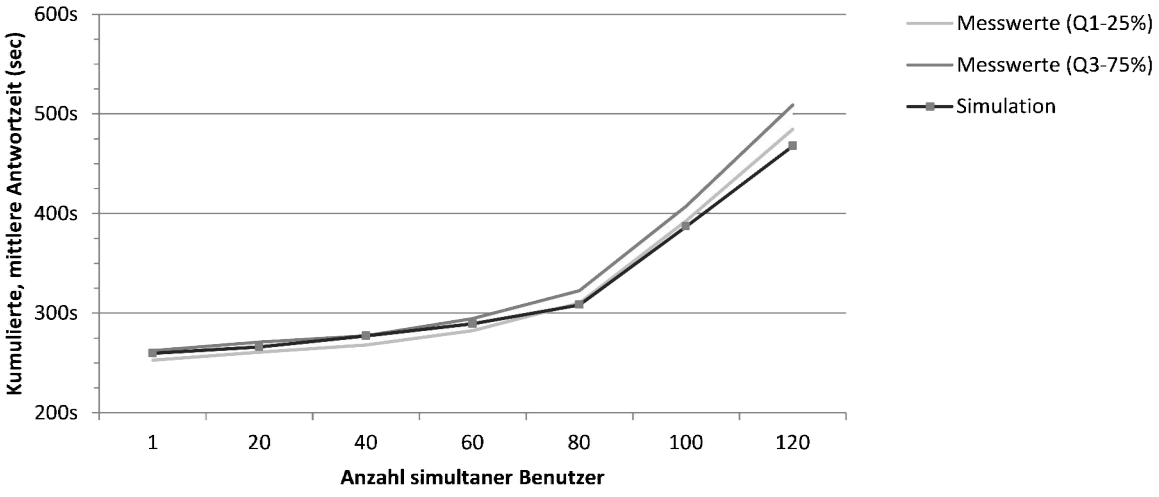
Zusammenfassend wird somit eine Vielzahl an statistischen Informationen zu den einzelnen Modellkomponenten dargestellt. Die in dieser Arbeit betrachtete Metrik Antwortzeit wird anhand von Mittelwerten sowie deren Verteilungen wiedergegeben. Antwortzeiten werden in der Ausgabedatei als Service-Zeiten bezeichnet und sind, wie bereits ausführlich erläutert, nicht mit den Service-Zeiten der Parametrisierung zur verwechseln. Sie stellen die gesamte Ausführungszeit dar. Im konkreten Fall des vorgestellten Modells müssen von der Service-Zeit der einzelnen Interaktionsschritte (dargestellt als Aktivitäten im Benutzer-Task mit einer Service-Zeit von 0) potentielle Denkzeiten abgezogen werden, um die Antwortzeit zu erhalten. Die Summe der einzelnen Antwortzeiten ergibt die Gesamtantwortzeit der Fallstudie (ohne Denkzeiten). Diese aggregierten Werte werden im folgenden Kapitel mit der Summe der gemessenen Antwortzeiten der Fallstudie verglichen.

5.7 Vergleich der Mess- und Simulationsergebnisse

Die Messwerte der beiden vorgestellten Szenarien werden in diesem Kapitel mit den Simulationswerten verglichen. Wie bereits in Kapitel 2.3.1 beschrieben, wird für die in dieser Arbeit durchgeführten Leistungsanalysen die Antwortzeit als Vergleichswert betrachtet. Daher wird in den folgenden Diagrammen die Gesamt-Antwortzeit der modellierten Fallstudie dargestellt, die sich aus den Antwortzeiten der einzelnen Interaktionsschritte zusammensetzt.

Die beiden Szenarien sollen, wie bereits in Kapitel 5.4 beschrieben, einen unterschiedlichen Schwerpunkt setzen. Mit der Gegenüberstellung von simulierten und gemessenen Werten wird dabei geprüft, ob das Modell auf die in Kapitel 4.1.2 identifizierten Einflussgrößen entsprechend reagiert.

5.7.1 Szenario 1 – Ausreichende Systemressourcen



Sim (μ)	259,722	266,09	277,189	289,196	308,252	386,882	467,859
Q1 (25%)	252,829	260,829	267,829	282,287	310,287	392,744	484,573
Q3 (75%)	262,573	270,573	277,573	294,466	322,466	407,359	508,932

Abbildung 5-19: Vergleich der Simulations- und Messwerte (Szenario 1)
Quelle: eigene Darstellung

Im ersten Szenario (siehe Abbildung 5-19) sind genügend Systemressourcen vorhanden, sodass lediglich das Portalsystem durch eine steigende Anzahl von Benutzern unter Last gesetzt wird. Wie bereits aus den Messungen in Kapitel 5.5.4 ersichtlich wird, erfolgt ein leichter Anstieg der Antwortzeiten aufgrund von zunehmenden Sperrsituationen, die durch die erhöhte Anzahl an simultan agierenden Benutzern hervorgerufen werden. Dieser Anstieg ist im Vergleich zur Gesamtantwortzeit verhältnismäßig gering, jedoch deutlich erkennbar. Das überschaubare Ausmaß hängt mit der Anzahl bzw. dem Gewicht der Datenbankanfragen zusammen, die sich beim verwendeten Workload in Grenzen halten. Bei datenbankintensiven Lastmustern würde sich ein deutlicherer Anstieg der Antwortzeiten aufgrund von Sperrsituationen zeigen.

Den entscheidenden Zuwachs des Antwortzeitenanstiegs erhält das System bei mehr als 80 simultan agierenden Benutzern. Dies hängt zum einen mit der Anzahl an konfigurierten Applikations-Threads zusammen, zum anderen mit der für diesen Workload geltenden 1-zu-1-

Beziehung zwischen verwendeten Applikations-Threads und Benutzern. Mit anderen Worten ist aufgrund der Begrenzung von maximal 80 Applikations-Threads die maximale Parallelität der durchgeführten Lastschritte erreicht. Sobald mehr als 80 Lastschritte zeitgleich durchgeführt werden sollen, können keine freien Applikations-Threads für die zusätzlichen Anfragen bereitgestellt werden und in der Folge treten Wartezeiten bei der Bearbeitung der Lastschritte auf. Da jeder Benutzer sequentiell einen Lastschritt nach dem anderen durchführt, belegt jeder Benutzer maximal einen Applikations-Thread. Sieht man von evtl. Denkzeiten ab, bei denen kein Applikations-Thread belegt wird, gilt die genannte 1-zu-1-Beziehung und somit die Aussage, dass bei mehr als 80 Benutzern der Bedarf an parallelen Applikations-Threads höher ist als die gegebene Multiplizität von 80.

Das soeben beschriebene Verhalten wird von der Simulation erfasst, sodass ab 80 Benutzern ein deutlich höherer Anstieg zu verzeichnen ist. Im Bereich zwischen 100 und 120 Benutzern nimmt die Simulationsgenauigkeit jedoch leicht ab. Der Grund für die zunehmende Ungenauigkeit kann zum einen einer nicht auszuschließenden wachsenden Ungenauigkeit des modellierten Sperrmanagements geschuldet sein, zum anderen an äußeren, nicht modellierten Systemeinflüssen liegen, die trotz ausreichender Systemressourcen auftreten.

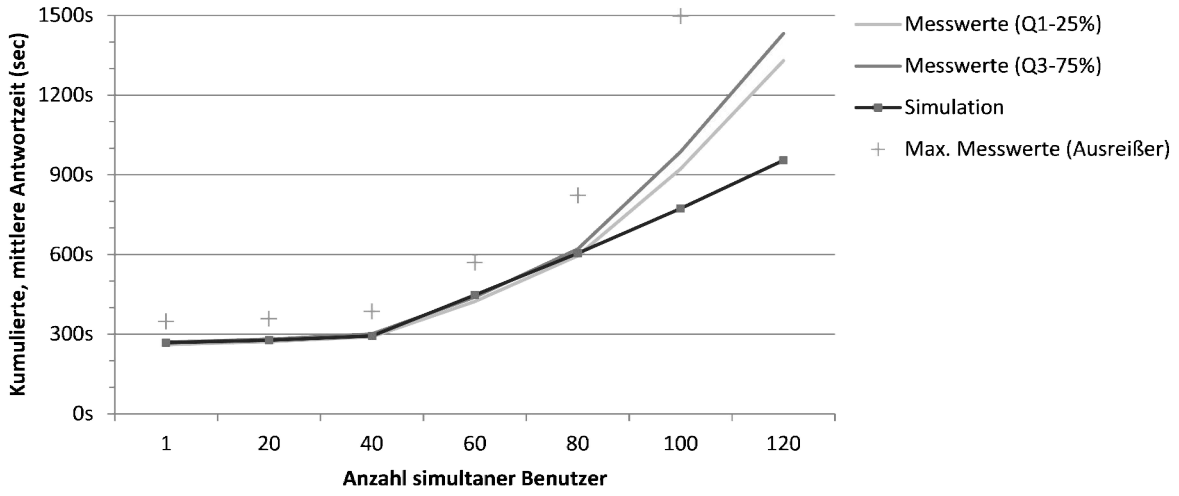
Für eine vollständige Betrachtung seien auch die verbleibenden Modellkomponenten angesprochen, die in diesem Szenario keinen (nennenswerten) Einfluss auf das Leistungsverhalten zeigen. Die CPU-Ressourcen sind, wie bereits erwähnt, nicht gekappt. Daher entsteht in diesem Bereich kein Engpass. Ebenfalls ist der für die JVM zur Verfügung gestellte Hauptspeicher größer als notwendig.

Auch die Größe des Tabellenpuffers ist ausreichend, sodass sich die Trefferrate mit einer zunehmenden Anzahl an Benutzern nicht reduziert. Der stetige Wert von knapp 99 Prozent entspricht dem in der Praxis erreichten und in der Literatur für ein gut konfiguriertes System vorgeschlagenen Wert. Da sich die Trefferrate nicht verändert, hat eine steigende Anzahl an Benutzern keinen Einfluss auf die Antwortzeit.

Die Datenbank wird als Black-Box betrachtet. Daher wurden genügend Systemressourcen für den Datenbank-Host zur Verfügung gestellt. Die Service-Zeiten der einzelnen SQL-Abfragen bleiben konstant und haben folglich keinen Einfluss auf die Antwortzeit. Ein möglicher limitierender Faktor ist die Anzahl der zur Verfügung gestellten Datenbankverbindungen. Allerdings zeigt die Nutzung, dass die vorhandenen 20 Verbindungen für diesen Workload ausreichen.

Dieses Szenario zeigt, dass sowohl der modellierte Sperrmechanismus, als auch die Multiplizität des Lastschritte-Tasks die realen Systemkomponenten (Sperrkonzept des SAP-Netweaver-Portal-Systems, Anzahl der Applikations-Threads) entsprechend abbilden. Es zeichnet sich durch den geringen, aber stetigen Anstieg der Antwortzeiten aufgrund von zunehmenden Sperrsituationen und dem Knick bei 80 simultanen Benutzern, der die erreichte maximale Parallelität der Applikations-Threads darstellt, aus.

5.7.2 Szenario 2 – Knappe Systemressourcen



Sim (μ)	267,508	276,934	293,339	447,584	604,324	773,128	954,559
Q1 (25%)	260,603	271,829	290,012	423,087	594,802	922,119	1329,207
Q3 (75%)	270,573	280,985	300,730	439,529	620,379	986,671	1431,516

Abbildung 5-20: Vergleich der Simulations- und Messwerte (Szenario 2)
Quelle: eigene Darstellung

Das zweite Szenario beschäftigt sich mit dem Fall, dass diverse Systemressourcen verknappen. Dies betrifft folgende Einflussgrößen:

- Die CPU-Ressourcen werden auf drei Kerne gekappt. Den Messungen zufolge entsprechen diese Ressourcen dem Bedarf, der von 40 simultanen Benutzern bei der Ausführung des modellierten Workloads benötigt wird.

  Wie bereits in Kapitel 5.5.5.2 dargestellt wurde, liegt der Maximalwert an erreichter CPU-Nutzlast bei ca. 85 bis 90 Prozent. Unter einer idealisierten Annahme würde der lineare Anstieg der CPU-Auslastung erst bei 44 Benutzern die Volllast und somit 100 Prozent erreichen, die auch von den Grundannahmen der Warteschlangennetze gefordert werden (siehe Kapitel 4.1). Eine exakte Parametrisierung der Prozessor-Multiplizität im LQN-Modell kann somit nur dann erfolgen, wenn Messdaten zur Verfügung stehen. Ansonsten kann diese nur durch eine Abschätzung erfolgen und würde der Literatur entsprechend bei 75 bis 80 Prozent angesetzt werden. Für die Simulation dieses Szenarios wurde der gemessene Wert parametrisiert und somit eine Multiplizität $m_{CPU} = 40$ verwendet. Dies stellt sicherlich eine idealisierte Sicht dar, da in der Praxis eben solche Messdaten meist nicht zur Verfügung stehen. Nichtsdestotrotz hätte eine angenommene maximale Nutzlast von 75 bis 80 Prozent zu einer Parametrisierung von ca. $m_{CPU} = 37$ geführt und die Simulationsergebnisse nur leicht verschoben.

- Der Hauptspeicher ist zwar immer noch ausreichend für die Speicherallokation der JVM, allerdings sorgt die Verknappung zu erhöhten GC-Aktivitäten im Hochlastbereich.

- Die Größe des Tabellenpuffers wurde reduziert, sodass nicht alle gepufferten Objekte im Tabellenpuffer Platz finden und es folglich zu Verdrängungen kommt.

Im Bereich zwischen 1 bis 40 Benutzern ist auch in diesem Szenario ein leichter Anstieg des Antwortzeitenzuwachses zu verzeichnen, der sich analog zu dem ersten Szenario über zunehmende Sperrsituationen erklären lässt. Hinzu kommt die mit steigender Benutzeranzahl abnehmende Trefferrate des Tabellenpuffers. Die erhöhte Grundantwortzeit bei einem Benutzer ist ebenfalls auf den Tabellenpuffer zurückzuführen, da die Trefferquote bereits bei einem Benutzer bei gut 80 Prozent liegt und somit geringer ist als im ersten Szenario.

Ab 40 simultanen Benutzern wirkt sich die erwähnte Begrenzung der CPU-Ressourcen auf die Antwortzeiten aus. Von nun an treten Wartezeiten am Prozessor auf, die sich auch in den Simulationsergebnissen im entsprechenden Abschnitt über Nutzung und Wartezeiten der Prozessoren ablesen lassen. Die Begrenzung der CPU-Ressourcen sowie die Begrenzung der maximalen Anzahl an Applikations-Threads stellen somit die Schranken dar, bei denen Wartezeiten für die Abarbeitung der Lastschritt-Anfragen entstehen. Im ersten Szenario wurde zuerst die Schranke der Applikations-Thread-Begrenzung erreicht (bei 80 Benutzern, CPU-Ressourcen waren nicht gekappt), während im zweiten Szenario zwar 80 Applikations-Threads konfiguriert sind, die CPU-Ressourcen jedoch bereits bei 40 simultanen Benutzern voll ausgelastet sind. Aufgrund des direkten Zusammenhangs zwischen Benutzern mit identischem Workload und 1 Applikations-Thread pro Benutzer entsprechen im übertragenen Sinne die gegebenen CPU-Ressourcen einer maximalen Parallelität von 40 Applikations-Threads (ohne Wartezeiten).

