



FAKULTÄT II - DEPARTMENT FÜR INFORMATIK
Abteilung Software Engineering

Diplomarbeit

Empirische Bewertung von Performance-Analyseverfahren für Software-Architekturen

28. Oktober 2004

Bearbeitet von:

Heiko Koziolk
Schützenweg 42
26129 Oldenburg
heiko.koziolk@informatik.uni-oldenburg.de

Betreut von:

Erstgutachter: Jun.-Prof. Dr. Ralf Reussner
Zweitgutachter: Prof. Dr. Wilhelm Hasselbring
Dipl.-Wirtsch.-Inform. Steffen Becker
Dipl.-Math. Viktoria Firus

Inhaltsverzeichnis

Inhaltsverzeichnis	v
1. Einleitung	1
1.1. Definition: Was ist Performance?	1
1.2. Motivation: Warum ist Performance heute noch problematisch?	2
1.3. Software Engineering und Performance	3
1.4. Offene Probleme	4
1.5. Ziele der Arbeit	4
1.6. Forschungsmethode	5
1.7. Struktur der Ausarbeitung	5
2. Grundlagen des Performance Engineering	7
2.1. Einführung und Abgrenzung	7
2.2. Ursprünge und Geschichte des Performance Engineering	9
2.3. Performance Modellierung	10
2.3.1. Markow-Ketten	10
2.3.2. Warteschlangenmodelle	12
2.3.3. Stochastische Prozessalgebren	15
2.3.4. Stochastische Petri-Netze	15
2.4. Software-Modellierung	16
2.5. Software Performance Engineering	17
2.6. Capacity Planning	20
3. Methodik	23
3.1. Vorgehensweise	23
3.2. Fragestellungen und Metriken	24
3.2.1. Validität der Vorhersagen	24
3.2.2. Aufwand der Performance-Analyse	25
3.2.3. Methodisches gegenüber intuitiven Vorgehen	25
3.2.4. Bewertung einzelner Eigenschaften	26
3.2.5. Zusammenfassung	27
3.3. Forschungsmethode	28
3.3.1. Kontrolliertes Experiment	28
3.3.2. Fallstudie	29
3.4. Verwandte Arbeiten	30

4. Auswahl und Vorstellung der untersuchten Verfahren	33
4.1. Überblick Performance-Analyseverfahren	33
4.2. Kriterien für die Auswahl	35
4.3. Verfahren von Smith/Williams (SPE)	37
4.4. Verfahren von Marzolla/Balsamo (umlPSI)	39
4.5. Verfahren von Menasce/Almeida (CP)	40
5. Entwurf und Durchführung der Fallstudie	43
5.1. Randbedingungen	43
5.2. Kurzüberblick	43
5.3. Versuchsteilnehmer	46
5.4. Übung: (Einführung und Vortest)	47
5.4.1. Aufgabenstellung	47
5.4.2. Auswertung der Übungslösungen	48
5.4.3. Schlussfolgerungen aus den Übungen	51
5.5. Experiment	52
5.5.1. Webserver-Architektur	52
5.5.2. Aufgabenstellung	54
5.5.3. Entwurfsalternativen	58
5.5.4. Durchführung	59
5.6. Performance-Analyse der Implementierung	61
5.7. Validität	63
5.7.1. Konstruktgültigkeit	63
5.7.2. Innere Gültigkeit	63
5.7.3. Äußere Gültigkeit	65
5.7.4. Replizierbarkeit	66
6. Ergebnisse	67
6.1. Validität der Vorhersagen	67
6.1.1. Vorhersagen SPE	68
6.1.2. Vorhersagen Capacity Planning	71
6.1.3. Vorhersagen umlPSI	72
6.1.4. Messergebnisse der Implementierung	74
6.1.5. Vergleich von Vorhersagen und Messungen	76
6.1.6. Entwurfsentscheidungen	78
6.1.7. Interpretation und Erklärungsversuche	81
6.2. Aufwand der Performance-Analyse	86
6.3. Methodisches gegen intuitives Vorgehen	88
6.4. Bewertung einzelner Eigenschaften	89
6.4.1. Notation	89
6.4.2. Werkzeugunterstützung	91
6.4.3. Eignung für verschiedene Domänen	94
6.4.4. Auswertungsmethode	94
6.4.5. Prozessintegration	95

6.5. Daten der Verfahren	96
6.6. Bewertung der Verfahren	97
7. Schlussfolgerungen und Ausblick	99
7.1. Zusammenfassung	99
7.2. Anknüpfungspunkte für Folgearbeiten	102
7.3. Zukunft des Performance Engineering	102
Literaturverzeichnis	105
Abbildungsverzeichnis	113
Tabellenverzeichnis	115
A. Vergleichstabelle Performance-Analyseverfahren	i
B. Folien der Übungen	vii
C. Übungszettel	xli
D. Musterlösung Übungen	xlvi
E. Aufgabenstellungen des Experiment	li
F. Lösung des Experiments	lix
G. Rohdaten der Ergebnisse	lxvii
H. Diagramme zur Veranschaulichung	lxxxix
I. Messungen am Prototypen	lxxxvii
Danksagung	xcix
Selbstständigkeitserklärung	xciii

Zusammenfassung

Schon seit den Anfängen der Informatik wird versucht, die Leistungsfähigkeit von Software und Hardware zu analysieren. Komplexe Berechnungen sollen ermöglicht werden, gleichzeitig müssen die Antwortzeiten von Rechnersystemen für den Benutzer möglichst gering gehalten werden.

Nach vorherrschender Meinung von Praktikern kann die Performance eines Systems erst untersucht werden, wenn bereits Programmcode geschrieben wurde und mit diesem Messungen angestellt werden können. Diese Praktik kann zu hohen Kosten aufgrund von Reengineering-Maßnahme führen, wenn das System zuerst komplett implementiert wurde und anschließend eine schlechte Performance aufgrund einer mangelhaften Software-Architektur festgestellt wird. Aus diesem Grund wird im Performance Engineering versucht die Performance von Hardware/Software-Systemen schon zu einem frühen Zeitpunkt der Entwicklung vorherzusagen. Man arbeitet dabei mit Performance-Modellen, die die Leistungscharakteristiken eines Systems frühzeitig abbilden und an denen verschiedene Entwurfsalternative getestet werden können.

Viele verschiedene Notationen und Verfahren wurden für diese frühe Performance-Analyse in den letzten 20 Jahren entwickelt. Von einem weiträumigen Einsatz dieser Verfahren in der Software-Industrie kann jedoch noch nicht gesprochen werden.

In dieser Arbeit wurde daher eine vergleichende Fallstudie mit drei Verfahren zur Performance-Analyse von Software-Architekturen durchgeführt, um deren Anwendbarkeit zu bewerten. Vier Forschungsfragen über die Genauigkeit, den Aufwand, den Nutzen und die Eigenschaften der Verfahren wurden formuliert. Mit einem Versuchsaufbau ähnlich eines Experiments wurde versucht, Antworten auf diese Fragen zu finden.

Eine Gruppe von 24 Studenten untersuchte nach einer entsprechenden Schulung in den drei Verfahren verschiedene Designvorschläge für einen experimentellen Webserver. Dabei lag ihnen kein Programmcode vor, mit dem Messungen hätten angestellt werden können. Jeder Teilnehmer gab auf Basis der Ergebnisse der Verfahren eine Empfehlung, von welcher Entwurfsalternative bei einer Implementierung die beste Performance zu erwarten sei. Zusätzlich untersuchte eine Kontrollgruppe von sieben Teilnehmern die Performance des Webserver ohne Kenntnis der Verfahren.

Der Webserver wurde in allen vorgeschlagenen Varianten implementiert, so dass Messungen der Antwortzeiten durchgeführt werden konnten. Anschließend wurden die Vorhersagen der Studenten mit den tatsächlichen Werten verglichen. Auf diese Weise konnte die Genauigkeit der Vorhersagen analysiert werden. Ergänzend wurde eine Reihe von Eigenschaften der Verfahren einer kritischen Bewertung unterzogen, darunter z.B. die Modellierungsmächtigkeit der eingesetzten Notationen und die Praxistauglichkeit der zugehörigen Software-Werkzeuge.

1. Einleitung

In der folgenden Ausarbeitung werden verschiedene Verfahren zur Performance-Analyse von Software-Architekturen in einer Fallstudie auf ihre Einsatzfähigkeit hin untersucht und verglichen. In dieser Einleitung soll zunächst genauer erläutert werden, was unter dem umfassenden Begriff *Performance* verstanden wird, und warum die Performance trotz immer leistungsfähigerer Rechner auch heute noch ein schwerwiegendes Problem in der Informatik darstellt. Danach folgt ein kurzer Überblick über die Disziplin des *Performance Engineering*, bevor die Fragestellungen dieser Arbeit vorgestellt werden und die weitere Struktur der Ausarbeitung dargestellt wird.

1.1. Definition: Was ist Performance?

Performance ist in Bezug auf Software folgendermaßen definiert:

Performance is the degree to which a software system or component meets its objectives for timeliness. [[Smi02](#), p.4]

Unter 'timeliness' (deutsch: Aktualität, Rechtzeitigkeit) versteht man dabei das Antwortverhalten und die Skalierbarkeit eines Systems.

Das Antwortverhalten eines Systems bestimmt sich durch die Antwortzeit und den Durchsatz. Die *Antwortzeit* ist die Zeit, die zwischen der Anfrage eines Benutzers und der Beantwortung durch das System vergeht. Beispielsweise müssen bei einer Abfrage bei einem Online-Banking-System nicht nur die Berechnungszeiten des Systems für die Abfrage eines Kontostands, sondern auch die Übertragungszeiten für den Netzwerkverkehr über das Internet einbezogen werden. Die Antwortzeit kann bei mehreren Anfragen an eine Ressource in Bearbeitungszeit und in Wartezeit aufgeteilt werden. Der *Durchsatz* gibt die Anzahl der Anfragen wieder, die innerhalb eines Zeitintervalls bearbeitet werden können. Ein Online-Banking-System kann z.B. 10 Anfragen pro Sekunde bearbeiten.

Unter der *Skalierbarkeit* versteht man die Fähigkeit eines Systems, seine Funktionalität aufrecht zu erhalten, auch wenn seine Beanspruchung in großem Maße erhöht wird. Beispielsweise skaliert ein Webserver gut, wenn er auf 1000 gleichzeitige Benutzeranfragen genauso schnell reagiert wie auf nur 10 Anfragen.

Weitere Metriken für Performance von Systemen sind z.B. die prozentuale *Auslastung* von Ressourcen (CPU, Speicher, Netzwerkverbindungen, usw.) und die *Reaktionszeiten* bei Echtzeitanwendungen.

Bei der Performance von Hardware/Software-Systemen unterscheidet man auch zwischen einer inneren und äußeren Sicht des Systems. Mit der äußeren Sicht sind solche Performance-Metriken gemeint, die für die Benutzer des Systems relevant sind, also vor allem die Antwortzeit und die Berechnungsdauer. Die innere Sicht stellt Performance-Metriken für die Entwickler und Betreiber

des Systems in den Mittelpunkt, darunter fallen z.B. der Durchsatz und die Auslastung einzelner Ressourcen sowie die Skalierbarkeit des Systems.

1.2. Motivation: Warum ist Performance heute noch problematisch?

Dass die Performance von Software-Systemen auch nach über 50 Jahren Forschung in der Informatik noch immer ein großes Problem darstellt, scheint zunächst verwunderlich. Schließlich besagt das auch nach fast 40 Jahren nicht widerlegte Gesetz von Moore [Moo65], dass sich die Komplexität von integrierten Schaltkreisen alle 18 Monate verdoppelt und als Folge daraus die Rechenleistung von Computern im Verlauf der Zeit exponentiell ansteigt. Aus diesem Grund müssten Performance-Probleme auf Dauer einfach durch immer schnellere Hardware zu lösen sein. Dass dies bis heute nicht der Fall ist, zeigen eine Reihe von Beispielen.

Im Jahr 1993 übertraf die Entwicklung eines neuen Gepäckverarbeitungssystems für den Flughafen von Denver die geplanten Kosten um 2 Mrd. Dollar unter anderem auch deshalb, weil die Performance des Systems zu schwach war [SS01]. Das System war zunächst nur für eine einzelne Fluglinie geplant worden, wurde dann aber für alle Terminals des Flughafens benutzt, ohne die Auswirkungen der höheren Systemlast zu berücksichtigen. Durch die mangelhafte Performance musste das System mit großem Aufwand nachgebessert werden, und die Eröffnung des Flughafens verzögerte sich wegen des schlechten Projektmanagements um 16 Monate.

Bei den Olympischen Spielen 1996 wurde ein Informationssystem von IBM benutzt, um die Ergebnisse der Wettbewerbe auszuwerten [SS01]. Getestet wurde das System vor der Auslieferung mit 150 Benutzern. Während der produktiven Phase jedoch nutzten über 1000 Personen das System, was einen Systemkollaps zur Folge hatte und den Start einiger Wettbewerbe verzögerte. IBM erlitt infolgedessen einen herben Imageschaden, der nur schwierig in Zahlen auszudrücken ist.

Bei der Entwicklung des neuen Online-Stellenmarktes der Bundesagentur für Arbeit ging man 2003 von einer zu geringen Nutzerzahl von 500.000 Benutzern pro Tag aus. [New03]. Nach der Eröffnung des Stellenmarktes wollten dann mehr als doppelt so viele Besucher pro Tag darauf zugreifen, so dass es zu langen Verzögerungen kam. Die mangelnde Performance des Web-Systems erntete heftige Kritik und schadete dem Ansehen der Agentur. Die zunächst auf 65 Millionen Euro veranschlagten Kosten wurden durch die Überarbeitung des Systems auf über 100 Millionen Euro gesteigert.

Viele weitere Beispiele lassen sich anführen. Glass [Gla98] identifiziert Performance-Probleme sogar als die zweithäufigste Ursache für gescheiterte Software-Projekte. Die Gründe für die anhaltenden Performance-Probleme trotz immer schnellerer Hardware sind dabei vielfältig. Moderne Software-Systeme werden ständig komplexer und bestehen heute häufig aus über das Internet verteilten, zusammenarbeitenden Komponenten. Die Anforderungen der Benutzer an ihre Software haben sich erheblich gesteigert. Die neue, immer schnellere Hardware macht neue Software-Lösungen erst möglich, so dass die Größe und Komplexität neuer Software die Hardware-Verbesserungen wieder aufwiegt [Smi01]. Bei den frühen Computer-Systemen wurden Performance-Probleme meist auf der algorithmischen Ebene betrachtet. Die Arbeiten von D.E. Knuth [Knu73] über die ausführlichen Effizienzanalysen von Such- und Sortieralgorithmen sind ein Beleg dafür. Nachdem diese Probleme heute größtenteils als gelöst angesehen werden können,

hat sich die Betrachtung von Performance-Problemen mittlerweile mehr auf die Architekturbene verlagert. Nicht zuletzt durch die rasante Entwicklung des Internets und die Entstehung immer komplexerer Hardware/Software-Systeme sind Performance-Analysen von Architekturen heute äußerst relevant.

1.3. Software Engineering und Performance

In der Praxis der Software-Entwicklung werden nicht-funktionale Eigenschaften von Software wie Performance, Zuverlässigkeit, Sicherheit usw. in den frühen Entwicklungsphasen meist nicht berücksichtigt oder vernachlässigt. Im Fokus der Entwicklung steht häufig allein die Funktionalität der neuen Software. Die Performance z.B. wird oftmals erst nach erfolgter Implementierung während des Systemtests untersucht. Viele Entwickler sind der Ansicht, dass die Performance-Probleme erst dann behoben werden müssen, wenn sie wirklich auftreten ('fix-it-later'-Einstellung [Smi02]). Bei komplexen Software-Architekturen jedoch lassen sich Performance-Mängel in den späten Entwicklungsphasen oft nur mit großem Aufwand beseitigen. Häufig ist dann ein Neuentwurf einzelner Teile der Software notwendig. Damit verbunden müssen große Abschnitte des Quellcodes geändert oder neu geschrieben werden, was sehr hohe Kosten nach sich ziehen kann.

Um diese Problematik zu lösen, wird bereits seit Anfang der 80er-Jahre in der Forschung versucht, die Analyse nicht-funktionaler Eigenschaften von Software bereits in die frühen Entwicklungsphasen zu integrieren. Flaschenhalse in der Architektur, die die Performance hemmen, sollen auf diese Weise frühzeitig erkannt werden, noch bevor mit der Implementierung begonnen wird. Ein kostspieliges, nachträgliches Redesign der Architektur kann vermieden werden.

Ziel ist es, Performance-Eigenschaften von Software während des Entwurfs mittels Erwartungswerten oder Messungen an Prototypen vorherzusagen. Dazu wird bereits beim Entwurf ein Performance-Modell des späteren Systems erstellt. Ein solches Modell wird so attribuiert, dass es die erwarteten Performance-Eigenschaften der geplanten Software-Architektur nachbildet. In den frühen Entwurfsphasen stützt sich das Modell beispielsweise auf Performance-Schätzungen der Entwickler, die aufgrund ihrer Erfahrung die Dauer einzelner Aktionen vorhersagen können. Schreitet die Entwicklung voran und liegen bereits erste Codefragmente oder Prototypen des Systems vor, so können diese verwendet werden, um das Performance-Modell mit Messungen zu verfeinern.

Auch wenn diese Modelle zu Beginn der Entwicklung meist keine exakten und absoluten Werte für Antwortzeiten der späteren Software oder Auslastungen von Ressourcen liefern können, so soll es dennoch möglich sein, relative Unterschiede zwischen unterschiedlichen Entwurfsalternativen festzustellen. Auf diese Weise können Designentscheidungen aus Performance-technischer Sicht validiert werden. Im Idealfall können so Performance-Probleme, die aus der Software-Architektur resultieren, von vornherein vermieden werden. Die Mehrkosten für frühe Performance-Analysen liegen nach Studien für typische große Softwaresysteme in der Größenordnung von 1-3 Prozent der gesamten Projektkosten [Smi02] und sind damit weit unterhalb der Kosten für ein nachträgliches Redesign der fertigen Implementierung.

1.4. Offene Probleme

An Verfahren zur frühen Performance-Analyse von Software-Architekturen wird bereits seit über 20 Jahre geforscht. Eine große Anzahl von Verfahren mit unterschiedlichsten Notationen wurde vorgeschlagen, ein Überblick über diese Verfahren findet sich in Kapitel 4. Dennoch hat sich bis heute keines der Verfahren in der Praxis weiträumig etabliert. Die Gründe dafür sind vielfältig und ähneln den Problemen anderer formaler Methoden im Software Engineering [RKFPS04]. Einige Verfahren nutzen komplizierte Notationen und erfordern umfangreiche Eingaben, die von den meisten Entwicklern nur mit großem Aufwand zu erstellen sind [Men02]. Teilweise werden die Performance-Modelle so komplex, dass sie nicht mehr in linearer Zeit ausgewertet werden können [Smi01]. Geeignete Werkzeuge, die die Erstellung von Performance-Modellen vereinfachen, sind noch kaum vorhanden. Eine Integration der Verfahren in verbreitete Modellierungswerkzeuge wie Together oder Rose ist noch nicht erfolgt [DS00]. Eine Kapselung der oftmals komplizierten Performance-Modelle bzw. deren automatische Erstellung aus bekannten Software-Spezifikationen z.B. in UML ist bei den meisten Verfahren nicht gegeben und hemmt deren Praxistauglichkeit. Auch sind die erhöhten Entwurfskosten durch die Einbeziehung von Performance-Aspekten für die Auftraggeber anfangs oft nicht plausibel. Ein weiteres Problem ist die noch völlig unzureichende Integration von Performance-Analysen in die Lehrpläne von Universitäten. Dugan zeigt in einer Studie [Dug04], dass fast 70 Prozent der Software-Engineering Kurse an amerikanischen Universitäten wenig oder gar keine Zeit für das Thema Performance aufwenden.