Betrachtet man die Auslastung (Nutzung) der Server-CPU-Ressource bei 40 Benutzern in den Simulationsergebnissen, erreicht diese einen Wert von ca. 36 von möglichen 40 (siehe Listing 5-12). Die Differenz lässt sich mit den Sperrzeiten erklären, bei der der Applikations-Thread eines Lastschrittes keine CPU-Last erzeugt, ähnlich zu einer blockierenden I/O-Operation.

```
  1  Utilization and waiting per phase for processor:  Processor_Server
  2
  3  Task Name     Pri    n     Entry Name     Utilization    […]
  4  W1            0      80    W1E1           0.7142
[…]
 53  Task Total:                                 36.1034
[…]
```

Listing 5-12: Beispielhafte Auslastung der Server-CPU-Ressource bei 40 Benutzern (Szenario 2)
Quelle: eigene Darstellung

Von nun an steigen die Antwortzeiten in den Simulationsergebnissen ziemlich linear, der leichte zusätzliche Anstieg ist den Sperrsituationen und der abnehmenden Trefferquote des Tabellenpuffers geschuldet. Im Gegensatz dazu zeigen die Messwerte vor allem ab ca. 70 bis 80 Benutzern einen beginnenden Anstieg des Antwortzeitenzuwachses. Sobald mehr als 100 Benutzer gleichzeitig agieren, ist der stark exponentielle Anstieg deutlich zu erkennen. Ebenso schwanken die Messergebnisse zunehmend, sodass die Maximalwerte gemessener Antwortzeiten (obere Ausreißer) bei 120 Benutzern einen Wert bis zu 2.240 Sekunden annehmen und folglich beinahe dem Doppelten des Mittelwertes entsprechen. In diesem Bereich liegen die Simulationsergebnisse deutlich unterhalb der gemessenen Zeiten. Der Unterschied lässt sich mit einem starken System-Overhead erklären (vgl. Jehle 2010, 185f.), der von dem

Simulationsmodell nicht erfasst wird. Des Weiteren weisen die GC-Traces einen starken Zuwachs der GC-Aktivitäten auf, die ebenso nicht vom LQN-Modell erfasst werden, da entsprechende GC-Pausen als konstanter Faktor nur in den Service-Zeiten enthalten sind (vgl. Kapitel 4.2.6). Die im folgenden Kapitel durchgeführte Analyse der GC-Aktivitäten beschäftigt sich ausführlich mit diesem Sachverhalt.

Zusammenfassend zeigt sich, dass bei knappen Systemressourcen die Simulationswerte im Hochlastbereich keine ausreichende Genauigkeit erzielen. Die Hauptursache liegt in der fehlenden Modellierung der äußeren System-Einflüsse und ist Bestandteil weiterer Untersuchungen. Wie bereits in Kapitel 4.2.6 aufgezeigt wurde, ist die LQN-Modellierung des Garbage-Collectors ein mühsames und fehlerträchtiges Unterfangen. In der Literatur ist kein Ansatz zur GC-Modellierung mittels LQN gegeben und der GC-Zeitanteil stets implizit in den Service-Zeiten enthalten (siehe z.B. Franks/Lau/Hrischuk (2011), Ufimtsev (2006), Ufimtsev/Murphy (2006), Xu et al. (2005)). Daher wurde in dieser Arbeit ein Ansatz zur Analyse der GC-Auswirkungen (mittels Modellierung der Einflussgrößen GC-Intervall und GC-Dauer) vorgestellt, der jedoch nicht die dynamischen Aspekte des GC-Verhaltens abbildet und auf vorhandene Messdaten angewiesen ist.

Ebenso ist die Simulationsgenauigkeit von der Qualität der Portalsystem-Konfiguration abhängig, so hat beispielsweise eine nicht ausreichende Tabellenpuffergröße einen negativen Einfluss auf die Simulationsgenauigkeit. Dies liegt wiederum, wie bereits in Kapitel 4.2.4 dargestellt, an der Schätzung der Trefferrate, deren exakte Berechnung bei fehlenden Aufzeichnungen des Tabellenpuffers nicht möglich ist.

Unabhängig davon reagieren die Simulationsergebnisse den Einflussgrößen entsprechend und erzielen bei einer mäßigen Auslastung der Ressourcen eine vielversprechende Genauigkeit. Eine detailliertere Darstellung und Interpretation der Ergebnisse erfolgt in Kapitel 6.1.

5.8 Analyse des Garbage-Collectors und System-Overheads

Die zunehmende Ungenauigkeit der Simulationswerte im Hochlastbereich führt zu dem Schluss, dass Einflüsse außerhalb des Portalsystems zu einer Beeinträchtigung der Leistungsfähigkeit des Systems führen. Neben dem Overhead auf Betriebssystemebene, der auch bei Jehle (2010, 185f.) beschrieben ist und im weiteren Verlauf dieses Kapitels betrachtet wird, lässt sich eine weitere Einflussgröße auf der Ebene der JVM feststellen, der Garbage-Collector. Zur Quantifizierung der Aktivitätsintensität werden die Kenngrößen durchschnittliche Dauer eines GC-Laufes und durchschnittliches Intervall zwischen den GC-Läufen betrachtet.

Die nachfolgenden Datenauswertungen und Diagramme des Garbage-Collectors wurden mit dem von IBM zur Verfügung gestellten Hilfsmittel „Garbage Collector and Memory Analyzer“ (siehe Kapitel 3.2.6) der IBM-Support-Assistant-Plattform (IBM 2011c) erstellt. Die Daten zur CPU-Auslastung wurden über das Betriebssystem-Hilfsmittel „topas_nmon“ aufgezeichnet.

5.8.1 Garbage-Collector-Aktivität

Bei einem hohen Füllgrad des Hauptspeichers nehmen die GC-Aktivitäten deutlich zu und beeinflussen dementsprechend das Leistungsverhalten des SAP-Netweaver-Enterprise-Portals. Diese Beobachtung deckt sich mit den Erkenntnissen aus der Literatur (z.B. Davis et al. 2006). Wie die folgenden Auswertungen zeigen, führen die GC-Läufe immer wieder zu Einbrüchen der CPU-Nutzung.

Für die Untersuchung des Garbage-Collectors wurden die Traces aus Szenario 2 bei 80 und 100 simultanen Benutzern verglichen. Zur Aktivierung der Traces wurde der standardmäßig eingeschaltete Parameter

- *Verbose:gc*

genutzt. Der Heap-Speicher wurde auf einen festen Wert gesetzt:

- *Xms1536* (maximale Heap-Größe)
- *Xmx1536* (minimale Heap-Größe)

Ebenso wurden die in Davis et al. (2006, 14) vorgeschlagenen Tuning-Parameter für ein SAP-Netweaver-Enterprise-Portal auf einer IBM-Power-Architektur verwendet:

- *Xloratio0.05* (minimale Large-Object-Heap-Größe)
- *Xconmeter0* (Auslöser für die Garbage-Collection)
- *Xconcurrentbackground1* (einzelner Thread für Concurrent-GC)
- *Xgcthreads1* (Anzahl GC-Threads)
- *Xconcurrentlevel4* (Startzeitpunkt des GCs)
- *Xparroot* (Reduziert GC-Pausen)

Die nachstehenden Diagramme zeigen die Kenngrößen GC-Dauer und GC-Intervalle für die betrachteten Benutzerzahlen.

Szenario 2 – 80 Benutzer

Bei 80 simultanen Benutzern wird im Schnitt nach ca. 29,2 Sekunden ein GC-Lauf mit einer mittleren Dauer von ca. 1,2 Sekunden gestartet (siehe Tabelle 5-2).

	Dauer (s)	Mark (s)	Sweep (s)	Intervall (s)
μ	1,211	1,013	0,198	29,276
σ	0,933	0,869	0,105	2,405
Min.	0,269	0,194	0,075	25,092
Max.	4,078	3,726	0,453	33,728

Tabelle 5-2: Statistische Daten zum Garbage-Collector (Szenario 2, 80 Benutzer)
Quelle: eigene Darstellung

Die Entwicklung der GC-Dauer kann in Abbildung 5-21 betrachtet werden.

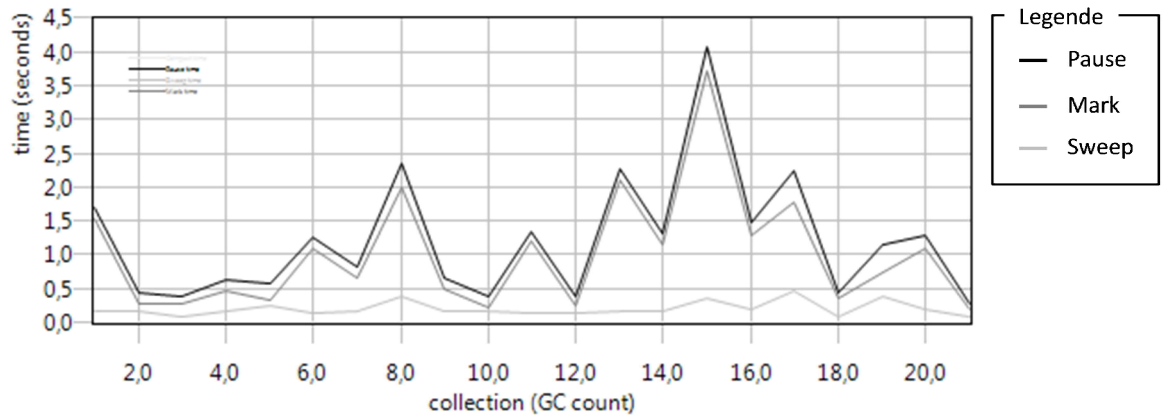


Abbildung 5-21: Dauer der Garbage-Collector-Läufe (Szenario 2, 80 Benutzer)
Quelle: eigene Darstellung

Das Intervall zwischen den GC-Läufen bleibt relativ konstant zwischen 25 und 34 Sekunden (siehe Abbildung 5-22).

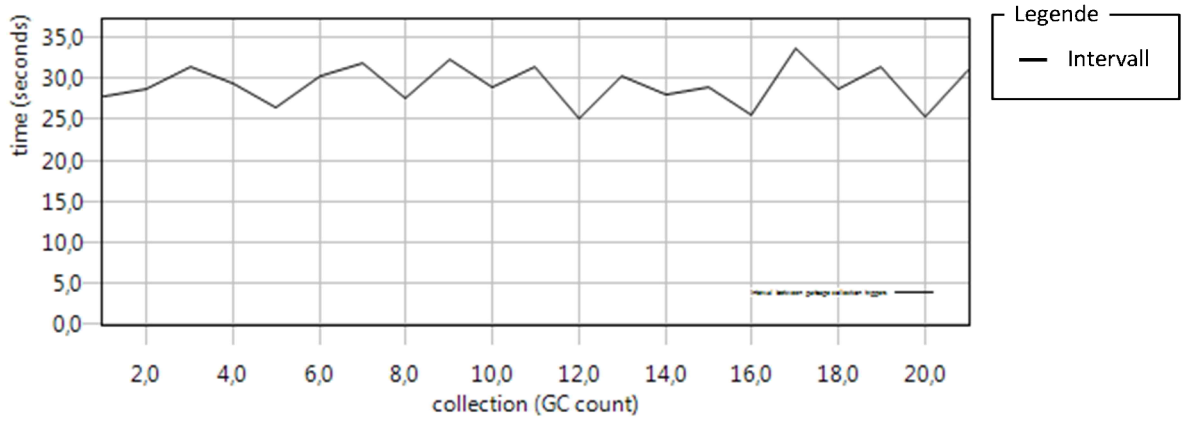


Abbildung 5-22: Intervall zwischen den Garbage-Collector-Läufen (Szenario 2, 80 Benutzer)
Quelle: eigene Darstellung

Szenario 2 – 100 Benutzer

Bei 100 simultanen Benutzern nehmen die GC-Aktivitäten deutlich zu. Im Schnitt wird nach ca. 19,6 Sekunden ein GC-Lauf mit einer mittleren Dauer von ca. 1,4 Sekunden gestartet (siehe Tabelle 5-3).

	Dauer (s)	Mark (s)	Sweep (s)	Intervall (s)
μ	1,423	1,198	0,225	19,603
σ	1,372	1,324	0,128	1,980
Min.	0,158	0,077	0,055	15,420
Max.	5,470	5,277	0,616	23,895

Tabelle 5-3: Statistische Daten zum Garbage-Collector (Szenario 2, 100 Benutzer)
Quelle: eigene Darstellung

In Abbildung 5-23 kann die Entwicklung der GC-Dauer betrachtet werden. Im Vergleich zur Dauer bei 80 Benutzern ist die durchschnittliche Dauer nur geringfügig erhöht.

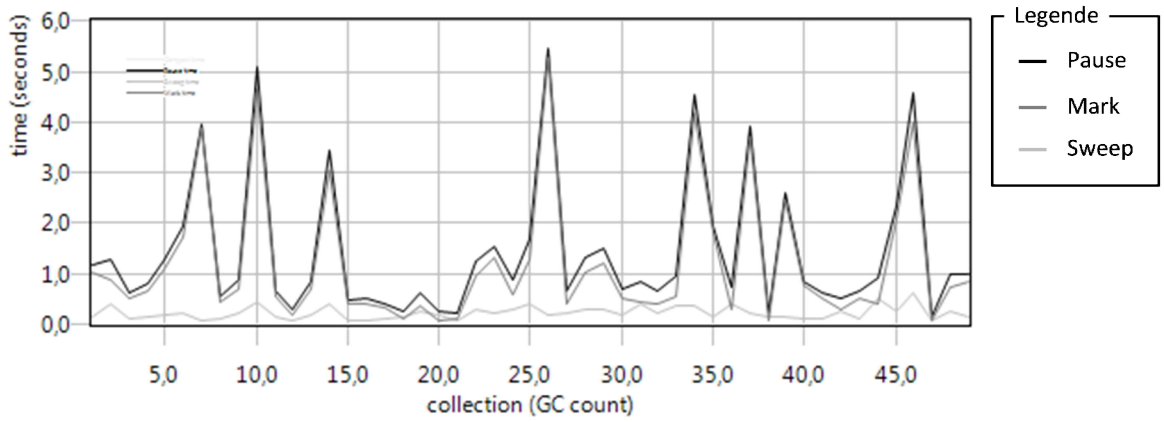


Abbildung 5-23: Dauer der Garbage-Collector-Läufe (Szenario 2, 100 Benutzer)
Quelle: eigene Darstellung

Allerdings ist das Intervall zwischen den GC-Läufen um ca. 10 Sekunden gesunken und stellt somit einen überproportionalen Anstieg der GC-Aktivitäten dar (siehe Abbildung 5-24).

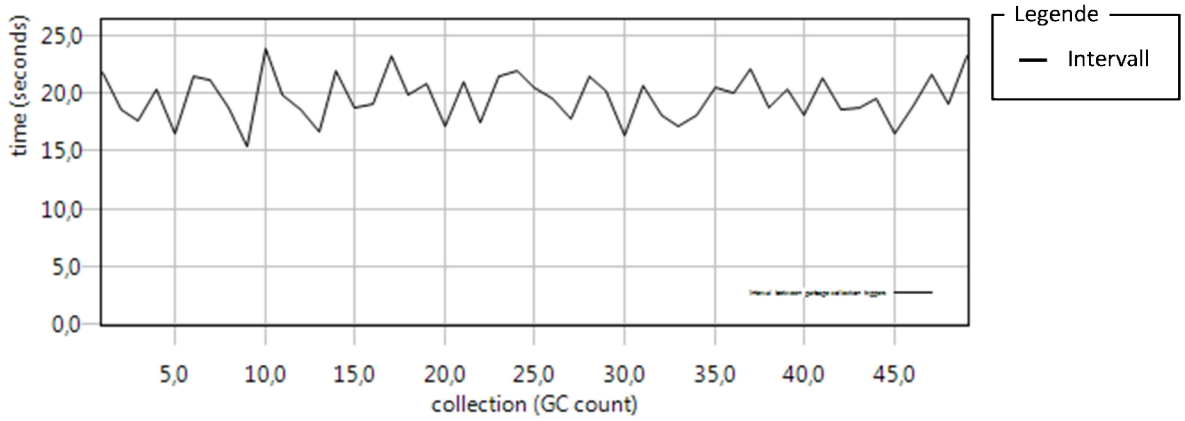


Abbildung 5-24: Intervall zwischen den Garbage-Collector-Läufen (Szenario 2, 100 Benutzer)
Quelle: eigene Darstellung

Vergleicht man die Aktivitätsintensität der GC-Läufe, ist ein überproportionaler Anstieg zu erkennen:

$$\frac{1{,}2}{29{,}3} * 100 \quad \ll \quad \frac{1{,}4}{19{,}6} * 80$$

Dieser Anstieg führt zu dem Schluss, dass der implizite GC-Einfluss in den Bearbeitungszeiten nicht ausreicht. Daher werden die Näherungsparameter GC-Dauer und -Intervall zur Spezifikation der Aktivitätsintensität, wie in Kapitel 4.2.6 dargestellt, im LQN-Modell integriert.