Für viele Verfahren besteht das Problem, dass keine Erfahrungsberichte, Feld- oder Fallstudien zur Validation vorliegen. Oftmals wurden sie nur von den Autoren selbst an kleinen Musterbeispielen getestet, ihre Tauglichkeit für große Praxisprojekte jedoch nur selten untersucht. Studien, bei denen die Performance-Vorhersagen der Verfahren mit Messungen an den späteren Implementierungen verglichen wurden, finden sich so gut wie gar nicht. Kontrollierte Experimente oder vergleichende Fallstudien, die verschiedene Verfahren auf die gleiche Architektur ansetzen, gibt es bisher auch nicht.

1.5. Ziele der Arbeit

Ziel dieser Ausarbeitung soll es daher sein, die Anwendbarkeit verschiedener Performance-Analyseverfahren aus der Sicht des Entwicklers durch eine empirische Studie zu evaluieren. Zur Erreichung dieses Ziels werden mehrere Forschungsfragen gestellt:

- Wie hoch ist die Genauigkeit der Verfahren?
- Wie hoch ist der Aufwand für die Erstellung eines Performance-Modells?
- Welche Vorteile bringt die Anwendung der Verfahren gegenüber einem intuitiven Vorgehen ohne Methodik?
- Wie sind ausgewählte Eigenschaften der Verfahren zu bewerten (z.B. Modellierungsmächtigkeit, Werkzeugunterstützung, Praxistauglichkeit)?

Für die Beantwortung der Fragen werden in Kapitel 3 entsprechende Metriken definiert.

1.6. Forschungsmethode

Im Idealfall müsste zum Vergleich von verschiedenen Verfahren ein kontrolliertes Experiment durchgeführt werden, das unter den empirischen Methoden den höchsten Grad an Vertrauen in die Beobachtungen genießt [Pre01]. Ein solches Experiment wird mit einer größeren Anzahl von Teilnehmern durchgeführt, um deren Qualifikation und Leistung anschließend durch statistische Mittelung zu nivellieren. Die Ergebnisse können dann direkt auf die Eigenheiten des untersuchten Verfahrens zurückgeführt werden und sind nicht mehr von den teilnehmenden Personen abhängig. Aufgrund der geringen Anzahl an Teilnehmern, die im Rahmen einer Diplomarbeit hinzugezogen werden können, und der nur bedingten Tauglichkeit der Verfahren für ein solches Experiment wird die Untersuchung hier in Form einer vergleichenden Fallstudie durchgeführt.

Die Fallstudie findet im Rahmen der Forschungsgruppe Palladio statt, die an der Universität Oldenburg Methoden und Werkzeuge zur Vorhersage der Zuverlässigkeit und Performance von komponentenbasierten Software-Architekturen entwickelt. Als Versuchsobjekt wird ein prototypischer Webserver untersucht, der in der Palladio-Gruppe von Yvette Teiken zu Testzwecken entwickelt wurde. Anhand der vorliegenden Entwurfsdokumente können Eingabeparameter für die Verfahren ermittelt werden. Eine Gruppe von Studenten wendet ausgewählte Performance-Analyseverfahren an, um verschiedene Entwurfsalternativen für den Webserver zu bewerten. Die Vorhersagen der Verfahren werden anschließend mit Messungen an der prototypischen Implementierung des Webserver verglichen.

Auf diese Weise können Erkenntnisse über die Verfahren gewonnen werden, auf deren Basis die Palladio-Gruppe eigene Verfahren entwickeln kann.

1.7. Struktur der Ausarbeitung

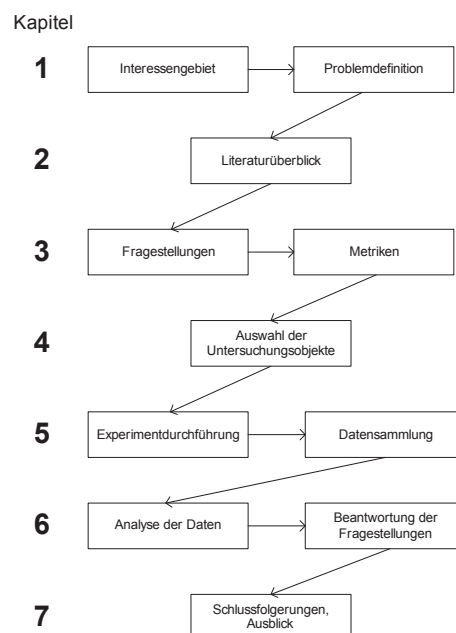


Abbildung 1.1.: Aufbau der Ausarbeitung

Der Aufbau dieser Ausarbeitung ist linear, die einzelnen Abschnitte bauen möglichst logisch aufeinander auf, um die gestellten Forschungsfragen zu beantworten. In Kapitel 2 wird zunächst ein Überblick über die Grundlagen des Performance Engineering gegeben, bevor dann in Kapitel 3 die Forschungsfragen an dieses Themengebiet formuliert und Metriken zu deren Beantwortung definiert werden. In Kapitel 4 findet sich ein Überblick über bekannte Performance-Analyseverfahren, aus denen dann drei Verfahren ausgewählt und näher vorgestellt werden. Der Aufbau des im Rahmen dieser Fallstudie durchgeführten Experiments wird in Kapitel 5 erläutert, bevor in Kapitel 6 die beobachteten Ergebnisse vorgestellt werden. Kapitel 7 fasst schließlich die Erkenntnisse der Arbeit zusammen, zieht Schlussfolgerungen und gibt Anknüpfungspunkte für Folgearbeiten. In den Anhang dieser Ausarbeitung wurden alle Materialien und Aufgabenblätter, die in dieser Fallstudie verwendet wurden, integriert. Es finden sich dort auch die Rohdaten der Ergebnisse, so dass eine nachträgliche Auswertung dieser Daten durch Dritte vorgenommen werden kann. Ebenfalls enthalten ist eine Vergleichstabelle der in Kapitel 4 kurz vorgestellten Performance-Analyseverfahren.

2. Grundlagen des Performance Engineering

Im nun folgenden Kapitel sollen Grundlagen des *Performance Engineering* näher ausgeführt werden. Es wird kurz auf die Geschichte des Performance Engineering eingegangen, verschiedene Ansätze für die Performance-Modellierung werden erläutert, und die Methoden des *Software Performance Engineering* und *Capacity Planning* werden näher vorgestellt. Ein Überblick über verschiedene Verfahren des Performance Engineering folgt in Kapitel 4.

2.1. Einführung und Abgrenzung

Der Begriff Performance Engineering leitet sich aus den beiden Begriffen *Software Engineering* und *Performance Management* ab. Definiert wird Performance Engineering wie folgt:

Definition (Performance Engineering): PE can be defined as a collection of methods for the support of the performance-oriented software development of application systems throughout the entire software development process to assure an appropriate performance-related product quality. PE becomes an interface between Software Engineering and Performance Management. [SS01]

Beim Performance Engineering soll also die Performance in jeder Phase des Software-Lebenszyklus berücksichtigt werden. Geht man von einem wasserfall-ähnlichen Vorgehensmodell bei der Softwareentwicklung aus, so können bereits während der Anforderungsanalyse die Performance-Ziele der Software festgelegt werden. Dies können z.B. Vorgaben für die maximalen vom Benutzer wahrgenommenen Antwortzeiten des Systems sein. Schon in dieser frühen Phase sollte ein Verständnis für die Schlüsselfaktoren der Systemperformance entwickelt werden. Die Performance-Ziele können in späteren Phasen der Entwicklung ergänzt und verfeinert werden.

Im anschließenden Entwurf erstellen die Entwickler eine Architektur für das spätere System. Die dabei entstehenden Entwurfsdokumente (z.B. UML-Diagramme) können im Performance Engineering dazu verwendet werden, erste Performance-Modelle zu konstruieren um Vorhersagen für die Performance-Charakteristiken des System anhand von Analysen oder Simulationen zu treffen. Da noch keine Implementierung vorliegt, müssen sich die Entwickler hier bei der Angabe von Performance-Werten für einzelne Berechnungen Erfahrung aus früheren Projekten einbringen und auf Messungen an ähnlichen Systemen oder Schätzungen zurückgreifen. Mit den Performance-Modellen sollen verschiedene Entwurfsalternativen auch ohne Implementierung bewertet werden können. Die in Kapitel 4 vorgestellten Verfahren zum *Software Performance Engineering* ermöglichen die frühe Performance-Analyse bereits während des Entwurfs.

Im Verlauf der Implementierung werden die einzelnen Komponenten der zuvor entworfenen Architektur in Programmcode realisiert. Schon an halbfertigen Teilen der Implementierung können Messungen der Performance vorgenommen werden. Die Ergebnisse können in die Performance-Modelle einfließen und dazu beitragen, die Vorhersagen für das Gesamtsystem zu verbessern.

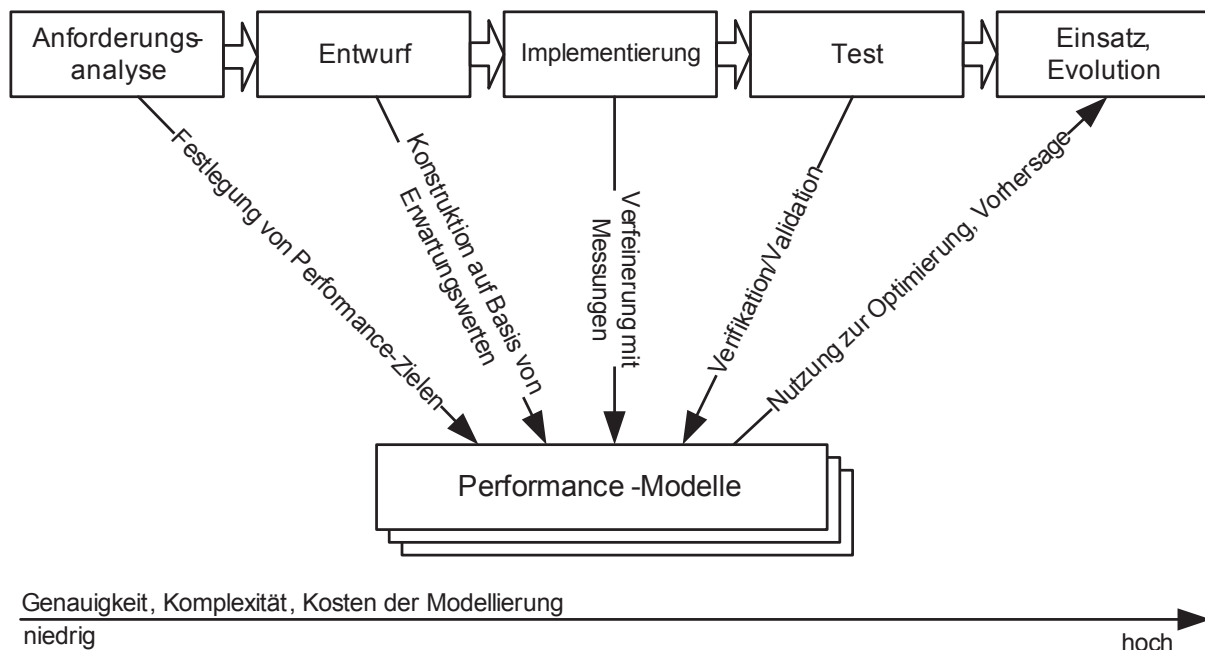


Abbildung 2.1.: Performance Engineering im Software-Lebenszyklus

Der anschließende Systemtest, bei dem die einzelnen Komponenten des Systems zusammengesetzt werden, dient nicht nur zur Überprüfung der Korrektheit der Implementierung, sondern auch zur Messung der Performance des Gesamtsystems. Dabei kommen Software-Werkzeuge wie Hardware/Software-Monitore ¹ und Profiler ² zum Einsatz oder es werden Benchmarks ³ durchgeführt. Die Performance-Modelle der Architektur werden in dieser Phase der Entwicklung verifiziert und validiert, d.h. es wird überprüft, ob die Modelle korrekt sind und tatsächlich die Performance-Eigenschaften des Systems abbilden. Da zu diesem Zeitpunkt meist noch keine echten Benutzereingaben vorliegen, wird das System oftmals mit Hilfe so genannter Lasttreiber (engl. Load Driver) getestet, die das Nutzerverhalten simulieren.

Wird das System in seiner eigentlichen Arbeitsumgebung zum Einsatz gebracht, so kann es zunächst in einer Pilotphase unter realen Bedingungen (bzgl. Hardware-Umgebung, Benutzereingaben) getestet werden. Die Performance-Modelle können nun genutzt werden, um weitere Performance-Optimierungen an dem System zu testen. Da sich das Nutzungsprofil des Systems in den allermeisten Fällen im Verlauf der Zeit ändert, dienen die Performance-Modelle zusätzlich dazu, Vorhersagen für das zukünftige Verhalten des Systems zu treffen. Dabei kann z.B. analysiert werden, wie sich die Performance verändert, wenn sich die Benutzerzahl innerhalb eines

¹Hardware-Monitore können Ereignisse innerhalb eines Computersystems (z.B. das Setzen eines Registers) mittels vordefinierter Signale aufzeichnen. Software-Monitore bestehen aus Programmroutinen die in Software eingebettet werden, um Ereignisse (z.B. Festplattenzugriffe) aufzuzeichnen. Ein bekanntes Beispiel ist der Windows XP Performance Monitor.

²Ein Profiler ist ein Programm, das die Berechnungsdauer und Aufruffrequenz der Funktionen eines anderen Programms aufzeichnen kann. Ziel ist es, die Teile des Programms zu identifizieren und optimieren, in denen die meiste Zeit verbracht wird. Für .NET-Applikationen gibt es z.B. den ANTS-Profiler [rgS04]

³Benchmarks sind standardisierte Testverfahren, mit denen die Leistung von Computersystemen verglichen werden kann. Der SPEC-Benchmark [Cor04] ist ein häufig zitiertes Beispiel.

Monats verdoppelt. Die Modelle lassen dann schnell die Flaschenhalse in der Architektur erkennen und ermöglichen die gezielte und effektive Optimierung des Systems. Das im übernächsten Abschnitt näher erläuterte *Capacity Planning* zielt genau auf diesen Anwendungsfall hin.

Die Performance-Modelle werden also in jeder Phase des Software-Lebenszyklus verfeinert und ständig verifiziert. Die Genauigkeit, Komplexität und die Kosten der Modelle korrelieren dabei mit dem Fortschritt der Entwicklung des Systems. Werden die oben genannten Phasen in mehreren Iterationen durchlaufen, können die Performance-Modelle in jeder dieser Iterationen verbessert werden. Das Vorgehen im Performance Engineering eignet sich somit auch gut für eine evolutionäre Software-Entwicklung.

Im Performance Engineering werden eine Reihe existierender Methoden zu einem Gesamtkonzept verbunden (Abbildung 2.2). Einige dieser Methoden werden in den folgenden Unterkapiteln näher erläutert. Im Fokus dieser Ausarbeitung stehen Konzepte zur Performance-Modellierung und Methoden des Capacity Plannings.

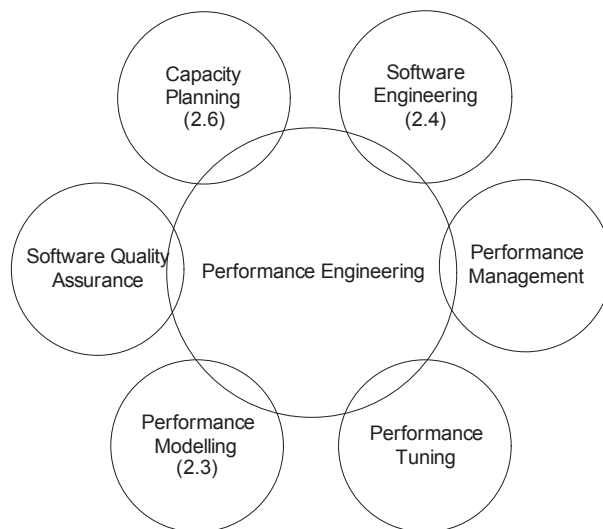


Abbildung 2.2.: Teilbereiche des Performance Engineering [SS01]

2.2. Ursprünge und Geschichte des Performance Engineering

Performance-Analysen wurden bereits in den Anfängen der Informatik auf den ersten Rechnersystemen durchgeführt. Die damals noch langsamen Prozessoren und die geringen Speicherkapazitäten zwangen Programmierer dazu, Techniken zu entwickeln, um eine Berechnung ihrer Algorithmen überhaupt zu ermöglichen. Knuth beschäftigte sich Anfang der 70er-Jahre ausgiebig mit der Effizienzanalyse von Sortier- und Suchalgorithmen [Knu73].

Die Grundlagen für die Performance-Modellierung ganzer Computersysteme wurden schon früh gelegt. Markow entwickelte Anfang des 20. Jahrhunderts eine Theorie zur Modellierung der stochastischen Charakteristiken von Ankunfts- und Bearbeitungsprozessen. Erlang beschäftigte sich etwa zur gleichen Zeit als erster mit der Warteschlangentheorie. Petri stellte 1962 ein wichtiges Modell zur Abbildung von dynamischen Systemeigenschaften vor, das Petri-Netz [Pet62]. 1971 schlug Buzen vor, Warteschlangenmodelle zur Performance-Analyse von Rechnersystemen

zu nutzen und veröffentlichte effiziente Algorithmen zur Lösung einiger dieser Modelle [Buz71]. Baskett et. al. definierten 1975 eine Klasse von Warteschlangenmodellen, die effizient analytisch lösbar sind [BCMP75]. Mit der Mean-Value-Analysis wurde ein effizienter Algorithmus vorgestellt, mit dem eine Reihe von Warteschlangenmodellen rekursiv auswertbar sind [LR80].

Nach den Fortschritten auf dem Gebiet der System-Modellierung begann man in den 70er-Jahren auch Modelle für die Ausführung von Software zu entwickeln. Diese Modelle können Schätzungen für Antwortzeiten liefern und Performance-Probleme identifizieren, jedoch nicht die Konkurrenz mehrerer Benutzeranfragen um begrenzte Systemressourcen abbilden. Damit sind die Ergebnisse solcher Modelle weniger präzise.

Anfang der 80er-Jahre schließlich verband Smith Software- und Systemmodelle für eine genauere Performance-Analyse und wies auf die Notwendigkeit hin, die Performance während des gesamten Lebenszyklus einer Software aktiv zu berücksichtigen [Smi80, Smi81]. Ihr Ansatz wurde als "Software Performance Engineering" (SPE) bekannt. In der Folgezeit wurden erste Software-Werkzeuge zur Performance-Modellierung entwickelt, so dass erstmals größere Software-Systeme kosteneffizient schon während früher Entwurfsphasen analysiert werden konnten.

In den 80er- und 90er-Jahren wurden verschiedenste Notationen zur Performance-Modellierung vorgeschlagen und das Performance-Engineering an unterschiedlichen Systemen (Verteilte Systeme, Web-Applikationen, Eingebettete Realzeitsysteme) demonstriert. Auch neue Werkzeuge wurden vorgestellt, jedoch kann von einer breiten Akzeptanz früher Performance-Analysen in der Software-Entwicklung noch nicht gesprochen werden.

1997 erschien mit der UML [OMG03b] schließlich eine standardisierte Modellierungssprache für objekt-orientierte Software-Systeme, die im Folgenden auch als Ausgangsbasis für Performance-Analyse von Entwürfen genutzt wurde. Seit 2002 existieren standardisierte Erweiterung für die UML, um Performance-Metriken in UML-Diagramme aufzunehmen, näheres dazu folgt in Abschnitt 2.4.