Es wird auf eine Darstellung der damit erreichten Antwortzeiterhöhung verzichtet, da sich dieser Effekt direkt von der Struktur sowie den verwendeten Parameterwerten der bereits vor-

gestellten GC-Modellierung ableitet. Die statische Modellierung des GC-Einflusses dient lediglich der speziellen GC-Untersuchung und löst nicht die für eine nicht gemessene Lastintensität unbekannte Variable des GC-Einflusses. Mit anderen Worten ist das vorgestellte Konzept der GC-Modellierung nur dann sinnvoll, wenn explizite Messdaten zur Verfügung stehen oder künstlich erhöhte GC-Aktivitäten zu Testzwecken eingesetzt werden. Eine explizite Modellierung des Garbage-Collectors, die dessen dynamische Eigenschaften allgemein abbildet, bleibt weiterhin offen und ist Bestandteil weiterer Untersuchungen.

5.8.2 System-Overhead

Neben dem überproportionalen Zuwachs der GC-Aktivitäten steigt mit zunehmender Auslastung der Overhead auf Betriebssystemebene. In den folgenden zwei Diagrammen ist die CPU-Auslastung über einen Zeitraum von fünf Minuten für die betrachteten Benutzerzahlen dargestellt. Die Einbrüche der CPU-Auslastung erklären sich über die soeben dargestellten GC-Aktivitäten. Wie man deutlich erkennen kann, steigt der Anteil des System-Overheads und führt somit zu einer weiteren Beeinträchtigung der Leistungsfähigkeit.

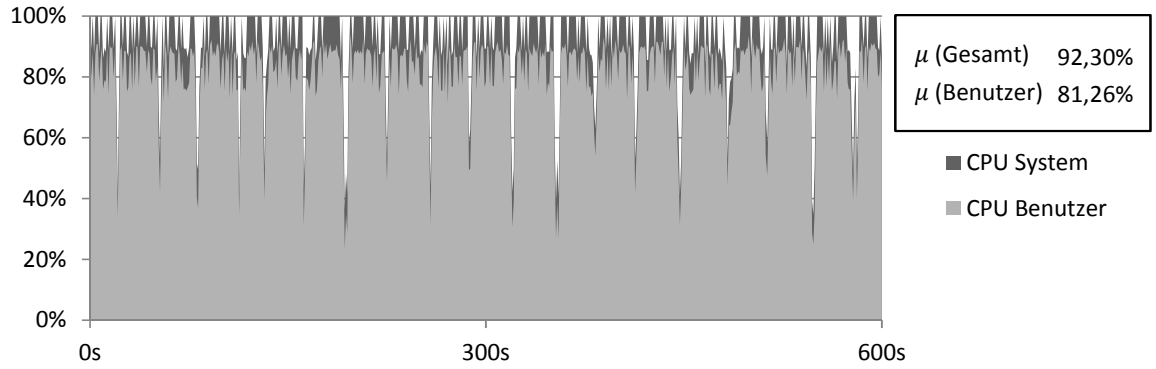


Abbildung 5-25: Vergleich zwischen Benutzer- und System-CPU-Zeit (Szenario 2, 80 Benutzer)
Quelle: eigene Darstellung

In Abbildung 5-25 lassen sich deutlich die GC-Läufe in einem durchschnittlichen 30-Sekunden-Intervall erkennen. Während der Ausführung der Garbage-Collection fällt die CPU-Auslastung auf bis zu 25 bis 30 Prozent. Der System-Overhead beläuft sich auf ca. 10 Prozent.

Bei 100 Benutzern steigt der Overhead deutlich an (siehe Abbildung 5-26). Auch hier fallen die GC-Zyklen sofort ins Auge, die in einem Intervall von ca. 20 Sekunden für einen Einbruch der CPU-Auslastung sorgen.

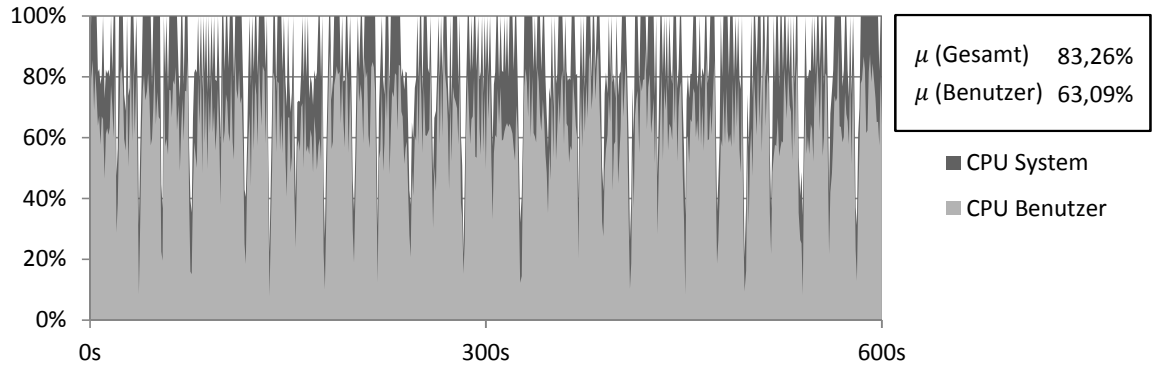


Abbildung 5-26: Vergleich zwischen Benutzer- und System-CPU-Zeit (Szenario 2, 100 Benutzer)
Quelle: eigene Darstellung

5.9 Zusammenfassung

Kapitel 5 beschäftigte sich mit der Simulation und Evaluation des LQN-Modells anhand der am SAP UCC der TU München eingesetzten Portalfallstudie und dient somit der Beantwortung von Forschungsfrage 3. Dazu wurden zu Beginn die Schulungsinhalte vorgestellt und daraus der modellierte Workload abgeleitet. Mit der Eingrenzung auf interne Portaloperationen haben sich die in Tabelle 5-1 aufgelisteten Portalfunktionen ergeben, die nach ihrer Service-Zeit, Varianz sowie den durchgeführten Datenbankaufrufen modelliert und parametrisiert wurden. Für die Lasterzeugung wurde der von Jehle (2010) entwickelte Lastgenerator verwendet.

Bereits in den Telefongesprächen mit einem SAP-Mitarbeiter der Java-Performance-Gruppe konnte das zu erwartende Verhaltensmuster des Portals skizziert werden. Wenngleich keine LQN-Simulationen durchgeführt wurden, wurden Key-Performance-Indikatoren ermittelt (Cheng 2008), die in dieser Arbeit als Grundlage für die Auswahl der zu modellierenden Komponenten dienten (vgl. Kapitel 4.1.2).

Die Leistungsanalyse des Portalsystems sowie die Simulation des Modells wurden für zwei unterschiedliche Szenarien durchgeführt, die den Schwerpunkt auf unterschiedliche Untersuchungsgegenstände setzen. Das erste Szenario kann als grundlegende Evaluation des Leistungsverhaltens des Portalsystems sowie der Simulationsergebnisse verstanden werden, da sich die zu untersuchende Systemumgebung auf ein gut konfiguriertes Portalsystem mit ausreichenden Systemressourcen bezieht. Dazu wurden genügend CPU-Ressourcen und eine ausreichende Tabellenpuffergröße zur Verfügung gestellt, damit zusätzliche Einflussfaktoren vermieden werden.

Das zweite Szenario beschäftigt sich mit der Frage, wie sich das Portalsystem bei knappen Systemressourcen verhält und welchen Einfluss dies auf die Simulationsgenauigkeit nimmt. Für die Untersuchung wurden daher zum einen die CPU-Ressourcen stark eingegrenzt. Dies ist mit der verwendeten, virtualisierten Systemumgebung von IBM ohne nennenswerten Aufwand möglich, da die Systemressourcen eines virtuellen Hosts dynamisch zur Laufzeit verändert werden können. Des Weiteren wurde die Größe des Heap-Speichers reduziert, sodass der erhöhte Füllgrad verstärkte Aktivitäten des Speichermanagements bewirkt. Schließlich wurde die Größe des Tabellenpuffers reduziert, sodass es zu Verdrängungen und Invalidierungen der Pufferobjekte kommt, die zu einer erhöhten Anzahl an Datenbankaufrufen führen.

Anschließend wurde der Fokus auf äußere Einflussgrößen gelegt. Dies betrifft auf der einen Seite den von Jehle (2010) festgestellten Overhead des Betriebssystems, auf der anderen Seite den zunehmenden Einfluss des Speichermanagements, der aufgrund der Stop-the-World-Pause zu Einbrüchen der CPU-Nutzung führt. Aufgrund der in der LQN-Literatur fehlenden Betrachtung des Garbage-Collectors wurde eine Modellierung der identifizierten GC-Einflussgrößen GC-Dauer und GC-Intervall angestrebt. Der Modellierungsansatz stellt eine Möglichkeit dar, die Aktivitätsintensität des Garbage-Collectors statisch festzulegen, statt sie wie in der Literatur üblich impliziten über die Bearbeitungszeit der Lastschritte zu erfassen. Es sei an dieser Stelle erneut darauf hingewiesen, dass dieser Ansatz keine allgemeine Abbildung des dynamischen GC-Verhaltens darstellt, sondern lediglich die Möglichkeit bietet, über die beiden Parameter GC-Dauer und GC-Intervall die Aktivitätsintensität des Garbage-Collectors zu parametrisieren. Dies ist vor allem dann hilfreich, wenn Erfahrungswerte existieren und die Simulation einer erhöhten Benutzeranzahl mögliche Engpässe aufdecken soll.

Allgemein lässt sich festhalten, dass bei einem gut konfigurierten Portalsystem unter normalen Lastbedingungen die Simulation vielversprechende Resultate liefert. Dies zeigen die Ergebnisse des Vergleichs der Mess- und Simulationswerte aus Szenario 1. In Szenario 2 wird deutlich, dass erschwerte Bedingungen wie knappe Systemressourcen oder eine mangelhafte Portalkonfiguration (wie z.B. eine nicht ausreichende Tabellenpuffergröße) die Simulationsgenauigkeit stark beeinträchtigen. Dies erklärt sich nicht zuletzt direkt aus den betrachteten Modellkomponenten und den Grundannahmen der Warteschlangennetze (siehe Kapitel 4.1).

Die Ergebnisse der Evaluation zeigen jedoch auch, dass die eintretenden Engpässe ab einer bestimmten Lastintensität, hervorgerufen durch eine bestimmte Anzahl von simultan agierenden Benutzern, von der Simulation erkannt werden. Im ersten Szenario wurde dies beispielsweise durch die Begrenzung der parallelen Applikations-Threads hervorgerufen, im zweiten Szenario konnten die Auswirkungen der begrenzten CPU-Ressourcen erfasst werden. Nicht zuletzt wurde der Einfluss des Sperrmanagements abgebildet, sodass der Anstieg der Antwortzeiten durch zunehmende Sperrsituationen ebenfalls in den Simulationsergebnissen widergespiegelt wird.

6 Fazit

Vor dem Hintergrund, die Leistungsfähigkeit von Computersystemen zu bewerten, haben sich im wissenschaftlichen Umfeld verschieden Methoden der Performance-Evaluation etabliert (vgl. Kapitel 2.3). In Bezug auf ERP-Systeme und Unternehmenssoftware im Allgemeinen werden diese Möglichkeiten in der Praxis jedoch nur teilweise genutzt. Die Analyse beschränkt sich hauptsächlich auf die Überwachung aktueller Leistungsdaten mittels integrierter Funktionen (vgl. Kapitel 1.1). Ziel dieser Arbeit war es daher, das Leistungsverhalten eines SAP-Netweaver-Portal-Systems mit den auf den Warteschlangennetzen basierenden LQNs zu beschreiben und durch Simulation des Performance-Modells die Leistung bei steigenden Benutzerzahlen zu bewerten. In diesem Kapitel werden nun die erzielten Ergebnisse zusammengefasst und bewertet. Abschließend werden die mit den Ergebnissen einhergehenden Limitationen aufgezeigt und ein Ausblick auf weitere Forschungsmöglichkeiten gegeben.

6.1 Bewertung und Interpretation der Ergebnisse

In Kapitel 3 wurde die in Forschungsfrage 1 gestellte Frage, wie ein SAP-Netweaver-Portal-System für die Performance-Modellierung und Simulation charakterisiert werden kann und welche Analyseinstrumente die benötigten Informationen bereitstellen, beantwortet. Dabei wurden einzelne Komponenten des Systems betrachtet, die in der Literatur als Performance-Indikatoren genannt werden. Mithilfe den theoretischen Grundlagen aus Kapitel 2, den Erfahrungen am SAP UCC der TU München sowie der SAP-Systemdokumentation konnten Methoden zur Leistungsdatenerfassung und -auswertung gesammelt werden. Die Qualität der Daten ist für die Parametrisierung der Modellkomponenten ausreichend, wenngleich nicht immer eine nicht intrusive Monitoring-Lösung eingesetzt werden konnte. Ebenso stellen die vorhandenen Möglichkeiten zur Leistungsdatenerhebung eine Limitation bei der Auswahl der Granularität dar, diese genügen jedoch den Anforderungen des in dieser Arbeit vorgestellten komponentenorientierten Ansatzes. Eine Übersicht der Komponenten, Einflussfaktoren sowie Möglichkeiten zur Datenerfassung wurde in Tabelle 4-1 dargestellt.

Die Beantwortung der Forschungsfrage 2, wie ein SAP-Netweaver-Portal-System mittels LQN modelliert und parametrisiert werden kann, wurde in Kapitel 4 behandelt. Dabei wurden zu Beginn die Grundannahmen, die sich direkt aus der Warteschlangentheorie ableiten, dargestellt. Sowohl die Erfahrungen in der Literatur (zum Beispiel Cheng (2008)) als auch die in dieser Arbeit erhobenen Leistungsdaten zeigen, dass sich Warteschlangennetze für die Modellierung eines SAP-Netweaver-Java-Systems bis zu einem hohen Auslastungsgrad eignen. Erst im Voll- und Überlastbereich bedingen äußere Einflüsse ein Systemverhalten, das sich nicht vollständig mit den idealisierten Grundannahmen deckt. Die Modellierung des Java-Speichermanagements ist mit LQN nur teilweise möglich. Ein Ansatz zur statischen Modellierung wurde in dieser Arbeit vorgestellt, der in bestimmten Situationen Vorteile gegenüber der in der Literatur stets verwendeten impliziten Parametrisierung verspricht (vgl. Kapitel 4.2.6 sowie Kapitel 5.8.1). Zudem ist mit der Modellierung des Sperrmechanismus eine detaillierte Analyse des Sperrverhaltens möglich. Diese Erweiterungen stellen einen Mehrwert zu der in der Literatur verwendeten impliziten Parametrisierung dar, da die explizite Modellierung die Anzahl der benötigten Messreihen stark reduziert.

In Kapitel 5 wurde Forschungsfrage 3 beantwortet, die sich mit der Durchführung sowie Analyse der Simulation beschäftigt. Bei dem Vergleich zwischen den Mess- und Simulationsergebnissen in Kapitel 5.7 wurde bereits detailliert dargestellt, wie sich das Antwortzeitverhalten der beiden durchgeführten Szenarien erklären lässt. Anschließend wurden die Ursachen für die divergierenden Mess- und Simulationswerte im Hochlastbereich von Szenario 2 analysiert und ein überproportionaler Anstieg der Garbage-Collector-Zeiten festgestellt.

Bei genauerer Betrachtung lassen sich allerdings weitere Schlüsse aus den erzielten Ergebnissen ziehen. Im ersten Szenario, das keine Engpässe in den bereitgestellten Systemressourcen beinhaltet und ein gut konfiguriertes Portalsystem mit ausreichender Tabellenpuffergröße darstellt, lassen sich zwei Bereiche identifizieren (siehe Abbildung 6-1):

- *Bereich 1*: Dieser Bereich stellt den Niedriglastbereich dar, in dem nicht alle Applikations-Threads ausgelastet sind. Es kommt daher nicht zu Wartezeiten beim Verteilen der Benutzeranfragen an die Applikations-Threads. Eine leichte Erhöhung der Antwortzeiten erklärt sich über auftretende Wartezeiten bei gesperrten Objekten.

- *Bereich 2a*: Bei mehr als 80 simultanen Benutzern wird die maximale Parallelität (80 konfigurierte Applikations-Threads) überschritten. Die nun auftretenden Wartezeiten bedingen steigende Antwortzeiten.

Ähnlich verhält es sich im zweiten Szenario, in dem drei Bereiche unterschieden werden können:

- *Bereich 1*: Dieser Bereich stellt auch hier den Niedriglastbereich dar, in dem die Systemressourcen noch nicht ausgeschöpft sind. Die ansteigenden Antwortzeiten erklären sich ebenfalls über auftretende Sperrzeiten, aber auch über zunehmende Verdrängungen im unzureichend dimensionierten Tabellenpuffer.

- *Bereich 2b*: Der zweite Bereich in Szenario 2 wird ebenfalls durch auftretende Wartezeiten charakterisiert, die allerdings nicht durch eine konfigurierte Limitation der Applikations-Threads, sondern durch die erschöpften CPU-Ressourcen bedingt sind. Wie bereits in Kapitel 5.7 dargestellt, führen beide Schranken zu Wartezeiten, die die Bearbeitung der Benutzeranfragen verzögern und folglich zu einem Anstieg der Antwortzeiten führen.

- *Bereich 3*: Der Überlastbereich im zweiten Szenario ist vor allem durch den wachsenden Einfluss des Speichermanagements gekennzeichnet, der in Kapitel 5.8 analysiert wurde. Da der überproportionale Anstieg nicht vom LQN-Modell erfasst wird, divergieren die gemessenen und simulierten Antwortzeiten.

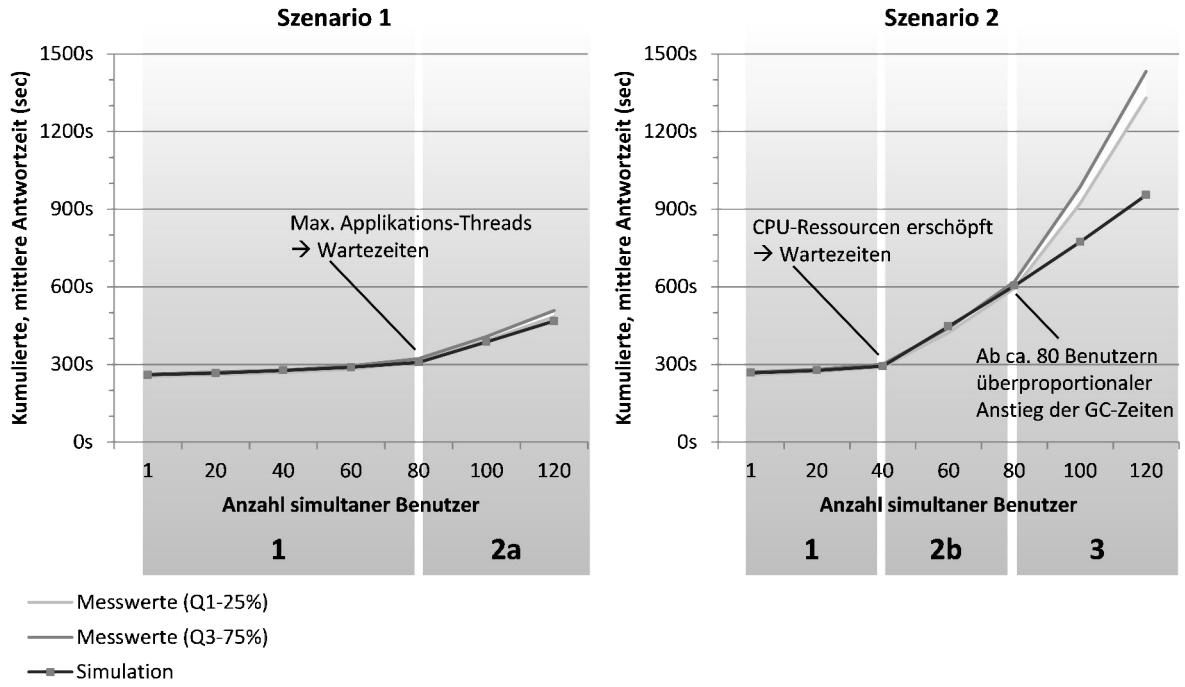


Abbildung 6-1: Bewertung der Ergebnisse nach dem Antwortzeitverhalten
Quelle: eigene Darstellung

Beurteilt man die identifizierten Bereiche nach den erreichten Simulationsergebnissen, können vor allem in den Bereichen 1 und 2a/2b vielversprechende Prognosen festgestellt werden. Das Auftreten von Wartezeiten, das durch eine Begrenzung der Applikations-Threads oder durch erschöpfte CPU-Ressourcen bedingt sein kann, wird von der Simulation zuverlässig erkannt. Ebenso werden die Wartezeiten im Sperrmanagement sowie zunehmende Datenbankzugriffe aufgrund von Verdrängungen und Invalidierungen im Tabellenpuffer simuliert. Lediglich den im dritten Bereich zunehmenden externen Einflüssen (Speicherverwaltung, System-Overhead) wird nicht ausreichend Rechnung getragen. Die in diesem Zusammenhang geltenden Limitationen bei der LQN-Modellierung wurden bereits in den Kapiteln 4.2.6 sowie 5.8 dargestellt.

Zur weiteren Bewertung soll eine Betrachtung der einzelnen Komponenten bzw. Einflussfaktoren erfolgen, die in Kapitel 4.1.2 identifiziert wurden. Eine grundsätzliche Einschätzung der Umsetzung sowie der erzielten Ergebnisse wird in Tabelle 6-1 vorgenommen, wobei die Tendenz anhand der Pfeilrichtung veranschaulicht werden soll.

Komponente	Einflussfaktor	Modellierungsart	Bewertung
Client	Anzahl, Denkzeit	Parameter	↑ Referenz-Komponente, die die Lastschritte aufruft. Einflussfaktoren sind parametrisierbar.
Lastschritt	Dispatching-Zeit und Wartezeiten	Implizit	↑ Dispatching- und Wartezeiten werden implizit über die Warteschlangen des Modells erfasst.
	Bearbeitungszeit	Parameter	↑ Angabe erfolgt über den Mittelwert und die Streuung.
	Abbrüche	Parameter (max_service_time)	↗ Gut geeignet zur Überprüfung von SLAs.
	Datenbankaufrufe (ggf.)	(Synch.) Aufrufe, Datenbanktask, Tabellenpuffer, Sperrmanagement	↑ Gut abbildbar, inkl. Sperrmanagement und Pufferung. Allerdings aufwändig bei der Modellierung.
Datenbank (Black-Box)	Bearbeitungszeit	Parameter	→ Die Datenbank wird lediglich als Blackbox betrachtet.
Tabellenpuffer	Verdrängungen, Invalidierungen	Weiterleitungswahrscheinlichkeit	↗ Die Berechnung der Trefferquote kann nur geschätzt werden.
Sperrmanagement	Wartezeit beim Enqueue-Server und blockierten Elementen.	Implizit	↑ Modelliertes Lese-/Schreibsperrenkonzept bildet das Verhalten der Sperrobjekte nach.
Garbage-Collector (GC)	GC-Dauer GC-Intervall	Zusätzliche GC-Komponente für eine statische Parametrisierung	↘ Das dynamische GC-Verhalten ist nicht exakt abbildbar. Die GC-Komponenten ermöglichen allerdings eine statische Parametrisierung.
CPU (System-Ressource)	System-Overhead	Ressource (Prozessor-Task)	↗ Die gemessene CPU-Nutzung folgt den Annahmen der Warteschlangentheorie bis in den Hochlastbereich, dann allerdings nimmt der zunehmende System-Overhead Einfluss auf die CPU-Nutzung der Benutzer-Prozesse.

Tabelle 6-1: Bewertung der Ergebnisse nach Modellkomponenten und Einflussfaktoren
Quelle: eigene Darstellung

Die Benutzer-Komponente (siehe Kapitel 4.2.1) stellt den Referenz-Task dar und initiiert die Aufrufe der Lastschritte. Eventuelle Ressourcenengpässe auf der Präsentationsebene wurden in der vorgestellten Arbeit nicht behandelt, können aber durch eine entsprechende Modellierung und Parametrisierung der Benutzer-CPU eingeführt werden. Zudem ist es möglich, verschiedene Benutzertypen sowie deren Anzahl und optionale Denkzeiten zu definieren. Die identifizierten Einflussfaktoren konnten somit ohne bekannte Einschränkungen umgesetzt werden.

Die Modellierung der Lastschritte umfasst verschiedene Elemente, die sich auf die Antwortzeit auswirken. Die erzielten Ergebnisse lassen auf eine gute bis sehr gute Abbildbarkeit der Lastschritte schließen. Die Angabe einer maximalen Service-Zeit führt zwar nicht zum Abbruch der Aktion während der Simulation, es wird allerdings in der Ausgabedatei des Simulators die Wahrscheinlichkeit für eine Überschreitung des Limits angegeben. Dies ermöglicht beispielsweise die Überprüfung, ob bestimmte SLAs eingehalten werden.

Die Datenbank wurde in dieser Arbeit als Black-Box betrachtet. Eine Bewertung der Simulationsgenauigkeit kann deswegen nur dahingehend erfolgen, dass bei akkurat parametrisierten Bearbeitungszeiten der Zugriff auf die Persistenzschicht entsprechend nachgezeichnet wird. Das mit den Datenbankzugriffen zusammenhängende Sperrmanagement konnte über ein neuartiges Modellierungskonzept der Lese- und Schreibsperren vielversprechend abgebildet werden. Die Abbildung des Tabellenpuffers wurde über die Angabe einer Weiterleitungswahrscheinlichkeit realisiert, die sich mit dem in Kapitel 4.2.4 dargestellten Ansatz berechnen lässt. Die Genauigkeit dieser Angabe ist sehr stark von vorhandenen Messdaten und der Tabellenpuffergröße in Relation zu der Größe der gepufferten Objekte abhängig und stellt folglich lediglich eine Schätzung der tatsächlichen Trefferrate dar. Die Bewertung der Simulationsgenauigkeit des Tabellenpuffers ist daher nicht eindeutig möglich. Allerdings wird in der Literatur erst eine Trefferrate von 95 Prozent oder höher als befriedigend (Heiss/Veirich/Gratzl 2005, 355) angesehen, sodass bei einem gut konfigurierten Portalsystem eine akkurate Parametrisierung möglich ist, da die Weiterleitungswahrscheinlichkeit sehr gering gehalten werden kann.

Die bei der Analyse des Garbage-Collectors in Kapitel 5.8 sowie bei der Modellierung in Kapitel 4.2.6 eingeführte GC-Komponente ermöglicht eine explizite Parametrisierung der Aktivitätsintensität. Dies war erforderlich, da der implizite Anteil der GC-Zeit in der Bearbeitungszeit aufgrund des überproportionalen Anstiegs der Speicherverwaltung im Hochlastbereich nicht mehr ausreicht. Wie bereits erläutert, ermöglicht der Modellierungsansatz zwar eine Untersuchung der Auswirkungen des Garbage-Collectors, bildet jedoch nicht das Verhalten des Garbage-Collectors dynamisch ab. Diese Einschränkung ist vor allem durch die begrenzten Mittel der LQN-Modellierung sowie die fehlenden Detailkenntnisse über die GC-Implementierung bedingt. Wie bereits dargestellt, sind in der Literatur keine Konzepte zur expliziten Modellierung des Garbage-Collectors mittels LQN präsentiert worden. Die GC-Zeiten waren stets implizit in den Bearbeitungszeiten enthalten. Die vorgestellte Möglichkeit, die Aktivitätsintensität explizit zu parametrisieren stellt somit ein neuartiges Konzept bei der Analyse des GC-Einflusses dar. Da die Aktivitätsintensität jedoch lediglich ex-ante festgelegt werden kann, bietet es sich an, die Modellierung der Java-Speicherverwaltung zum Gegenstand weiterer Untersuchungen zu machen.

Die CPU-Ressourcen werden in dem LQN-Modell als spezieller Task-Typ, als Prozessor-Task, modelliert. Die Parametrisierung erfolgt über die Multiplizität und somit die Anzahl an simultan berechneten Lastschritten ohne Wartezeiten am Prozessor. Unter der aus der Warteschlangentheorie abgeleiteten Annahme einer linear ansteigenden CPU-Nutzung bis zum Erreichen der Vollauslastung (vgl. Kapitel 4.1.1), werden die Rechenkapazitäten vom LQN-Modell akkurat nachgebildet. Die Ergebnisse haben allerdings gezeigt, dass die CPU aufgrund des System-Overheads und des Speichermanagements von den Benutzerprozessen nicht voll ausgelastet werden kann. Dieser Sachverhalt muss über die Multiplizität des Prozessor-Tasks entsprechend parametrisiert werden, bildet aber nicht den wachsenden Anteil der Systemprozessorzeit im Überlastbereich ab und führt folglich zu divergierenden Ergebnissen zwischen gemessenen und simulierten Antwortzeiten. Die Abbildung der CPU-Ressource im LQN-Modell kann im Normallastbereich als sehr gut bewertet werden, im Hochlastbereich wird den äußeren Einflüssen jedoch nicht ausreichend Rechnung getragen.

Führt man die Bewertung der Ergebnisse nach dem Antwortzeitverhalten und nach den modellierten Komponenten zusammen, lässt sich die in Abbildung 6-2 dargestellte Simulationsgenauigkeit nach der Auslastung des Systems bewerten.

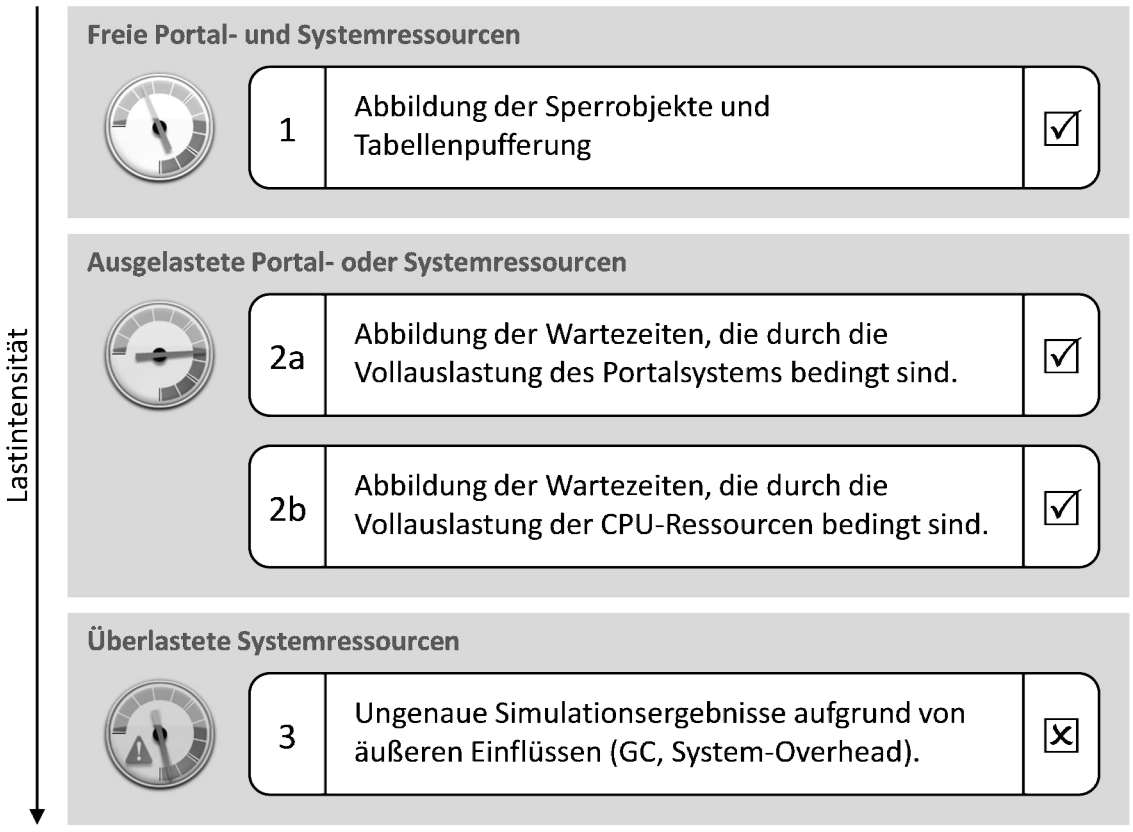


Abbildung 6-2: Bewertung der Ergebnisse nach der Lastintensität
Quelle: eigene Darstellung

Die in Abbildung 6-2 dargestellten Situationen entsprechen den in Abbildung 6-1 identifizierten Lastbereichen. Im ersten Bereich sind die Portal- und Systemressourcen nicht voll ausgelastet. Der leichte Anstieg der Antwortzeiten lässt sich über die Sperrwartezeiten und eventuellen Verdrängungen bzw. Invalidierungen im Tabellenpuffer erklären. Dies wird von dem Simulationsmodell entsprechend abgebildet. Der zweite Bereich stellt eine Vollauslastung der

Applikations-Threads (auf Portalsystem-Seite) bzw. der CPU-Ressourcen (auf Seiten des Hosts) dar, je nachdem, welche Grenze zuerst erreicht wird. Beide Limitationen rufen Wartezeiten hervor, die von dem Simulationsmodell ebenfalls erfasst werden. Der dritte Bereich stellt schließlich den Überlastbereich dar, bei dem sich hohe GC-Zeiten sowie ein allgemeiner System-Overhead auf die Antwortzeiten auswirken. Diese äußeren Einflüsse werden vom Simulationsmodell nicht hinreichend genau erfasst.