Ende der 90er-Jahre stellte Smith allgemeine Performance-Prinzipien und, angelehnt an die Entwurfsmuster nach Gamma et. al. [GHJV95], auch Software Performance Patterns und Antipatterns vor [Smi02]. Diese sind auf einer höheren Abstraktionsebene als die bekannten Design-Patterns angesiedelt und eignen sich daher für jegliche Art von System und Programmierparadigma.

2.3. Performance Modellierung

2.3.1. Markow-Ketten

Die meisten Notationen zur Performance-Modellierung basieren auf *Markow-Ketten*. Diese können vereinfacht als endliche Automaten mit Übergangswahrscheinlichkeiten an den Transitionen beschrieben werden. Markow-Ketten sind formal definiert als diskrete Stochastische Prozesse, für die die Einschränkung gilt, dass die Wahrscheinlichkeit in einen Folgezustand überzugehen nur vom Ursprungszustand abhängt.

Für ein Computersystem kann jeder Zustand, in dem sich das System abhängig von der Nutzerzahl und den verfügbaren Ressourcen befindet, durch einen Zustand in einer Markow-Kette modelliert werden. Beispielsweise kann sich ein einfacher Rechner bestehend aus einer CPU und zwei Festplatten bei zwei Benutzern in sechs verschiedenen Zuständen befinden, wenn

von einem Benutzer immer abwechselnd auf CPU und Festplatte zugegriffen wird (Abbildung 2.3). Ist ein Zustandsdiagramm zur Modellierung der Systemzustände erstellt, müssen noch die Übergangswahrscheinlichkeiten an den Transitionen zwischen den Zuständen ergänzt werden und die Zeit angegeben werden, die in jedem Zustand verbracht wird.

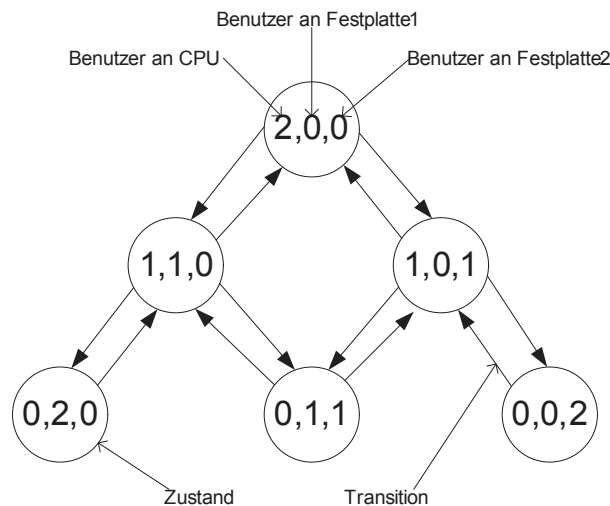


Abbildung 2.3.: Markow-Kette eines einfachen Rechners [MAD04, p.268]

Dieses System kann durch die Lösung linearer Gleichungssystem ausgewertet werden, wobei Performance-Metriken wie die Antwortzeit des Systems oder die Auslastung einzelner Ressourcen geliefert werden können. Diese Modelle können nicht nur zur formalisierten Beschreibung der Performance-Charakteristiken eines Computersystems genutzt werden, sondern auch um Vorhersagen über das System zu treffen, wenn z.B. eine schnellere CPU eingebaut wird oder einzelne Ressourcen verdoppelt werden.

Problem der Markow-Ketten ist, dass die Anzahl der Zustände exponentiell im Verhältnis zur Anzahl der Benutzer und Ressourcen im System ansteigt. In einem mittleren Büronetzwerk mit 25 Benutzern und 25 Rechnern beträgt die Anzahl der zu modellierenden Zustände bereits mehr als 63 Milliarden. Die zugehörigen linearen Gleichungssysteme, die zur Analyse des Systems ausgewertet werden müssen, können auch von modernen Rechnern nicht mehr gelöst werden. Zur Auswertung von Markow-Ketten wurde deshalb eine Technik namens "Mean-Value-Analysis" (MVA) [LR80] entwickelt. Es handelt sich um einen eleganten, rekursiven Algorithmus, der zu einer Benutzerzahl n die mittleren Performance-Metriken berechnen kann, wenn diese für $n-1$ Benutzer bekannt sind. Initialisiert man den Algorithmus mit einer Benutzerzahl von Null, so können leicht die Performance-Metriken für beliebige Benutzerzahlen iterativ berechnet werden, wobei der Berechnungsaufwand vernachlässigbar ist. Der MVA-Algorithmus lässt sich auf eine große Zahl von Performance-Modellen anwenden und kommt in fast allen Software-Werkzeugen zur analytischen Performance-Evaluation zum Einsatz. Menasce et. al. [MAD04, p.332] sehen den MVA-Algorithmus als einen der wichtigsten Beiträge zum Forschungsgebiet der Performance-Modellierung in den letzten 25 Jahren an.

2.3.2. Warteschlangenmodelle

Zur Modellierung der Performance von Computersystemen werden am häufigsten *Warteschlangenmodelle* (engl. Queueing Networks, QN) verwendet [BMIS04, p.303], welche auf Markow-Ketten basieren. Ressourcen können in der Regel nur eine Anfrage gleichzeitig bearbeiten. Treffen mehrere Anfragen auf eine Ressource, so bildet sich vor ihr eine Warteschlange (Abbildung 2.4). Die Anfragen werden dann in einer bestimmten Reihenfolge (Warteschlangendisziplin) bearbeitet, normalerweise in der Reihenfolge, in der sie ankommen sind (First Come First Serve, FCFS).

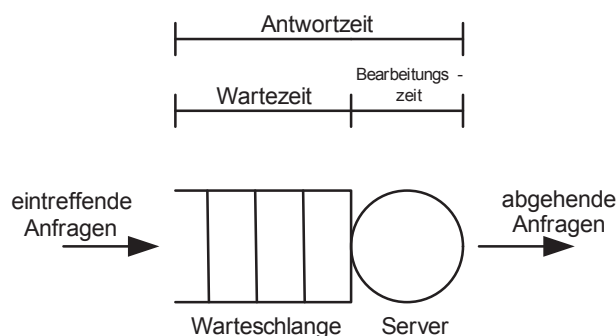


Abbildung 2.4.: Einfaches Warteschlangenmodell [MAD04, p.292]

Einige grundlegende Gesetzmäßigkeiten zwischen den verschiedenen Performance-Metriken in einem solchen Warteschlangenmodell konnten hergeleitet werden. Beispielsweise gibt das Auslastungsgesetz an, dass die Auslastung eines Servers gleich dem Produkt der mittleren Bearbeitungszeit einer Anfrage und dem Durchsatz des Servers ist. Das Bearbeitungszeitgesetz besagt, dass die mittlere Bearbeitungszeit auf einem Server gleich dem Quotienten seiner Auslastung und dem Gesamtdurchsatz des Systems ist. Am bekanntesten jedoch ist das Gesetz von Little [Lit61]. Es sagt aus, dass die mittlere Anzahl von Anfragen in einem Warteschlangenmodell gleich dem Produkt der Antwortzeit und des Durchsatzes des Servers ist ($L = \lambda W$). Das bedeutet z.B. dass man die Antwortzeit (=Wartezeit+Bearbeitungszeit) eines Servers ausrechnen kann, wenn man die Länge der Warteschlange und den Durchsatz des Servers kennt. Bemerkenswert ist, dass Littles Gesetz unabhängig von der benutzten Wahrscheinlichkeitsverteilung ist und somit keine Annahmen über die Verteilung der Ankunftszeiten oder die Warteschlangendisziplin gemacht werden müssen.

Aufbauend auf diesen Gesetzmäßigkeiten konnten eine Reihe von analytischen Ergebnissen für Warteschlangenmodelle abgeleitet werden. Ist die Ankunftsrate von Anfragen an eine Warteschlange generisch (engl. "generic", G), d.h. also beliebig verteilt, die Bearbeitungsdauer ebenfalls generisch verteilt und gibt es nur einen Server, dann liegt eine so genannte G/G/1-Warteschlange vor. Für diese Warteschlange kann beispielsweise die Auslastung des Servers berechnet werden. Mit der Einschränkung, dass die Ankunftsrate der Anfragen exponential (engl. auch "memoryless", M) verteilt ist (eine M/G/1-Warteschlange), können weitere Ergebnisse, wie die Wartezeit, die Antwortzeit und die Warteschlangenlänge bestimmt werden. Eine große Anzahl von Ergebnissen für andere Varianten (z.B. M/M/1, M/G/n, G/G/n usw.) sind vorhanden. Teilweise können keine exakten Lösungen berechnet werden, es existieren in diesen Fällen

jedoch Näherungslösungen. So können weitere Aussagen über Server mit Ruhezuständen, Systeme mit mehreren Warteschlangen an einem Server und verschiedenen Bearbeitungsreihenfolgen oder Systeme mit einer Warteschlange und mehreren Servern gemacht werden.

Einzelne Server mit Warteschlangen können zu Warteschlangennetzwerken verbunden werden, um ganze Rechnersysteme mit Komponenten wie Prozessoren, Festplatten, Netzwerkanbindungen zu modellieren (Abbildung 2.5).

Die Benutzer, die mit solchen Systemen interagieren, sind in ihrem Verhalten und ihrer Nutzung des Systems meist sehr unterschiedlich. Einige Benutzer sind beispielsweise schon erfahren im Umgang mit dem System und schöpfen alle vorhandenen Möglichkeiten aus. Andere Benutzer sind noch Neulinge und stellen weniger bzw. einfachere Anfragen. Würde man die unterschiedlichen Benutzer in einem System durch das mittlere Nutzungsverhalten charakterisieren, so ergäben sich ungenaue Performance-Vorhersagen. Daher sind für Warteschlangenmodelle Techniken entwickelt worden, um mehrere unterschiedliche *Benutzerklassen* zu modellieren.