Flaschenhalsanalyse

Die Bewertung der Simulationsergebnisse kann ebenfalls der Analyse von potentiellen Flaschenhälsen dienen. Aus Sicht der LQN-Modellierung stellt ein voll ausgelasteter Task einer übergeordneten Ebene, dessen genutzte Ressourcen untergeordneter Ebenen nicht voll ausgelastet sind, einen Flaschenhals dar (vgl. Neilson et al. 1995).

Der Ausgabedatei des Simulators können Informationen zur Auslastung der einzelnen Entries und Aktivitäten entnommen werden. Der wiedergegebene Wert der Nutzung liegt im Bereich $0 \leq Nutzung \leq m,$ wobei m die Multiplizität der Komponente spezifiziert und somit die durchschnittliche Anzahl an verwendeten Threads darstellt. Zeigt zum Beispiel der Datenbank-Task eine sehr hohe Nutzung der Threads bei einer durchschnittlichen Systemlast, kann eine Erhöhung der Datenbankverbindungen den Flaschenhals beseitigen. Ebenso können hohe Sperrzeiten einzelne Applikations-Threads blockieren. Sollten noch Systemressourcen vorhanden sein, jedoch keine freien Applikations-Threads mehr zur Verfügung stehen, liegen diese brach.

Damit ein potentieller Flaschenhals bei der konfigurierten Anzahl an Applikations-Threads verhindert werden kann, muss die Anzahl für das zugrundeliegende System optimiert werden. In Szenario 1 konnten, wenngleich bewusst durchgeführt, die Auswirkungen einer zu früh greifenden Einschränkung der zur Verfügung stehenden Applikations-Threads verdeutlicht werden. Obwohl noch genügend Systemressourcen vorhanden waren, wurde eine höhere Parallelität durch die konfigurierte Grenze verhindert. Auf der anderen Seite darf jedoch die Menge an zur Verfügung stehenden Applikations-Threads trotz vorhandener CPU-Ressourcen nicht zu hoch angesetzt werden, da eine höhere Anzahl an simultan abgearbeiteten Benutzeranfragen einen höheren Speicherverbrauch mit sich bringt und zu erhöhten GC-Aktivitäten führen kann, falls der Heap-Speicher fast zur Gänze belegt wird.

Kapazitätsplanung

Einen weiteren Blickwinkel auf die Simulationsergebnisse stellt die Betrachtung hinsichtlich der Kapazitätsplanung und dem damit einhergehenden Bemessen der Systeme („Sizing“) dar. Der in Kapitel 2.7 vorgestellte Kapazitätsplanungsprozess nach Menascé/Almeida (2002, 179) sieht eine Charakterisierung des Workloads und die Erstellung eines Performance-Modells vor, das zur Performance-Vorhersage genutzt wird. Dies kann mit dem in dieser Arbeit vorgestellten Modellierungs- und Simulationsvorgehen bewerkstelligt werden. Daneben muss ein Kostenmodell zur Kostenvorhersage erstellt werden, das im Zusammenspiel mit der Performance-Vorhersage zur Kosten-Performance-Analyse führt. Daraus kann wiederum ein Konfigurationsplan entwickelt werden, der notwendige Hardware- und Softwarekonfigurationen spezifiziert.

6.2 Limitationen

Die Performance-Analyse von Unternehmenssoftware kann sowohl aus betriebswirtschaftlicher Sicht, als auch in Hinblick auf technische Leistungskennzahlen erfolgen. Die in dieser Arbeit durchgeführte Performance-Modellierung und Simulation eines SAP-Netweaver-Portal-Systems beschränkt sich auf die technische Leistungsbewertung. Dazu wurde die Antwortzeit als betrachtete Metrik spezifiziert und das Leistungsverhalten bei gleichbleibendem Workload sowie einer steigenden Anzahl von Benutzern beobachtet.

Dem vorgestellten Ansatz zur Performance-Modellierung eines SAP-Netweaver-Portal-Systems liegt die Annahme zugrunde, dass durchgeführte Portaloperationen über fest definierte Key-Performance-Indikatoren beschrieben werden können. Diese wurden in Kapitel 4.1.2 identifiziert, wie zum Beispiel durchschnittliche Bearbeitungszeiten, Datenbankzugriffe, gepufferte Tabelleninhalte oder erzeugte Sperrobjekte.

Die Modellierung umfasst sowohl (logische) Software-Komponenten des Portalsystems, als auch Prozessor-Ressourcen, die mit den modellierten Software-Komponenten verbunden sind. Die Abbildung der Präsentationsebene dient lediglich dem Erzeugen von Benutzeranfragen. Ebenso wurde davon ausgegangen, dass das Datenbanksystem auf einem eigenen Host installiert ist und über genügend Systemressourcen verfügt. Die Abarbeitung der Datenbankanfragen wurde als Black-Box modelliert, sodass interne Abläufe des Datenbanksystems von dem Performance-Modell nicht erfasst werden.

Es wurde zudem vorausgesetzt, dass genügend Hauptspeicher für die Allokation des Heap-Speichers zur Verfügung steht. Diese Annahme schließt den Abbruch der Programmausführung bei unzureichendem Speicher aus. Die Freisetzung von nicht mehr benötigtem Speicherinhalt ist Aufgabe des Garbage-Collectors, dessen Aktivitäten sich ebenfalls auf die Performance des Portalsystems auswirken. Eine detaillierte Modellierung des GC-Verhaltens ist, wenn überhaupt möglich, nicht nur mühsam, sondern auch bei der Parametrisierung schwierig umzusetzen, da die GC-Zeiten in den aufgezeichneten Bearbeitungszeiten der Performance-Traces enthalten sind und somit ausgeklammert werden müssten. Der in dieser Arbeit vorgestellte Modellierungsansatz zur Angabe der Aktivitätsintensität des Garbage-Collectors dient lediglich speziellen Analysezwecken.

Der Einfluss des Betriebssystems (System-Overhead) und die Defizite in der Abbildung des Speichermanagements führen im Überlastbereich zu ungenauen Simulationsergebnissen. Die Einsatzmöglichkeiten des Performance-Modells in diesem Lastbereich sind somit sehr begrenzt. Dies erklärt sich durch die fehlende Modellierung des zugrundeliegenden Systems (Hardware, Betriebssystem).

Neben der rein technischen Betrachtung des Leistungsverhaltens eines SAP-Netweaver-Portal-Systems kann der vorgestellte Ansatz zur Performance-Modellierung und Performance-Vorhersage mittels Simulation auch im Kontext der Kapazitätsplanung herangezogen werden. Allerdings beschränkt sich die Aussagekraft hauptsächlich auf die Prognose der Skalierbarkeit eines vorhandenen Systems, da neue Hardwarekomponenten und eine Vielzahl an Konfigurationsmöglichkeiten erst vermessen werden müssten.

6.3 Ausblick

Das in dieser Arbeit entwickelte Artefakt bzw. die bei der Evaluation erzielten Simulationsergebnisse sind abgesehen vom Überlastbereich als positiv zu bewerten. In Hinblick auf zukünftige Forschungstätigkeiten können verschiedene Modellerweiterungen angestrebt werden, die sich aus den im vorangegangenen Abschnitt dargestellten Limitationen ableiten.

Das Datenbanksystem wurde in dieser Arbeit als Black-Box betrachtet. Da jedoch die Performance der Persistenzschicht einen wesentlichen Einfluss auf die Gesamtleistung des Systems nimmt, kann eine feingranulare Modellierung des Datenbanksystems besseren Aufschluss über potentielle Leistungsengpässe im I/O-Bereich geben. Die heterogenen Datenbankarchitekturen verschiedener Hersteller erfordern jedoch unterschiedliche Submodelle, sodass auch hier nur ein herstellerabhängiger Ansatz gewählt werden kann. Eine weitere Möglichkeit zur Parametrisierung der Datenbank als Black-Box stellen multidimensionale Regressionsverfahren dar, die beispielsweise über einen evolutionären Algorithmus realisiert werden können (Tertilt/Krcmar 2011; Tertilt et al. 2010).

Der in dieser Arbeit entwickelte Modellierungsansatz für Lese- und Schreibsperren war notwendig, da in der LQN-Modellierung kein Element für einen derartigen Mechanismus vorgesehen ist (vgl. Franks 2011). Damit die in dieser Arbeit eingeführte Modellierung mittels Aktivitäten und Semaphore vereinfacht werden kann, könnte der Semaphor-Task um eine entsprechende Funktion erweitert werden.

Zudem bildet die tiefergehende Untersuchung der GC-Modellierung eine weitere Forschungsmöglichkeit. Wie sich aus der durchgeführten Analyse des GC-Einflusses schließen lässt, würde eine detailliertere Abbildung des GC-Verhaltens zu akkurateren Simulationsergebnissen im Hoch- und Überlastbereich führen. Ebenso stellt die Berechnung der Weiterleitungswahrscheinlichkeit des Tabellenpuffers lediglich eine Annäherung der tatsächlichen Trefferquote dar. Eine weitergehende Forschungsarbeit könnte in diesem Zusammenhang untersuchen, ob eine Abbildung des Puffermechanismus nach dem Least-Recently-Used-Prinzip mit LQN-Mitteln durchführbar ist.

Im Bereich der Modellgenerierung wäre die Entwicklung einer (grafischen) Anwendung hilfreich, die aus den aufgezeichneten Performancedaten ein erstelltes Modell parametrisieren und die Eingabedatei des Simulators ausgeben bzw. den Simulator damit starten kann. In Kombination mit der Ausführung der Simulation und der Auswertung der Ergebnisse könnte so ein Rahmenwerk geschaffen werden, das die effiziente Analyse und Optimierung eines modellierten Systems ermöglicht.

Nicht zuletzt stellt die Modellierung externer Funktionen, die innerhalb des Portals ausgeführt werden, eine weitere Forschungsmöglichkeit dar. Eine häufig genutzte, externe Portalfunktionalität ist beispielsweise die Integration von klassischen ERP-Funktionen. Mit den Ergebnissen der derzeit am Lehrstuhl für Wirtschaftsinformatik der TU München durchgeführten Arbeit zur Performance-Modellierung und Simulation von ERP-Systemen und einem vereinheitlichten Modell könnten so im Portal integrierte ERP-Funktionen, die über den ABAP-Applikationsserver ausgeführt werden, modelliert werden.

Die bis dato dargestellten Forschungsmöglichkeiten beziehen sich auf die Weiterentwicklung des erstellten Artefakts. Weitere Untersuchungen können zudem im Bereich der Evaluation des Performance-Modells erfolgen. Die in dieser Arbeit durchgeführte Demonstration und Evaluation basiert auf einem für eine Schulungsumgebung typischen Workload. Ergebnisse zu anderen Lastmustern würden zusätzlichen Aufschluss über die Aussagekraft des in dieser Arbeit entwickelten Performance-Modells geben und dessen Einsatzmöglichkeiten spezifizieren.

Literaturverzeichnis

Alisch, K. (2004): Gabler-Wirtschaftslexikon. [die ganze Welt der Wirtschaft: Betriebswirtschaft, Volkswirtschaft, Recht, Steuern]. 16. vollst. überarb. u. aktual. Aufl., Gabler, Wiesbaden 2004.

Alpar, P.; Grob, H.L.; Weimann, P.; Winter, R. (2008): Anwendungsorientierte Wirtschaftsinformatik. 5. überarb. u. aktual. Aufl., Vieweg & Teubner, Wiesbaden 2008.

Amberg, M.; Remus, U.; Holzner, J. (2003): Portal-Engineering - Anforderungen an die Entwicklung komplexer Unternehmensportale. In: Wirtschaftsinformatik 2003, Band II - Medien, Märkte, Mobilität (Vol. 2). Hrsg.: Uhr, W.; Esswein, W.; Schoop, E. Physica-Verlag, Springer, Heidelberg 2003, S. 795-817.

Anderson, P.; Arrow, K.; Pines, D. (1988): The Economy as an Evolving Complex System. Addison-Wesley, Redwood City 1988.

Anonymous; Bitton, D.; Brown, M.; Catell, R.; Ceri, S.; Chou, T.; DeWitt, D.; Gawlick, D.; Garcia-Molina, H.; Good, B.; Gray, J.; Homan, P.; Jolls, B.; Lukes, T.; Lazowska, E.; Nauman, J.; Pong, M.; Spector, A.; Trieber, K.; Sammer, H.; Serlin, O.; Stonebraker, M.; Reuter, A.; Weinberger, P. (1985): A measure of transaction processing power. In: Datamation, Vol. 31 (1985) Nr. 7, S. 112-118.

Arnold, D.; Isermann, H.; Kuhn, A. (2004): Handbuch Logistik. 2. aktual. u. korr. Aufl., Springer, Berlin 2004.

Baier, C.; Haverkort, B.R.; Hermanns, H.; Katoen, J.-P. (2003): Model-Checking Algorithms for Continuous-Time Markov Chains. In: IEEE Transactions on Software Engineering, Vol. 29 (2003) Nr. 7, S. 1-18.

Banks, J.; Carson, J.; Nelson, B.L.; Nicol, D. (2004): Discrete-Event System Simulation. Prentice-Hall, Englewood Cliffs, NJ, USA 2004.

Barham, P.; Dragovic, B.; Fraser, K.; Hand, S.; Harris, T.; Ho, A.; Neugebauery, R.; Pratt, I.; Warfield, A. (2003): Xen and the Art of Virtualization. In: 19th ACM Symposium on Operationg Systems Principles Hrsg., Bolton Landing, NY 2003, S. 164-177.

Baumgarten, U.; Siegert, H.-J. (2007): Betriebssysteme: Eine Einführung. München, Oldenbourg Wissenschaftsverlag 2007.

Bause, F. (1993): Queueing Petri Nets: A Formalism for the Combined Qualitative and Quantitative Analysis of Systems. In: 5th International Workshop on Petri Nets and Performance Models (IEEE) Hrsg. IEEE Computer Society, Toulouse, Frankreich 1993, S. 14-23.

Bause, F.; Beilner, H. (1989): Eine Modellwelt zur Integration von Warteschlangen- und Petri-Netz-Modellen. In: 5. GI/ITG-Fachtagung "Messung, Modellierung und Bewertung von Rechensystemen und Netzen" Hrsg., Braunschweig 1989, S. 190-204.

Berman, S. (2008): Streuung und Streuungsmaße. In: http://homepage.ruhr-uni-bochum.de/stephen.berman/Statistik/Streuung.html, zugegriffen am 19.10.2011.2011.

Bichler, M. (2006): Design science in information systems research. In: Wirtschaftsinformatik, Vol. 48 (2006) Nr. 2, S. 133-135.

Blättel, B.; Koch, V.; Mosel, U. (1993): Transport-Theoretical Analysis of Relativistic Heavy-Ion-Collisions. In: Reports on Progress in Physics, Vol. 56 (1993) Nr. 1.

Boegelsack, A. (2012): Performance und Skalierung von SAP ERP Systemen in virtualisierten Umgebungen, Technische Universität München, Springer Gabler 2012.

Boegelsack, A.; Wittges, H.; Krcmar, H. (2010): Scalability and Performance of a Virtualized SAP System. In: Proceedings of the 16th American Conference on Information Systems Hrsg., Lima, Peru 2010, S. Paper 13.

Boegelsack, A.; Wittges, H.; Krcmar, H.; Kuehnemund, H. (2011): Zachmanntest - A Synthetic Memory Benchmark for SAP ERP Systems. In Zhang, R.; Cordeiro, J.; Li, X.; Zhang, Z.; Zhang, J. (Eds.), *ICEIS (1)* (S. 348-355): SciTePress.

Bolch, G.; Greiner, S.; de Meer, H.; Trivedi, K.S. (2006): Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications. John Wiley and Sons, Hoboken, NJ, USA 2006.

Bond, B.; Genovese, Y.; Miklovic, D.; Wood, N.B. (2000): ERP is dead - Long live ERP II. In: Strategic Planning SPA, Vol. 12 (2000), S. 12-15.

Bönnen, C.; Herger, M. (2007): SAP Netweaver Visual Composer. Galileo Press, Bonn, Boston 2007.

Bortz, J.; Döring, N. (1995): Forschungsmethoden und Evaluation. Für Sozialwissenschaftler. 2. vollst. überarb. Aufl., Springer-Verlag GmbH, Berlin, Heidelberg 1995.

Buchholz, P. (1992): A Hierarchical View of GCSPNs and its Impact on Qualitative and Quantitative Analysis. In: Journal of Parallel and Distributed Computing, Vol. 15 (1992) Nr. 3, S. 207-224.

Chamberlin, D.D.; Boyce, R.F. (1974): SEQUEL: A structured English Query Language. In: ACM SIGFIDET (jetzt SIGMOD) Workshop on Data Description, Access and Control Hrsg., Ann Harbor, Michigan 1974, S. 249-264.

Cheng, X. (2008): Performance, Benchmarking and Sizing in Developing Highly Scalable Enterprise Software. *Proceedings of the SPEC international workshop on Performance Evaluation: Metrics, Models and Benchmarks* (S. 174-190). Darmstadt, Germany: Springer Verlag.

Cheng, X.; Morrison, T. (2007): Best Practices for Java Performance and Load Tests. In SAP (Ed.), *Teched 2006*. Las Vegas.

Davis, C.; Orb, W.; Koechl, M.; Calio, C. (2006): SAP Enterprise Portal on AIX 5.3 and POWER5 - IBM SAP Technical White Paper, Walldorf 2006.

Dijkstra, E.W. (1971): Hierarchical Ordering of Sequential Processes. In: Acta Informatica, Vol. 1 (1971) Nr. 2, S. 115-138.

Doherty, W.J. (1970): Scheduling TSS/360 for Responsiveness. In: AFIPS 1970 FJCC (Vol. 37), Hrsg. AFIPS Press, Montvale, N.J. 1970, S. 97-111.

Domschke, W.; Drexl, A. (2007): Einführung in Operations Research. 7. überarb. Aufl., Springer, Berlin 2007.

Dongarra, J.J.; Moler, C.B.; Bunch, J.R.; Steward, G.W. (1979): LINPACK User's Guide. Society for Industrial and Applied Mathematics 1979.

Eichorn, J. (2006): Understanding AJAX: Using JavaScript to Create Rich Internet Applications. Prentice Hall PTR, Upper Saddle River, NJ 2006.

Eilert, J.; Eisenhaendler, M.; Matthaeus, D.; Salm, I. (2003): Linux on the Mainframe. Prentice Hall PTR, Englewood Cliffs, NJ 2003.

Erlang, A.K. (1909): The Theory of Probabilities and Telephone Conversations. In: Nyt Tidsskrift for Matematik, Vol. 20 (1909) Nr. B, S. 33-39.

Feldt, K.C. (2007): Programming Firefox. O'Reilly, Beijing 2007.

Ferrari, D. (1986): Considerations on the insularity of performance evaluation. In: IEEE Transactions on Software Engineering, Vol. 12 (1986), S. 678-683.

Ferrari, D. (1978): Computer Systems Performannce Evaluation. Prentice-Hall 1978.

Ferrari, D.; Serazzi, G.; Zeigner, A. (1983): Measurement and Tuning of Computer Systems. Prentice-Hall, Englewood Cliffs, NJ, USA 1983.

Ferschl, F. (1970): Markovketten. Springer-Verlag, Heidelberg 1970.

Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Masinter, L.; Leach, P.; Berners-Lee, T. (1999): Hypertexst Transfer Protocol -- HTTP/1.1. In: http://www.ietf.org/rfc/rfc2616.txt, zugegriffen am 10.12.2011.2011.

Fink, A.; Schneidereit, G.; Voß, S. (2005): Grundlagen der Wirtschaftsinformatik. 2. überarb. Aufl., Physica-Verlag, Heidelberg 2005.

Finkelstein, L. (1984): A Review of the Fundamental Concepts of Measurement. In: Measurement, Vol. 2 (1984) Nr. 1, S. 25-34.

Franks, G. (2011): Simulating layered queueing networks with passive resources. *Proceedings of the 2011 Theory of Modeling & Simulation Symposium: DEVS Integrative M&S Symposium* (S. 8-15). Boston, Massachusetts: Society for Computer Simulation International.

Franks, G. (1999): Performance Analysis of Distributed Server Systems, Carleton University 1999.

Franks, G.; Lau, D.; Hrischuk, C. (2011): Performance measurements and modeling of a java-based session initiation protocol (SIP) application server. *Proceedings of the joint ACM SIGSOFT conference -- QoSA and ACM SIGSOFT symposium -- ISARCS on Quality of software architectures -- QoSA and architecting critical systems -- ISARCS* (S. 63-72). Boulder, Colorado, USA: ACM.

Franks, G.; Maly, P.; MWoodside, M.; Petriu, D.; Hubbard, A. (2012): Layered Queueing Network Solver and Simulator User Manual. Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada 2012.

Gadatsch, A. (2002): Management von Geschäftsprozessen. 2. überarb. u. erw. Aufl., Vieweg Verlag, Braunschweig, Wiesbaden 2002.

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. (1995): Design Patterns: Elements of Reusable Object-Oriented Software. 1. Aufl., Addison-Wesley Longman Publishing Co., Inc. , Amsterdam 1995.

Gibson, J.C. (1970): The Gibson Mix (TR 00.2043). IBM Systems Development Division, Poughkeepsie, NY, USA 1970.

Gomaa, D.A.; Menascé, D.A.; Kerschberg, L. (1996): A Software Architectural Design Method for Large-Scale Distributed Information Systems. In: Distributed Systems Engineering, Vol. 3 (1996) Nr. 3, S. 162-172.

Gootzit, D. (2008): Key Issues for Enterprise Portals. In: Gartner Research, (2008).

Gordon, W.J.; Newell, G.F. (1967): Closed Queueing Systems with Exponential Servers. In: Operations Research, Vol. 15 (1967) Nr. 2, S. 254-265.

Görtz, M.; Hesseler, M. (2008): Basiswissen ERP-Systeme. Auswahl, Einführung & Einsatz betriebswirtschaftlicher Standardsoftware. 1. Aufl., W3L-Verlag, Herdecke, Witten 2008.

Gradl, S. (2012): Performance-Modellierung und Simulation eines SAP-ERP-Systems, Technische Universität München 2012.

Graham, R. (1975): Performance prediction. In: Software Engineering. Lecture Notes in Computer Science (Vol. 30). Hrsg.: Bauer, F.; Dennis, J.; Waite, W.; Gotlieb, C.;

Graham, R.; Griffiths, M.; Helms, H.; Morton, B.; Poole, P.; Tsichritzis, D. Springer Berlin / Heidelberg, 1975, S. 395-463.

Gray, K. (1991): The Benchmark Handbook for Database and Transaction Processing Systems. Morgan Kaufmann, San Francisco 1991.

Großmann, M.; Koschek, H. (2005): Unternehmensportale - Grundlagen, Architekturen, Technologien. Springer Verlag, Berlin/Heidelberg 2005.

Haas, M.; Zorn, W. (1995): Methodische Leistungsanalyse von Rechensystemen. Oldenbourg, München, Wien 1995.

Haerder, T.; Reuter, A. (1983): Principles of transaction-oriented database recovery. In: ACM Comput. Surv., Vol. 15 (1983) Nr. 4, S. 287-317.

Harris, N.; Jeyakumar, L.R.; Mann, S.; Sumarga, Y.; Wei, W. (2005): Logical Partitions on the IBM PowerPC. IBM 2005.

Hartmann, S. (1996): The World as a Process: Simulations in the Natural and Social Sciences. In: Modelling and Simulation in the Social Sciences from the Philosophy of Science Point of View. Hrsg.: Hegselmann, R. 1996.

Heilig, L.; Karch, S. (2007): SAP NetWeaver Master Data Management. Galileo Press, Bonn, Boston 2007.

Heinrich, L.J.; Lehner, F. (2005): Informationsmanagement. Planung, Überwachung und Steuerung der Informationsinfrastruktur. 8. vollst. überarb. u. erg. Aufl., Oldenbourg, München 2005.

Heiss, F.J.; Veirich, E.; Gratzl, e. (2005): SAP NetWeaver Web Application Server. Pearson Deutschland GmbH, München 2005.

Henning, J.L. (2006): SPEC CPU2006 Benchmark Descriptions. In: SIGARCH Comput. Archit. News, Vol. 34 (2006) Nr. 4, S. 1-17.

Hermann, C.; Hermann, M. (1972): An Attempt to Simulate the Outbreak of World War I. In: Simulations in Social and Administrative Science: Overview and Case-Examples. Hrsg.: Guetzkow, H.; Kotler, P.; Schultz, R. Prentice-Hall, Englewood Cliffs, NJ 1972.

Hertz, H. (1894): Die Prinzipien der Mechanik. Aus: "Kollektion Naturwissenschaften und Technik". Thüringer Universitäts- und Landesbibliothek Jena, Jena 1894.

Hevner, A.R. (2007): A Three Cycle View of Design Science Research. In: Scandinavian Journal of Information Systems, Vol. 19 (2007) Nr. 2, S. 87-92.

Hevner, A.R.; March, S.T.; Park, J.; Ram, S. (2004): Design Science in Information Systems Research. In: MIS Quarterly, Vol. 28 (2004) Nr. 1, S. 75-105.

Hoetzel, A.; Benhaim, A.; Griffiths, N.; Holliday, C. (1998): Benchmarking in Focus. IBM 1998.

Hofmann, R.; Klar, R.; Mohr, B.; Quick, A.; Siegle, M. (1994): Distributed Performance Monitoring: Methods, Tools, and Applications. In: IEEE Trans. Parallel Distrib. Syst., Vol. 5 (1994) Nr. 6, S. 585-598.

Hower, R. (2009): Web Site Test Tools and Site Management Tools - 420 Tools Listed. In: http://www.softwareqatest.com/qatweb1.html, zugegriffen am 28.12.2011.2011.

Hu, L.; Gorton, I. (1997): Performance Evaluation for Parallel Systems: A Survey (9707). University of New South Wales, Sydney, Australia 1997.

IBM (2011a): IBM Active Memory Expansion. In: https://www.ibm.com/developerworks/wikis/display/WikiPtype/IBM+Active+Memory+Expansion, zugegriffen am 10.07.2011.2011.

IBM (2011b): IBM AIX - Unix on Power Systems. In: http://www-03.ibm.com/systems/power/software/aix/index.html, zugegriffen am 11.07.2011.2011.

IBM (2011c): IBM Support Assistant Tool Add-Ons List. In: http://www-01.ibm.com/support/docview.wss?uid=swg27013116#IBM%20Monitoring%20and%20Diagnostic, zugegriffen am 19.09.2011.2011.

IBM (2011d): IBM XIV Storage System. In: http://www-03.ibm.com/systems/de/storage/disk/xiv/, zugegriffen am 07.07.2011.2011.

IBM (2011e): iostat Command. In: http://publib.boulder.ibm.com/infocenter/aix/v6r1/index.jsp?topic=%2Fcom.ibm.aix.cmds%2Fdoc%2Faixcmds5%2Ftopasout.htm, zugegriffen am 08.07.2011.2011.

IBM (2011f): mpstat Command. In: http://publib.boulder.ibm.com/infocenter/aix/v6r1/index.jsp?topic=%2Fcom.ibm.aix.cmds%2Fdoc%2Faixcmds5%2Ftopasout.htm, zugegriffen am 08.07.2011.2011.

IBM (2011g): Pattern Modeling and Analysis Tool for Java Garbage Collector. In: https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=22d56091-3a7b-4497-b36e-634b51838e11, zugegriffen am 03.08.2011.2011.

IBM (2011h): Topas. In: http://publib.boulder.ibm.com/infocenter/aix/v6r1/index.jsp?topic=%2Fcom.ibm.aix.cmds%2Fdoc%2Faixcmds5%2Ftopas.htm, zugegriffen am 08.08.2011.2011.

IBM (2011i): topas CEC Analyser. In: https://www.ibm.com/developerworks/wikis/display/WikiPtype/topas+CEC+Analyser, zugegriffen am 08.07.2011.2011.

IBM (2011j): topasout Command. In: http://publib.boulder.ibm.com/infocenter/aix/v6r1/index.jsp?topic=%2Fcom.ibm.aix.cmds%2Fdoc%2Faixcmds5%2Ftopasout.htm, zugegriffen am 08.07.2011.2011.

IBM (2011k): topasrec Command. In: http://publib.boulder.ibm.com/infocenter/aix/v6r1/index.jsp?topic=%2Fcom.ibm.aix.cmds%2Fdoc%2Faixcmds5%2Ftopasrec.htm, zugegriffen am 07.08.2011.2011.

IBM (2011l): Understanding the IBM JVM. In: http://publib.boulder.ibm.com/infocenter/javasdk/v1r4m2/index.jsp?topic=%2Fcom.ibm.java.doc.diagnostics.142%2Fhtml%2Funderthejvm.html, zugegriffen am 04.12.2011.2011.

IBM (2011m): vmstat Command. In: http://publib.boulder.ibm.com/infocenter/aix/v6r1/index.jsp?topic=%2Fcom.ibm.aix.cmds%2Fdoc%2Faixcmds5%2Ftopasout.htm, zugegriffen am 09.07.2011.2011.

Iivari, J. (2007): A Paradigmatic Analysis of Information Systems as a Design Science. In: Scandinavian Journal of Information Systems, Vol. 19 (2007) Nr. 2, S. 39-64.

iOpus (2012): iOpus iMarcros. In: http://www.iopus.com/de/imacros/, zugegriffen am 12.02.2012.2012.

Jackson, J.R. (1964): Jobshop-Like Queueing Systems. In: Management Science, Vol. 10 (1964), S. 131-142.

Jaenecke, P. (1982): Grundzüge einer Meßtheorie. In: Journal for General Philosophy of Science, Vol. 13 (1982) Nr. 2, S. 234-279.

Jain, R. (1991): The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. Wiley/Interscience, New York, NY, USA 1991.

Janssen, S.; Marquard, U. (2007): Sizing SAP Systems. SAP Press 2007.

Jay, R. (2008): SAP NetWeaver Portal Technology. McGraw-Hill Professional 2008.

Jehle, H. (2010): Performance Measurement of an SAP Enterprise Portal System in a Virtualized Environment, Technische Universität München 2010.

Jianyi, N.; Zhengqiu, Y. (2008): An Automated Test Tool of Web Application Based on Struts. In: IEEE Computer Society, Vol. 1 (2008) Nr. 2008, S. 488-490.