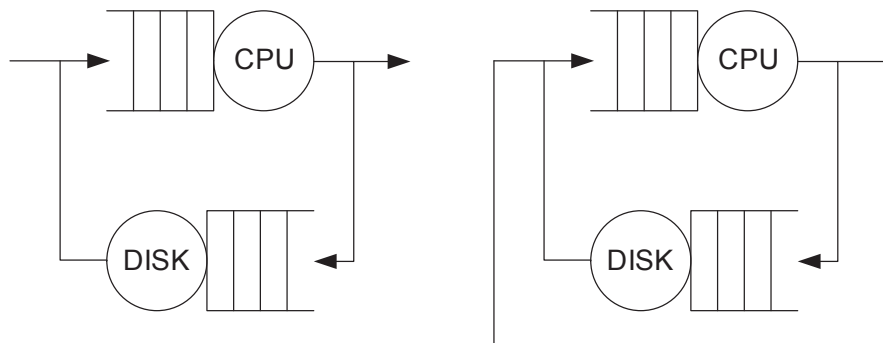
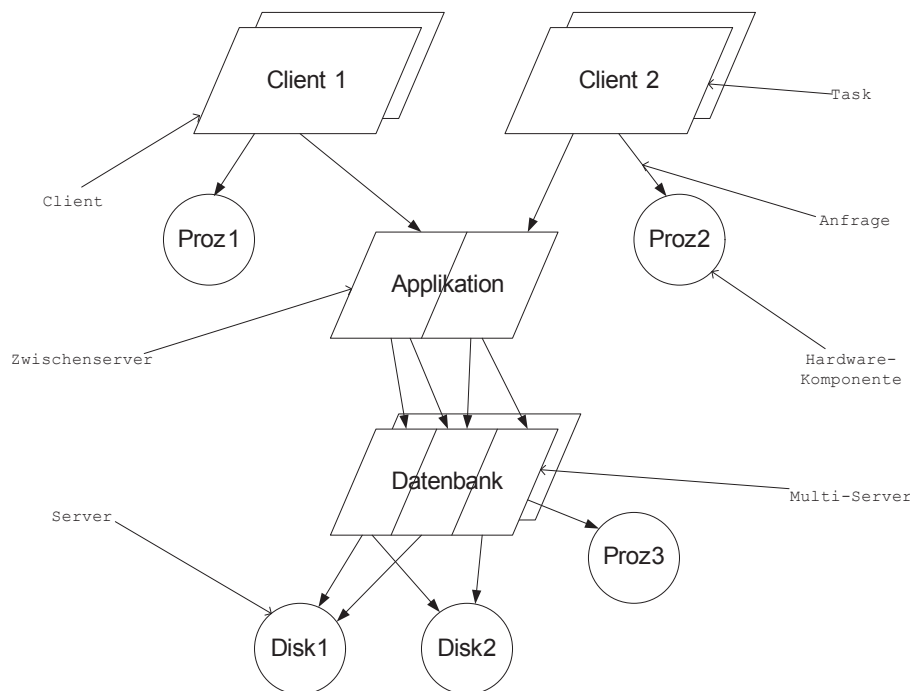


Abbildung 2.5.: Offenes und geschlossenes Warteschlangennetzwerk

Man unterscheidet darüber hinaus zwischen *offenen* und *geschlossenen* Benutzerklassen. Bei offenen Benutzerklassen kommen ständig neue Anfragen von außen an das System, die es nach ihrer Bearbeitung wieder verlassen. Die Anzahl der Anfragen ist unbegrenzt, für die Modellierung wird die mittlere Zeit zwischen zwei Anfragen angegeben. Damit können z.B. die Benutzer eines Webservers oder einer Datenbank charakterisiert werden. Geschlossene Warteschlangenmodelle haben eine feste Anzahl von Benutzern, die das System nach ihrer Bearbeitung nicht verlassen, sondern nach einer Wartezeit (engl. "think time") wieder in das System eintreten. Damit können z.B. Batch Jobs, die sich periodisch wiederholen (etwa die tägliche Generierung von Berichten aus einer Datenbank) oder interaktive Anfragen, bei denen der Benutzer im Dialog mit dem Server arbeitet, modelliert werden. Auch gemischte Benutzerklassen sind möglich, d.h. offene und geschlossene Benutzerklassen innerhalb eines Systems.

Weitere Formeln existieren, um z.B. lastabhängige Server in ein Warteschlangenmodell einzubeziehen. Solche Server verringern ihre Bearbeitungsgeschwindigkeit je mehr Anfragen an sie gestellt werden bzw. je höher sie belastet werden. Beispielsweise können mit solchen Servern Festplatten genauer modelliert werden, da diese bei einer großen Anzahl von Anfragen ihren Lese/Schreibkopf häufiger hin und her bewegen müssen, was zu Performance-Einbußen führt. Auch Netzwerkadapter sind in der Regel lastabhängig, da bei einer großen Anzahl von Anfragen auch ein immer größerer Overhead für die Bestätigung und den Versand von Netzwerkpaketen entsteht.



denen der Absender nicht blockiert wird und der aufgerufene Server nicht antwortet. Mit LQNs können Performance-Metriken wie Antwortzeit, Durchsatz, Auslastung und Wartezeiten durch Analyse oder Simulation bestimmt werden.

2.3.3. Stochastische Prozessalgebren

Als weitere Notation zur Performance-Modellierung werden von einigen Verfahren Stochastische Prozessalgebren [HH95, HHK02] verwendet. Diese eignen sich besonders gut zur Darstellung nebenläufiger Systeme. Sie stellen Systeme als eine Menge von Prozessen dar, die atomare Aktionen ausführen. Prozesse können mittels einer Menge von Operatoren zusammengesetzt werden, das System kann auf mehreren Abstraktionsebenen beschrieben werden. Vorteil dieser Notation ist die Möglichkeit zur formalen Verifikation der Spezifikation.

Um auch nicht-funktionale Eigenschaften wie die Performance darzustellen, gibt es verschiedene stochastische Erweiterungen für Prozessalgebren. Bekannte stochastische Prozessalgebren sind TIPP (Time Processes and Performability evaluation) [HUM00], EMPA (Extended Markovian Process Algebra) [BG98, BCD00] und PEPA (Performance Evaluation Process Algebra) [Hil93, GH94], für die auch entsprechende Tools verfügbar sind (TIPPtool, PEPA Workbench und 'The Two Towers' für EMPA). Bei diesen Notationen werden die Aktionen der Prozess-Algebren mit exponential verteilten Zufallsvariablen assoziiert, woraus dann Markow-Ketten generiert werden können. Meistens werden die funktionalen und nicht-funktionalen Anforderungen an ein System mit Stochastischen Prozessalgebren in einem einzigen Modell dargestellt.

Da Stochastische Prozessalgebren den meisten Entwicklern in der Praxis nicht geläufig sind, arbeitet man an Verfahren, um diese Modelle automatisch aus UML-Diagrammen zu erstellen [Poo99]. Von Balsamo et. al. [BBS02] stammt zusätzlich eine auf Stochastischen Prozessalgebren basierende Architekturbeschreibungssprache namens AEmilia, deren Semantik in der EMPA Spezifikation ausgedrückt wird.

2.3.4. Stochastische Petri-Netze

Auch die aus der Automatentheorie bekannten *Petri-Netze* können für die Erstellung von Performance-Modellen verwendet werden. Petri-Netze sind eine mathematische Repräsentation von verteilten Systemen und wurden Anfang der 60er-Jahre von C. A. Petri definiert [Pet62]. Ein Petri-Netz besteht aus einer Menge von Systemzuständen (Stellen), einer Menge von Übergängen (Transitionen), einer Eingabefunktion zur Verbindung von Stellen mit Transitionen und einer Ausgabefunktion zur Verbindung von Transitionen mit Stellen. Zusätzlich gibt es eine Markierungsfunktion, die jeder Stelle eine nicht-negative Zahl zuweist. Das dynamische Verhalten dieses Systems wird durch eine Menge von Transitionsaktivierungen festgelegt. Näheres zu Petri-Netzen findet sich beispielsweise in [PW02b].

Aufbauend auf der Basisversion von Petri-Netzen können Übergangswahrscheinlichkeiten und exponential verteilte Aktivierungszeiten an Stellen und Transitionen hinzugefügt werden, womit sich die Netze dann als so genannte *stochastische Petri-Netze* (SPN) zur Performance-Analyse eignen [Mol82, Mar90]. Dazu werden SPNs auf zeitkontinuierliche Markow-Ketten abgebildet. Performance-Metriken wie Durchsatz, Auslastung und Antwortzeiten können dann mit numerischen Methoden oder durch Simulation bestimmt werden.

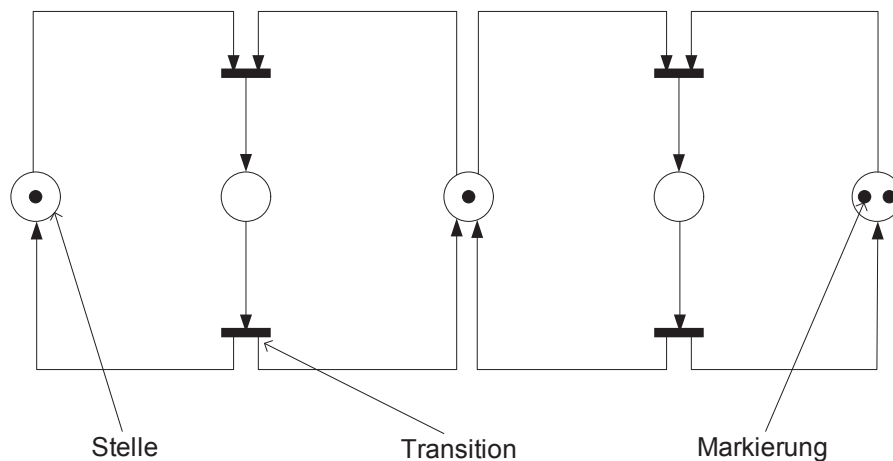


Abbildung 2.7.: Petri-Netz [CS03]

Problem dieser Notation ist ähnlich wie bei den Warteschlangenmodellen, dass es bei nicht-trivialen Modellierungen schnell zu einer exponentiell ansteigenden Anzahl von Zuständen (engl. "state-space explosion") kommen kann, so dass diese nicht mehr in linearer Zeit ausgewertet werden können. Zur Lösung dieses Problems wurden *generalisierte stochastische Petri-Netze* (GSPN) vorgeschlagen, in denen Transitionen auch ohne Zeitverzug schalten können, und somit keinen Einfluss auf die Komplexität der Lösung haben [MBG84].