John, L.K.; Eeckhout, L. (2006): Performance Evaluation and Benchmarking. CRC-Press, Boca Raton 2006.

Jonkers, H. (1994): Queueing models of parallel applications: the Glamis methodology. In: Proceedings of the 7th International Conference on Computer Performance Evaluation Hrsg. Springer-Verlag New York, Inc., Secaucus, NJ, USA 1994, S. 123-138.

Joslin, E.O. (1965): Evaluation and performance of computers: application benchmarks: the key to meaningful computer evaluations. *Proceedings of the 1965 20th national conference* (S. 27-37). Cleveland, Ohio, United States: ACM.

Kay, W. (2001): Capacity Planning for SAP- Concepts and tools for performance monitoring and modelling.

Kemper, A.; Eickler, A. (2011): Datenbanksysteme. Eine Einführung., 8. aktual. u. erw. Aufl., Oldenbourg, München 2011.

Kendall, D.G. (1951): Some Problems in the Theory of Queues. In: Royal Statistic Society, Vol. 13B (1951), S. 151-182.

Khintchine, A.Y. (1932): Mathematical Theory of a Stationary Queue. In: Matematicheskii Sbornik, Vol. 39 (1932) Nr. 4, S. 73-84.

Kippenhahn, R.; Weigert, A. (1991): Stellar Structure and Evolution. Springer, Berlin 1991.

Klein, J.M. (2000): Building Enhanced HTML Help with Dhtml and CSS. Prentice Hall PTR, Upper Saddle River, NJ 2000.

Knuth, D.E. (1970): Von Neumann's First Computer Program. In: ACM Computer Surveys, Vol. 2 (1970) Nr. 4, S. 247-260.

Kolonko, M. (2008): Stochastische Simulation. 1. Aufl., Vieweg+Teubner Verlag, Wiesbaden 2008.

Konradin (2011): Einsatz von ERP-Lösungen in der Industrie (Konradin ERP-Studie 2011). Konradin Business GmbH, Leinfelden-Echterdingen 2011.

Kounev, S.; Buchmann, A. (2003): Performance modelling of distributed e-business applications using Queuing Petri Nets. In: Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software Hrsg. IEEE Computer Society, Washington, DC, USA 2003, S. 143-155.

Kounev, S.; Weis, B.; Buchmann, A.P. (2004): Performance Tuning and Optimization of J2EE Applications on the JBoss Platform. In: Journal of Computer Resource Management, Vol. 113 (2004), S. 40-49.

Kourjanski, M.; Varaiya, P. (1995): Stability of Hybrid Systems. In: Hybrid Systems III, LNCS 1066, Springer, (1995), S. 413-423.

Krcmar, H. (2010): Informationsmanagement. 5. vollst. überarb. u. erw. Aufl., Springer-Verlag, Berlin, Heidelberg 2010.

Kühnemund, H. (2007): Documentation for SLCS v.2.3. SAP AG, Linux Lab, Walldorf, Germany 2007.

Lantzsch, G.; Schneider, A.; Schwarz, P. (2000): Verteilte ereignisorientierte Simulation auf Basis von High Level Architecture (HLA). In: ASIM-KI-Workshop

"Multiagentensysteme und individuenbasierte Simulatoren" Hrsg., Würzburg 2000, S. 121-127.

Lazowska, E.D.; Zahorjan, J.; Graham, G.S.; Sevcik, K.C. (1984): Quantitative System Performance: Computer System Analysis Using Queueing Network Models. Prentice-Hall, Englewood Cliffs, NJ, USA 1984.

Li, L.; Franks, G. (2009): Performance Modeling of Systems Using Fair Share Scheduling with Layered Queueing Networks. In: 17th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2009) Hrsg., Imperial College, London 2009, S. 1-10.

Libic, P.; Tuma, P.; Buley, L. (2009): Issues in performance modeling of applications with garbage collection. *Proceedings of the 1st international workshop on Quality of service-oriented software systems* (S. 3-10). Amsterdam, The Netherlands: ACM.

Lilienthal, C. (2008): Komplexität von Softwarearchitekturen, Universität Hamburg 2008.

Lilja, D.J. (2000): Measuring Computer Performance - A Practitioner's Guide. Cambridge University Press, Cambridge 2000.

Lindley (1952): The Theory of Queues with a Single Server. In: Cambridge Philosophical Society, Vol. 48 (1952), S. 277-289.

Little, J.D.C. (1961): A Proof for the Queueing Formula L = lamda * W. In: Operations Research, Vol. 9 (1961) Nr. 3, S. 383-387.

LQsim (2011): LQsim Man Page. In: http://www.sce.carleton.ca/rads/lqns/lqn-documentation/lqsim.txt, zugegriffen am 18.10.2011.2011.

Lunze, J. (2006): Ereignisdiskrete Systeme. Modellierung und Analyse dynamischer Systeme und Automaten, Markovketten und Petrinetzen., Oldenbourg, München, Wien 2006.

March, S.T.; Smith, G. (1995): Design and Natural Science Research on Information Technology. In: Decision Support Systems, Vol. 15 (1995) Nr. 4, S. 251-266.

March, S.T.; Storey, V.C. (2008): Design science in the information systems discipline: an introduction to the special issue on design science research. In: MIS Q., Vol. 32 (2008) Nr. 4, S. 725-730.

Marquard, U.; Götz, C. (2008): SAP Standard Application Benchmarks - IT Benchmarks with a Business Focus. In: (Vol. 5119). Hrsg.: Kounev, S.; Gorton, I.; Sachs, K. Springer Berlin / Heidelberg, 2008, S. 4-8.

Mayer, M.; Gradl, S.; Schreiber, V.; Wittges, H.; Krcmar, H. (2011): A Survey on Performance Modelling and Simulation of SAP Enterprsie Resource Planning Systems. In: 10th International Conference on Modeling and Applied Simulation Hrsg., Rom 2011, S. 347-352.

McMahon, F.H. (1986): Livermore Fortran Kernels: A Computer Test of Numerical Performance Range. Lawrence Livermore National Laboratory, Livermore, CA 1986.

Meissner, G. (1999): SAP die heimliche Software-Macht. Wilhelm Heyne Verlag, München 1999.

Melzer, I. (2010): Service-orientierte Architekturen mit Web-Services. 4. Aufl., Spektrum, Akademischer Verlag, Heidelberg, Neckar 2010.

Menascé, D.A.; Almeida, V.A.F. (2002): Capacity Planning for Web Services: Metrics, Models, and Methods. Prentice-Hall, Englewood Cliffs, NJ 2002.

Merlin, P.M.; Farber, D.J. (1976): Recoverability of Communication Protocols: Implications of a Theoretical Study. In: IEEE Transactions on Communications, Vol. 24 (1976) Nr. 9, S. 1036-1043.

Mertens, P. (2001): Integrierte Informationsverarbeitung 1. 13. Aufl., Gabler, Wiesbaden 2001.

Meyer, E.; Guicking, D. (1974): Schwingungslehre. Vieweg-Verlag, Braunschweig 1974.

Michel, D. (2010): Active Memory Expansion Performance 2010.

Microsoft (2011): MSDN: WebBrowser-Control - Reference for C/C++ Developers. In: http://msdn.microsoft.com/en-us/library/aa752040(v=vs.85).aspx, zugegriffen am 30.12.2011.2011.

Mohr, M.; Simon, T.; Krcmar, H. (2005): Building an Adaptive Infrastructure for Education Service Providing. In: Tagung Wirtschaftsinformatik 2005 Hrsg. Springer, Bamberg 2005, S. 847-859.

Moore, G.E. (1965): Cramming More Components onto Integrated Circuits. 1965.

Müller, J. (2005): Workflow-based Integration. Grundlagen, Technologien, Management. Springer, Berlin 2005.

Nah, F.F.; Zuckweiler, K.M.; Lau, J.L. (2003): ERP Implementation: Chief Information Officers' Perceptions of Critical Success Factors. In: Int. J. Human-Computer Interaction, Vol. 16 (2003), S. 101-123.

Neilson, J.E. (1991): PARASOL: A Simulator for Distributed and/or Parallel Systems. (Vol. SCS TR-192). Ottawa, Kanada: Carleton University.

Neilson, J.E.; Woodside, C.M.; Petriu, D.C.; Majumdar, S. (1995): Software Bottlenecking in Client-Server Systems and Rendezvous Networks. In: IEEE Trans. Softw. Eng., Vol. 21 (1995) Nr. 9, S. 776-782.

Nicolescu, V.; Klappert, K.; Krcmar, H. (2007): SAP Netweaver Portal. 1. Aufl., Galileo Press, Bonn 2007.

Niehans, J. (1990): History of Economic Thought. The John Hopkins University Press, Baltimore 1990.

Nissen, V.; Petsch, M.; Schorcht, H. (2008): Service-orientierte Architekturen. Chancen und Herausforderungen bei der Flexibilisierung und Integration von Unternehmensprozessen., Deutscher Universitäts-Verlag / GWV Fachverlage GmbH, Wiesbaden 2008.

Nunamaker, J.F.; Dennis, A.R.; Valacich, J.S.; Vogel, D.; George, J.F. (1991): Electronic Meeting Systems. In: Communications of the ACM - Special Issue on Computer Graphics: State of the Arts, Vol. 34 (1991) Nr. 7, S. 40-61.

Obaidat, M.S.; Boudriga, N. (2010): Fundamentals of performance evaluation of computer and telecommunication systems. John Wiley & Sons., Hoboken, N.J. 2010.

Oed, W.; Mertens, B. (1981): Characterization of computer system workload. In: Computer Performance, Vol. 2 (1981) Nr. 2, S. 77-83.

Oracle (2011): Java SE HotSpot at a Glance. In: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136373.html, zugegriffen am 08.12.2011.2011.

Orth, B. (1974): Einführung in die Theorie des Messens. Kohlhammer, Stuttgart 1974.

Park, A.; Becker, J.C. (1990): IOStone: A Synthetic File System Benchmark. In: Computer Architecture News, Vol. 18 (1990) Nr. 2, S. 45-52.

Parupudi, M.; Winograd, J. (1972): Interactive Task Behavior in a Time-Sharing Environment. In: ACM Annual Conference Hrsg., Boston, MA 1972, S. 680-692.

Peterson, J.L. (1977): Petri Nets. In: ACM Computing Surveys, Vol. 9 (1977) Nr. 3, S. 223-252.

Petri, C.A. (1962): Kommunikation mit Automaten, Institut für Instrumentelle Mathematik, Universität Bonn 1962.

Petriu, D. (1994): Approximate Mean Value Analysis of Client-Server Systems with Multi-Class Requests. In: ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems Hrsg., Nashville, TN 1994, S. 77-86.

Petriu, D.; Amer, H.; Majumdar, S.; Al-Fatah, I. (2000): Using Analytic Models for Predicting Middleware Performance. In: Second International Workshop on Software and Performance Hrsg., Ottawa, Canade 2000, S. 189-194.

Petriu, D.; Shousha, C.; Jalnapurkar, A. (2000): Architecture-Based Performance Analysis Applied to a Telecommunication System. In: IEEE Transactions on Software Engineering, Vol. 26 (2000) Nr. 11, S. 1049-1065.

Pfeffers, K.; Tuunanen, T.; Gengler, C.E.; Rossi, M.; Hui, W.; Virtanen, V.; Bragge, J. (2006): The Design Science Research Process: A Model for Producing and Presenting Information Systems Research. In: Proceedings of the 1st International Conference on Design Science in Inofrmation Systems and Technology Hrsg.: Chatterjee, S.; Hevner, A., Claremont 2006, S. 83-106.

Pollaczek, F. (1930): Über eine Aufgabe der Wahrscheinlichkeitstheorie. In: Mathematische Zeitschrift, Vol. 32 (1930), S. 64-100.

Popp, K. (2002): Nutzbarmachung von Portaltechnologie: mySAP Enterprise Portals. In: HMD, Vol. 39 (2002) Nr. 225, S. 21-29.

Prechelt, L. (2001): Kontrollierte Experimente in der Softwaretechnik: Potenzial und Methodik. Springer, Berlin 2001.

Prior, D. (2003): Who Sets the Pace in the SAP Performance 'Olympics'? In: Gartner, (2003), S. 6.

Raghavachari, M.; Reimer, D.; Johnson, R.D. (2003): The Deployer's Problem: Configuring Application Servers for Performance and Reliability. In: 25. International Conference on Software Engineering Hrsg. IEEE Computer Society, Portland, OR 2003, S. 484-489.

Raychaudhuri, S. (2008): Introduction to Monte Carlo Simulation. In: Winter Simulation Conference (2008) Hrsg., Miami, FL 2008, S. 91-100.

Rechenberg, P.; Pomberger, G.; Pirklbauer, K. (2006): Informatik.Handbuch. 4. aktual. u. erw. Aufl., Hanser Fachbuchverlag, München 2006.

Risse, T. (2006): Design and Configuration of Distributed Job Processing Systems. Thesis (PhD), Technische Universität Darmstadt 2006.

Robertazzi, T.G. (2000): Computer Networks and Systems: Queueing Theory and Performance Evaluation. Springer, New York, New York, USA 2000.

Rolia, J.; Casale, G.; Krishnamurthy, D.; Dawson, S.; Kraft, S. (2009): Predictive modelling of SAP ERP applications: challenges and solutions. *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools* (S. 1-9). Pisa, Italy: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Rolia, J.A. (1988): Performance Estimates for Systems with Software Servers: The Lazy Boss Method. In: VIII SCCC International Conference on Computer Science Hrsg., 1988, S. 25-43.

Rolia, J.A.; Servcik, K.C. (1995): The Method of Layers. In: IEEE Transactions on Software Engineering, Vol. 21 (1995) Nr. 8, S. 689-700.

Rolia, J.A.; Sevcik, K.C. (1995): The Method of Layers. In: IEEE Trans. on Software Engineering, Vol. 21 (1995) Nr. 8, S. 689-700.

Rubinstein, R.Y.; Kroese, D.P. (2008): Simulation and the Monte Carlo Method. 2. Aufl., Wiley-Interscience, Hoboken, NJ 2008.

SAP (2012): SAP In-Memory Computing. In: http://www.sap.com/germany/plattform/in-memory-computing/index.epx, zugegriffen am 03.01.2012.2012.

SAP (2011a): Active Monitors. In: http://help.sap.com/saphelp_nw04/Helpdata/EN/43/84be64d44a22aee10000000a1553f6/content.htm, zugegriffen am 10.09.2011.2011.

SAP (2011b): Application Platform by Key Capability. In: http://help.sap.com/saphelp_nw70/helpdata/en/17/f1b640c9aa054fa12493e48912909c/content.htm, zugegriffen am 18.10.2011.2011.

SAP (2011c): Architecture of the Locking Service Adapter. In: http://help.sap.com/saphelp_nw04/helpdata/en/9a/4cdcc80fa4c747a2ccb5859f467412/content.htm, zugegriffen am 19.09.2011.2011.

SAP (2011d): Composite Application Framework by Key Capability. In: http://help.sap.com/saphelp_nw70/helpdata/en/0c/58497f411a7848985ae2aa0dda0bd3/frameset.htm, zugegriffen am 18.10.2011.2011.

SAP (2011e): Configuring Tables. In: http://help.sap.com/saphelp_nw04/helpdata/de/bf/86eb3f0c49f106e10000000a1550b0/content.htm, zugegriffen am 12.10.2011.2011.

SAP (2011f): Distributed Statistics Records (DSRs). In: http://help.sap.com/saphelp_nw04/helpdata/de/ee/40fd2b396dd442a1ac1e0409bce5c9/content.htm, zugegriffen am 03.08.2011.2011.

SAP (2011g): DSR Example: Simple Transaction Step. In: http://help.sap.com/saphelp_nw04/helpdata/en/1d/58f85b87f12e49ab9bf4a93a799d58/content.htm, zugegriffen am 04.09.2011.2011.

SAP (2011h): Employee Self-Service Portal (EP-ESS). In: http://www.sap.com/solutions/benchmark/ep-ess.epx, zugegriffen am 14.12.2011.2011.

SAP (2011i): Information Integration: Key Areas. In: http://help.sap.com/saphelp_nw70/helpdata/en/0c/58497f411a7848985ae2aa0dda0bd3/frameset.htm, zugegriffen am 18.10.2011.2011.