Genau wie bei den Stochastischen Prozessalgebren vereinen die meisten Ansätze zur Performance-Analyse mit Petri-Netzen die funktionalen und nicht-funktionalen Eigenschaften einer Software innerhalb eines einzigen Modells. Es existiert eine Reihe von Frameworks zur Spezifikation und quantitativen Auswertung von Stochastischen Petri-Netzen (z.B. GreatSPN, HiQPN, DSPNExpress 2000, eine Liste findet sich in [MR04]).

In der letzten Zeit wurden Versuche unternommen, einzelne Diagramm-Typen der UML automatisch in Stochastische Petri-Netze zu transformieren. King et. al. [KP00] stellen Überlegungen an, wie ein Stochastisches Petri-Netz aus Anwendungsfalldiagrammen und kombinierten Kollaborations- und Zustandsdiagrammen erstellt werden kann. Bernardi et. al. [BDM02] leiten generalisierte Stochastische Petri-Netze aus Zustands- und Sequenzdiagrammen ab.

2.4. Software-Modellierung

In der Softwaretechnik gibt es eine Reihe von Modellen, um die Architektur einer Software vor der Implementierung zu beschreiben. Im Performance Engineering können solche Entwürfe von Software dazu genutzt werden, die Leistung eines Software-Systems vorherzusagen, bevor das System tatsächlich ausprogrammiert wird. Für Performance-Analysen sind dabei solche Modelle relevant, die den Kontrollfluss durch eine Architektur abbilden und deren dynamische Eigenschaften darstellen. Statische Modelle, die die Struktur eines Systems abbilden, sind nur dann brauchbar, wenn sie Informationen über die Verteilung der einzelnen Aktivitäten auf verschiedene Ressourcen enthalten.

Für die Modellierung objektorientierter Softwaresysteme hat die 1997 spezifizierte *Unified Modelling Language* (UML) [OMG03b] weite Verbreitung gefunden und sich de-facto als ein

Standard etabliert. Diese Modellierungssprache erlaubt verschiedene Sichten auf ein System und enthält in der Version 2.0 insgesamt 13 unterschiedliche Diagrammtypen. Zur Spezifikation des dynamischen Verhaltens einer Software-Architektur werden daraus im Performance Engineering am häufigsten die Sequenz-, Aktivitäts- und Zustandsdiagramme verwendet. Vorteile der UML sind die Flexibilität und Einfachheit der Handhabung, weshalb sie bei Entwicklern beliebt ist. Nachteil ist das Fehlen einer exakt standardisierten Semantik, denn die UML ist nur eine semi-formale Notation und kann nicht formal verifiziert werden. Trotzdem bauen die meisten aktuellen Performance-Analyseverfahren auf der UML zur Spezifikation der Software auf. Näheres über diese Modellierungssprache findet sich z.B. in [Fow03].

Da die UML an sich nicht für die Modellierung von Performance-Eigenschaften gedacht ist, musste ein Weg gefunden werden Leistungsdaten wie Antwortzeiten oder Ressourcennutzung in die Diagramme zu integrieren. Nachdem zunächst einige Forscher eigene Ansätze dazu entwickelten, wurde 2002 das *UML Profile for Schedulability, Performance and Time* [OMG03a] (im Folgenden als UML Performance Profile bezeichnet) von der Object Management Group als Standard spezifiziert. Es erlaubt die Beschriftung von UML-Elementen mit Performance-bezogenen Informationen auf Basis der Standarderweiterungsmechanismen der UML (Stereotypes, Constraints, Tagged Values) und stellt so eine standardisierte Schnittstelle zwischen Software-Modell und Performance-Modell dar. Beispielsweise kann für einzelne Schritte in einem Sequenzdiagramm eine Bearbeitungsdauer festgelegt werden. Durch die einheitliche Syntax soll es möglich sein, dass UML-Modelle von unterschiedlichen Performance-Analysewerkzeugen ausgewertet werden können.

Neben der UML sind weitere semi-formale Notationen für Softwaresysteme für die Vorhersagen von Performance-Eigenschaften benutzt worden. *Message Sequence Charts* (MSC) [Sta99] ähneln von ihrer Struktur her den Sequenzdiagrammen aus der UML und können die Interaktion zwischen mehreren Objekten darstellen. Sie enthalten darüber hinaus Konstrukte zur Modellierung von Schleifen oder Verzweigungen im Kontrollfluss und sind somit mächtiger als Sequenzdiagramme. Anwendung finden diese Diagramme z.B. beim Verfahren von Smith und Williams [Smi02], das in dieser Fallstudie noch näher untersucht wird. *Use Case Maps* (UCM) [RB96] modellieren Softwaresysteme auf einer hohen Abstraktionsebene und zeichnen den Kontrollfluss durch die Komponenten des Systems auf. Die Komponenten werden als "Blackbox" dargestellt, über ihr internes Verhalten ist somit nichts bekannt, daher ist die Modellierung recht grobgranular. Petriu et. al. stellen beispielsweise ein Verfahren vor um UCMs automatisch in geschichtete Warteschlangenmodelle zu transformieren, mit denen dann Performance-Analysen durchgeführt werden können [PW02a].

2.5. Software Performance Engineering

Das *Software Performance Engineering* (SPE) geht vor allem auf die Arbeiten von Connie U. Smith zurück, die bereits Anfang der 80er Jahre diesen Begriff prägte [Smi81]. SPE kann als eine Sammlung von Methoden, Prozessen und Werkzeugen verstanden werden, um die Performance einer Software-Architektur schon zu einem frühen Entwicklungszeitpunkt vorherzusagen. Ziel ist es, die bei vielen Entwicklern vorherrschende "fix-it-later"-Einstellung in Bezug auf Performance zu überwinden und die Performance von Softwaresystemen zu einem integralen Bestandteil ihrer

Entwicklung zu machen.

Definition (Software Performance Engineering): SPE is a systematic, quantitative approach to constructing software systems that meet performance objectives. SPE is an engineering approach to performance, avoiding the extremes of performance-driven development and 'fix-it-later'. SPE uses model predictions to evaluate trade-offs in software functions, hardware size, quality of results, and resource requirements [Smi02].

Performance-Analysen sollen so früh wie möglich durchgeführt werden, auch dann wenn noch kein Programmcode vorliegt. Bei der Konstruktion von Performance-Modellen kann in diesem Fall noch nicht auf Messungen zurückgegriffen werden, sondern es müssen Schätzungen angestellt werden oder Erfahrungen aus ähnlichen Projekten eingebracht werden. Dabei ist es nicht das Ziel die absolute Performance einer Software genauestens vorherzusagen. Vielmehr sollen durch das SPE verschiedene Entwurfsalternativen bewertet werden und deren relative Unterschiede bezüglich der Performance analysiert werden. Entwürfe, die eine gute Performance erwarten lassen, sollen von solchen Entwürfen unterschieden werden, bei denen die Implementierungen später wahrscheinlich Performance-Probleme haben werden [MAD04]. Schreitet die Entwicklung weiter voran, können die Schätzungen durch Messungen an erste Codefragmenten ersetzt und die Performance-Vorhersagen für das Gesamtsystem verfeinert werden.

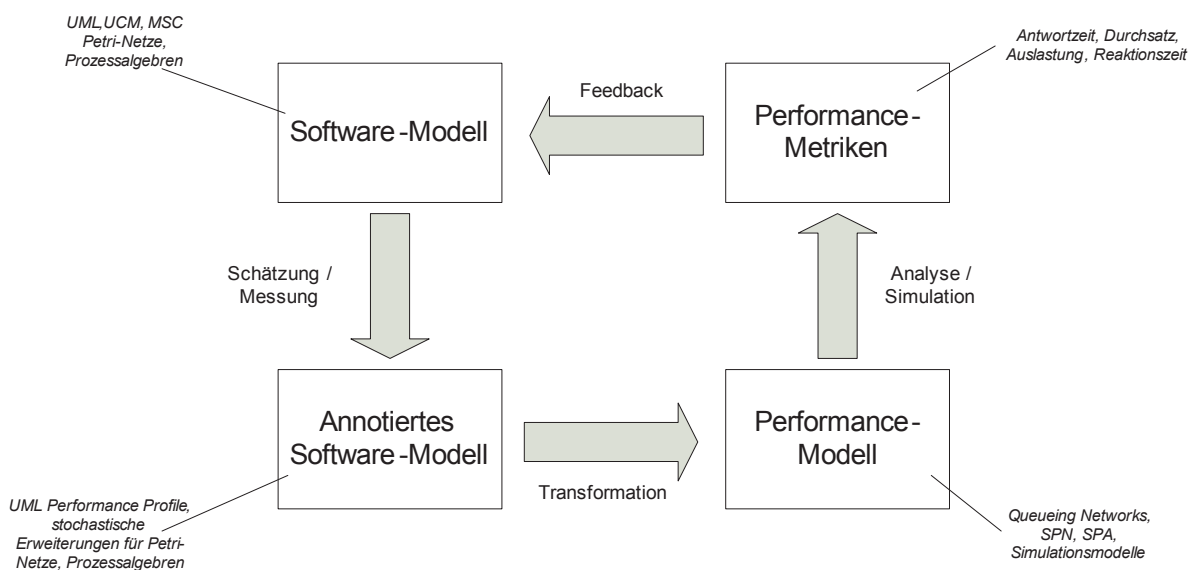


Abbildung 2.8.: Generischer Prozess der Performance-Analyse

Seit den Pionier-Arbeiten von Smith wurden eine Reihe von Verfahren für das SPE mit unterschiedlichen Notationen entwickelt, die jedoch alle einem ähnlichen Vorgehensmodell folgen (Abbildung 2.8). Ausgangspunkt der Vorhersage von Performance-Attributen ist ein Modell der Software-Architektur, welches bei der Software-Entwicklung üblicherweise während der frühen Phasen (Anforderungsdefinition, Entwurf) entsteht. Die dabei benutzten Notationen wie UML, UCMs, MSCs oder Petri-Netze wurden bereits in den vorherigen Abschnitten vorgestellt. Das Software-Modell wird anschließend um Informationen über Performance-Eigenschaften ergänzt.