SAP (2011j): Integration Processes (ccBPM). In: http://help.sap.com/saphelp_nw70/helpdata/en/3c/831620a4f1044dba38b370f77835cc/frameset.htm, zugegriffen am 09.10.2011.2011.

SAP (2011k): Lock Table. In: http://help.sap.com/saphelp_gts72/helpdata/en/f4/670f9b9d62f94db4ea7361b34ea214/frameset.htm, zugegriffen am 18.10.2011.2011.

SAP (2011l): People Integration by Key Capability. In: http://help.sap.com/saphelp_nw70/helpdata/en/0c/58497f411a7848985ae2aa0dda0bd3/frameset.htm, zugegriffen am 18.10.2011.2011.

SAP (2011m): Portal Architecture. In: http://help.sap.com/saphelp_nw04/helpdata/en/44/42bfc481ce2152e10000000a114a6b/content.htm, zugegriffen am 03.09.2011.2011.

SAP (2011n): Solution Life Cycle Management by Key Capability. In: http://help.sap.com/saphelp_nw70/helpdata/en/0c/58497f411a7848985ae2aa0dda0bd3/frameset.htm, zugegriffen am 18.10.2011.2011.

SAP (2002): Siemens and SAP Agree to Provide mySAP(tm) Enterprise Portal to All Siemens Employees as Well as Its Global Partners, Customers, and Suppliers. In: http://www.sap.com/press.epx?pressid=1400, zugegriffen am 01.04.2012.2012.

SAP Education (2011): SAP Education. In: www.sap.com/germany/services/education/index.epx, zugegriffen am 19.12.2011.2011.

SAP Help (2011): SAP Help. In: http://help.sap.com, zugegriffen am 19.12.2011.2011.

SAP UA (2012): SAP University Alliances. In: http://www.sap.com/corporate-de/sustainability/csr/education/alliance.epx, zugegriffen am 10.03.2012.2012.

SAP UCC TUM (2012): SAP University Alliances EMEA - SAP University Competence Center. In: http://www.sap-ucc.com/, zugegriffen am 03.03.2012.2012.

Sauer, C.H.; Chandy, K.M. (1981): Computer Systems Performance Modeling. Prentice-Hall, Englewood Cliffs, NJ, USA 1981.

Scherr, A.L. (1965): An Analysis of Time-Shared Computer Systems, Massachusetts Institute of Technology 1965.

Schneider, T. (2008): SAP-Performanceoptimierung. 5. Aufl., Galileo Press, Bonn, Boston 2008.

Schulze, P.M. (2007): Beschreibende Statistik. 6. Aufl., Oldenbourg Wissenschaftsverlag GmbH, München 2007.

Schwarze, J. (1997): Einführung in die Wirtschaftsinformatik. 4. Aufl., Herne, Berlin 1997.

Schwarzer, B.; Krcmar, H. (2004): Wirtschaftsinformatik: Grundzüge der betrieblichen Datenverarbeitung. 3. Aufl., Schäffer-Poeschel Verlag, Stuttgart 2004.

Shein, B.; Callahan, M.; Woodbuy, P. (1989): NFSStone - A Network File Server Performance Benchmark. In: USENIX Summer Tech. Conf. Hrsg., Baltimore, MD 1989, S. 269-275.

Shousha, C.; Petriu, D.; Jalnapurkar, A.; Ngo, K. (1998): Applying Performance Modelling to a Telecommunication System. In: First International Workshop on Software and Performance Hrsg., Santa Fe, New Mexico 1998, S. 1-6.

Siebert, G.; Kempf, S.; Maßalski, O. (2008): Benchmarking. Leitfaden für die Praxis. Hanser, München 2008.

Simon, H.A. (1996): The Sciences of the Artificial. 3. Aufl., MIT Press, Cambridge, MA 1996.

Simon, H.A. (1970): Administrative Behavior. A Study of Decision-Making Process in Administrative Organization. Macmillan, New York, NY 1970.

Snodgrass, R. (1988): A Relational Approach to Monitoring Complex Systems. In: ACM Transactions on Computer Science, Vol. 6 (1988) Nr. 2, S. 157-195.

Söbbing, T. (2006): Handbuch IT-Outsourcing. 3., völlig neu bearb. Aufl., C.F. Müller, Heidelberg, München, Landsberg, Berlin 2006.

Staples, E. (1987): The Tower of Hanoi: Problem with Arbitrary Start and End Positions. In: ACM SIGACT News, Vol. 18 (1987) Nr. 3, S. 61-64.

Tanenbaum, A.S. (2006): Moderne Betriebssysteme. Pearson Studium, München 2006.

Tertilt, D.; Krcmar, H. (2011): Generic Performance Prediction for ERP and SOA Applications. In: 19th European Conference on Information Systems Hrsg., Helsinki, Finnland 2011.

Tertilt, D.; Leimeister, S.; Gradl, S.; Mayer, M.; Krcmar, H. (2010): Towards an evolutionary model generation for ERP performance simulation. *International Conference on Intelligent Systems and Agents*. Freiburg.

Teuffel, M.; Vaupel, R. (2010): Das Betriebssystem z/OS und die zSeries. Die Darstellung eines modernen Großrechnersystems., Oldenbourg, München 2010.

Thompson, W. (1884): Notes of Lectures on Molecular Dynamics and the Wave Theory of Light. Baltimore 1884.

TPC (1990): TPC Benchmark B Standard Specification. Transaction Processing Performance Council, San Jose, CA 1990.

TPC (1989): TPC Benchmark A Standard Specification. Transaction Processing Performance Council, San Jose, CA 1989.

Trivedi, K.S.; Haverkort, B.R.; Rindos, A.; Mainkar, V. (1994): Techniques and tools for reliability and performance evaluation: problems and perspectives. In: Proceedings of the 7th International Conference on Computer Performance Evaluation Hrsg. Springer-Verlag New York, Inc., Secaucus, NJ, USA 1994, S. 1-24.

Tschense, A. (2004): Java-Monitoring-Infrastruktur in SAP NetWeaver '04. Galileo Press, Bonn 2004.

Ufimtsev, A. (2006): Vertical Performance Modelling and Evaluation of Component-Based Software Systems, National University of Ireland 2006.

Ufimtsev, A.; Murphy, L. (2006): Performance modeling of a JavaEE component application using layered queuing networks: revised approach and a case study. *Proceedings of the 2006 conference on Specification and verification of component-based systems* (S. 11-18). Portland, Oregon: ACM.

Versick, D. (2010): Verfahren und Werkzeuge zur Leistungsmessung, -analyse und -bewertung der Ein-/Ausgabeeinheiten von Rechensystemen, Universität Rostock 2010.

Vogel-Heuser, B. (2003): Systems Software Engineering [angewandte Methoden des Systementwurfs für Ingenieure. Oldenbourg, München 2003.

W3C (2011): Markup Validation Service. In: http://validator.w3.org, zugegriffen am 29.12.2011.2011.

Wallace, T.F.; Kremzar, M.H. (2001): ERP: Make It Happen - The Implementers' Guide to Success with Enterprise Resource Planning. 1. Aufl., Wiley, Hoboken, NJ 2001.

Wannenwetsch, H.H.; Nicolai, S. (2004): E-Supply-Chain-Management. 2. Aufl., Vieweg, Wiesbaden 2004.

Wassermann, G.; Yu, D.; Chander, A.; Dhurjati, D.; Inamura, H.; Su, Z. (2008): Dynamic Test Input Generation for Web Applications. In: International Symposium on Software Testing and Analysis Hrsg., Seattle, WA 2008, S. 249-260.

Weicker, R.P. (1990): An Overview of Common Benchmarks. In: IEEE Computer, Vol. 23 (1990) Nr. 12, S. 65-75.

Weiss, H.P. (1993): Benchmarking: Learning from the Best. . In: CPA Journal Online, Vol. 63 (1993) Nr. 10, S. 81.

Wilhelm, K. (2001): Capacity Planning for SAP - Concepts and Tools for Performance Monitoring and Modelling, Essen 2001.

Winkler, S. (2010): Monitoring. Kritische Prozess- und Projektaktivitäten mithilfe persönlicher Assistenten. 1. Aufl., Josef Eul Verlag Gmbh, Erlangen-Nürnberg 2010.

Winter, R. (2009): Interview mit Alan R. Hevner zum Thema „Design Science“. In: Wirtschaftsinformatik, Vol. 51 (2009) Nr. 1, S. 148-151.

Wöhe, G. (2005): Allgemeine Betriebswirtschaftrslehre. Vahlens, München 2005.

Woodside, C.M. (1989): Throughput Calculation for Basic Stochastic Rendezvous Networks. In: Performance Evaluation, Vol. 9 (1989) Nr. 2, S. 143-160.

Woodside, C.M.; Neilson, J.E.; Petriu, D.C.; Majumdar, S. (1995): The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software. In: IEEE Transactions on Computers, Vol. 44 (1995) Nr. 1, S. 20-34.

Woodside, C.M.; Neron, E.; Ho, E.D.S.; Mondoux, B. (1986): An "Active-Server" Model for the Performance of Parallel Programms Written Using Rendezvous. In: Journal of Systems and Software, Vol. 6 (1986) Nr. 1-2, S. 125-131.

Woodside, M. (2002): Tutorial Introduction to Layered Modeling of Software Performance. Carleton University, Ottawa, Canada 2002.

Xu, J.; Oufimtsev, A.; Woodside, M.; Murphy, L. (2005): Performance modeling and prediction of enterprise JavaBeans with layered queuing network templates. In: SIGSOFT Softw. Eng. Notes, Vol. 31 (2005) Nr. 2, S. 5.

Zieher, M. (1989): Kopplung von Rechnernetzen - Techniken zu Planung, Entwurf, Vermessung und Leistungsoptimierung, Universität Karlsruhe 1989.

Zitterbart, M. (1990): Monitoring and debugging transputer-networks with NETMON-II. *Proceedings of the joint international conference on Vector and parallel processing* (S. 200-209). Zurich, Switzerland: Springer-Verlag New York, Inc.

Zuse, H. (1998): A Framework of Software Measurement. Walter de Gruyter & Co., Berlin 1998.

Zwerenz, K. (2009): Statistik: Einführung in die computergestützte Datenanalyse. Oldenbourg Wissensch.Vlg, München 2009.

7 Anhang

Portaloperationen der Fallstudie

ID	Funktion	Duplikat von	Fremd-funktion
Tag 1 – Vorbereitungen			
1.0	Portal-Seite öffnen		
1.1	Logon Admin		
1.2	User anlegen		
1.3	Gruppe anlegen		
1.4	User zu Gruppen (Administrator) hinzufügen		
1.5	Ordner anlegen		
1.6	Rolle anlegen		
1.7	Objekt (Rolle) öffnen		
1.8	Einstiegspunkt aktivieren		
1.9	Rolle zu Gruppe zuordnen		
1.10	Rolle anpasse		
1.11	Leserechte für Portal-Content vergeben		
1.12	Schreibrechte für Ordner anlegen		
1.13	Kurskonfiguration prüfen		
1.14	Logout		
Tag 2 – Login, Navigation, Personalisierung, Menü			
2.0	Benutzer anmelden	1.1	
2.1	Initiales Passwort ändern		
2.2	Benutzer abmelden	1.14	
2.3	Benutzer wieder anmelden	1.1	
2.4	Zur Startseite wechseln		
2.5	Personalisieren: Darstellungsschema ändern		
2.6	Personalisieren: Darstellungsschema speichern		
2.7	User-Mapping (für R/3-Verbindungen) einrichten		X
2.8	Detaillierte Benutzerinformationen eintragen		
2.9	Top-Level-Navigation: Unterpunkte aufrufen		
2.10	Startseite: Portal-Information abrufen		
2.11	Content Management: Direct-Links zu Komponenten		
2.12	Kollaboration: Grundlage für asynchrone Zusammenarbeit		
2.13	Curriculum-Congress: Personalisierter Inhalt		
2.14	Wechseln zur Inhaltsverwaltung	2.4	
2.15	History-Funktion: Wechseln zu Curriculum-Congress		
2.16	Create-Inquiry (VA11) – R/3-System einbinden		X
2.17	Navigation: Überblick über „field areas“		
2.18	Benutzer abmelden	1.14	
Tag 3 - Inhaltsverwaltung			
3.0	Benutzer anmelden	1.1	

3.1	Vorschau SAP-iViews		
3.2	Ordner anlegen	1.5	
3.3	iView in neuen Ordner kopieren		
3.4	Neues iView anlegen		
3.5	Neues iView konfigurieren		
3.6	Portal-Webseite erstellen		
3.7	Portal-Webseite konfigurieren		
3.8	neues Workset erstellen		
3.9	neues Workset konfigurieren		
3.10	Webseite mit Workset verlinken (Delta-Link)		
3.11	Objekt (Rolle „class role“) öffnen	1.7	
3.12	Workset mit Rolle verlinken (Delta-Link)		
3.13	Workset schließen		
3.14	Rolle „Class role“ aktualisieren		
3.15	Erstellte Webseite öffnen	3.1	
3.16	Benutzer abmelden	1.14	
Tag 4 – Forgeschrittene Inhaltsverwaltung			
4.0	Benutzer anmelden	1.1	
4.1	Neues iView anlegen (Transaktion-iView)	3.4	
4.2	Neues iView konfigurieren	3.5	
4.3	Neues iView aufrufen		X
4.4	Neues iView anlegen (RSS-Feed-iView)		X
4.5	Neues iView konfigurieren	3.5	
4.6	Neues iView aufrufen	4.3	
4.7	Benutzer abmelden	1.14	
Tag 5 – Kollaboration			
5.0	Benutzer anmelden	1.1	
5.1	SAP-Meeting-Room erstellen		
5.2	SAP-Meeting-Room konfigurieren		
5.3	Benutzer/Gruppen zum SAP-Meeting-Room hinzufügen		
5.4	Änderungen abspeichern		
5.5	„Room Directory“ aufrufen		
5.6	Eigenen Meeting-Room aufrufen		
5.7	Eigenen Task anlegen (Single-Step)		
5.8	Collaboration-Launch-Pad (CLP) aufrufen		
5.9	User zum CLP hinzufügen		
5.10	E-Mail an CLP-Benutzer verschicken		X
5.11	Instant-Messaging mit CLP-Benutzer		X
5.12	Application-Sharing mit CLP-Benutzer		X
5.13	Benutzer abmelden	1.14	
Tag 6 – Öffentliche Dokumente, Versionierung			
6.0	Benutzer anmelden	1.1	
6.1	Ordner für öffentliche Dokumente erstellen	1.5	
6.2	Öffentliches Dokument erstellen		

6.3	Inhalt für öffentliches Dokument eingeben		
6.4	Öffentliches Dokument versionieren		
6.5	Öffentlichen Ordner öffnen		
6.6	Benutzer abmelden	1.14	
Tag 7 – Benutzerverwaltung und Sicherheit			
7.0	Benutzer anmelden	1.1	
7.1	Benutzer anlegen	1.2	
7.2	Benutzerdaten eintragen	2.8	
7.3	Benutzer zu Gruppen hinzufügen (Administrator)	1.4	
7.4	Benutzerprofil speichern	2.6	
7.5	Ordner anlegen	1.5	
7.6	Rolle anlegen	1.6	
7.7	Portal-Content-Objekt öffnen	1.7	
7.8	Einstiegspunkt aktivieren	1.8	
7.9	Rolle einer Gruppe zuordnen	1.9	
7.10	Benutzerpasswort ändern		
7.11	Bneutzer abmelden	1.14	
Tag 8 – Developer Studio, Visual Composer			
8.0	Benutzer anmelden	1.1	
8.1	Startseite: zu Composer navigieren		X
8.2	Visual Composer öffnen		X
8.3	Benutzer abmelden	1.14	
X – Nachbereitungen			
X.0	Benutzer anmelden	1.1	
X.1	Benutzer löschen		
X.2	Gruppen löschen		
X.3	Ordner löschen		
X.4	Benutzer abmelden	1.14	

Tabelle 7-1: Vollständige Transkription der Schulungsinhalte nach Funktionen
Quelle: Jehle (2010)