Dabei werden z.B. einzelne Berechnungsabschnitte der Software mit Bearbeitungszeiten versehen, die auf Schätzungen basieren oder schon gemessen wurden.

Aus dem annotierten Software-Modell wird dann ein Performance-Modell (z.B. ein Warteschlangenmodell) erstellt. Die Transformation des Software-Modells in ein Performance-Modell kann bei einigen Verfahren werkzeuggestützt erfolgen. Idealerweise kapselt ein Verfahren die Erstellung des Performance-Modells vollständig vom Entwickler ab, so dass sich dieser lediglich um die Entwicklung des ihm eher vertrauten Software-Modells (z.B. in UML) kümmern muss. In diesem Fall können die Analysen auch ohne Beteiligung von Performance-Experten durchgeführt werden.

Ist das Performance-Modell vollständig erstellt und wurden alle Eingabeparameter spezifiziert, können die gewünschten Performance-Metriken wie Antwortzeiten oder Ressourcenauslastungen berechnet werden. Die Auswertung kann auf mehrere Arten geschehen:

- **Analytisch:** Die den Performance-Modellen zugrunde liegenden Markow-Ketten können durch Verfahren wie z.B. die Mean-Value-Analysis ausgewertet werden. Die Ergebnisse liegen dabei in Sekundenschnelle vor, und es können leicht verschiedene Alternativen untersucht werden. Bei analytischen Verfahren müssen allerdings in den meisten Fällen einschränkende Annahmen über das Performance-Modell gemacht werden. Zum Beispiel muss die Ankunftsrate von Anfragen an ein System exponential verteilt sein, oder es ist keine simultane Ressourcennutzung möglich. Bei großen Systemen kann es zudem vorkommen, dass keine exakten Lösungen mehr berechnet werden können und mit Approximationsverfahren gearbeitet werden muss.
- **Simulativ:** Einige Verfahren verwenden zur Ableitung von Performance-Ergebnissen Simulationen [BINN99]. Dabei werden die Ausführungseigenschaften des Performance-Modells durch ein Programm emuliert. Vorteil von Simulationen ist, dass Software-Architekturen auf einem beliebig hohen Abstraktionsgrad ausgewertet werden können. Simulationen unterliegen keinen Restriktionen bzgl. der Mächtigkeit der Modellierung. Allerdings kann die Ausführung einer Simulation hohe Anforderung an Rechenzeit und Speicher stellen, so dass es unter Umständen recht lange dauert bis Performance-Ergebnisse vorliegen. Die Ergebnisse einer Simulation erfordern eine statistische Auswertung und können oftmals nur von Experten analysiert werden [Mar04].
- **Hybrid:** Die Berechnung von Performance-Werten erfolgt durch eine Mischform von Analyse und Simulation.

Nachdem die Performance-Metriken berechnet wurden, sollten diese Werte idealerweise wieder in das ursprüngliche Software-Modell zurückgemeldet werden. Die Entwickler können dann aufgrund der Performance-Daten sofort Flaschenhälse in ihrem Entwurf erkennen und verschiedene Alternativen ausprobieren. Ein solches Feedback von Leistungsdaten ist bisher aber nur bei wenigen Verfahren umgesetzt worden, da Software-Modell und Performance-Modell oft auf unterschiedlichen Abstraktionsebenen angesiedelt sind und eine Zuordnung der Daten zu Elementen im Software-Modell daher nur schwierig automatisierbar ist.

Die SPE-Community hat sich in einem seit 1998 jeweils alle 18 Monate abgehaltenen Workshop organisiert. Auf diesem "International Workshop on Software and Performance" der ACM

(WOSP) wurden bereits eine Reihe von Performance-Analyseverfahren im SPE-Kontext vorgeschlagen. Balsamo et. al. [BMDI04] haben einen ausführlichen und aktuellen Überblick über diese Verfahren erstellt. Dabei klassifizieren sie die Verfahren nach

- der Phase im Software-Entwicklungsprozess, in der das Verfahren angewendet werden kann (Anforderungsdefinition, Entwurf, usw.),
- nach dem Grad der Integration von Software-Modell und Performance-Modell (syntaktisch verwandt, semantisch verwandt, identisch) sowie
- nach dem Grad der Automatisierbarkeit (niedrig, mittel, hoch).

Vermisst wird in ihrem Klassifikationsschema eine Dimension für die Präzision und Qualität der Vorhersage der Performance-Ergebnisse eines Verfahren, wobei jedoch informell festgehalten werden kann, dass die Verfahren exaktere Ergebnisse liefern je weiter hinten sie im Entwicklungsprozess angesiedelt sind. In diesem Fall können die Verfahren schon mit Messungen an Teilen der Implementierung arbeiten und müssen nicht ausschließlich auf Schätzungen zurückgreifen.

Die für diese Fallstudie in Betracht gezogenen Performance-Analyseverfahren werden in Kapitel 4 vorgestellt.

2.6. Capacity Planning

Unter *Capacity Planning* (CP) versteht man die Modellierung und Vermessung meist schon bestehender Hardware-/Software-Systeme zur Vorhersage der Performance-Eigenschaften bei einem sich ändernden Nutzungsprofil. Ziel ist es, Flaschenhälse in Architekturen zu identifizieren, die dann kritisch werden, wenn sich die Belastung des Systems erhöht. Eine Definition des CP stammt von Menasce et. al.:

Definition (Capacity Planning): CP is the process of predicting when future load levels will saturate the system and determining the most cost-effective way of delaying system saturation as much as possible. The prediction must consider the evaluation of the workload, due to existing and new applications, and the desired service level [MAD94].

Das CP ist in der Industrie weit verbreitet und wird heute häufig für verteilte Systeme durchgeführt, die über eine hohe Nutzerzahl verfügen. Eine proaktive und andauernde Kapazitätsplanung ist für Systembetreiber äußerst wichtig, um Überlastungen des Systems zu vermeiden und die Verfügbarkeit jederzeit sicherzustellen. Kann ein System seine Dienste aufgrund von Überlastung durch eine steigende Benutzerzahl nicht mehr anbieten, drohen den Systembetreibern nicht nur finanzielle Einbußen und Kundenunzufriedenheit sondern auch Imageschäden, zumal solche Performance-Probleme meist nicht in kurzer Zeit zu beheben sind. Oftmals lassen sich Performance-Probleme wegen Überlastungen auch nicht einfach durch den Zukauf von Hardware lösen, wenn z.B. eine mangelhafte Systemarchitektur vorliegt.

Im CP wird die Performance-Modellierung meistens mit den klassischen Warteschlangenmodellen durchgeführt. Für diese müssen die Bearbeitungszeiten gemessen werden und ein Nutzungsprofil der Ressourcen erstellt werden. Dies erfordert schon fertige Systeme oder zumindest

Prototypen, an denen Messungen durchgeführt werden können. Ist ein Performance-Modell für eine bestehende Systemarchitektur erstellt worden, werden z.B. mit statistischen Methoden Vorhersagen über die zukünftige Nutzung des Systems getroffen. Mit diesen veränderten Nutzungsdaten werden dann die Performance-Modelle erneut ausgewertet. Dabei können meist leicht die begrenzenden Flaschenhälse in der Architektur identifiziert werden. Beispielsweise könnte eine Festplatte aufgrund der zu hohen Belastung nicht mehr alle Anfragen beantworten oder eine CPU mit den Berechnungen nicht mehr nachkommen.

Anders als beim SPE steht beim CP nicht die Entwicklung neuer Systeme im Mittelpunkt. Daher werden hier auch keine Entwurfsdokumente für Software verwendet, um Vorhersagen für das System zu treffen. Dieses Verfahren bewegt sich fast ausschließlich auf der Systemebene, d.h. im Zentrum der Beobachtungen steht die Hardware, während Aspekte der Software eher vernachlässigt werden. So gibt es bei diesem Verfahren auch keine Modellierung des Kontrollflusses innerhalb einer Ressource.

CP hat nicht zuletzt durch das Internet an Bedeutung gewonnen. Von Menasce et. al. liegen mehrere Bücher zum CP vor, in denen das Thema jeweils mit unterschiedlichen Schwerpunkten behandelt wird (Client/Server-Systeme [[MAD94](#)], E-Business [[MA00](#)], Web-Applikationen [[MA98](#), [MA02](#)]). Eine bessere Integration von SPE und CP ist angedacht [[Smi01](#)]. Der Prozess des CP nach Menasce et. al. wird in Kapitel 4 näher ausgeführt.

3. Methodik

3.1. Vorgehensweise

Forschung muss, insbesondere bei einer Fallstudie, zielgerichtet ablaufen. Yin [Yin94] warnt davor, dass bei einer Fallstudie in der Regel dermaßen viele Daten anfallen, dass ohne eine vorherige Auswahl von relevanten Variablen kaum eine Auswertung vorgenommen werden kann. Im Folgenden soll daher die *Goal, Question, Metric-Methode* [BCR94] (GQM) nach Basili und Rombach verwendet werden. Dabei werden die Forschungsfragen und Kriterien zu deren Beantwortung in einem Top-Down-Ansatz abgeleitet. Zunächst wird das umfassende Ziel (Goal) der Studie definiert und danach die zur Erreichung dieses Ziels notwendigen Fragestellungen (Question) formuliert. Anschließend werden Metriken (Metric) definiert, die zur Beantwortung dieser Fragen sinnvoll erscheinen. Am Ende der Studie können die gemessenen Daten anhand der Metriken zur Beantwortung der Fragestellung verwendet werden und die beantworteten Fragen die Zielsetzung der Studie erfüllen.

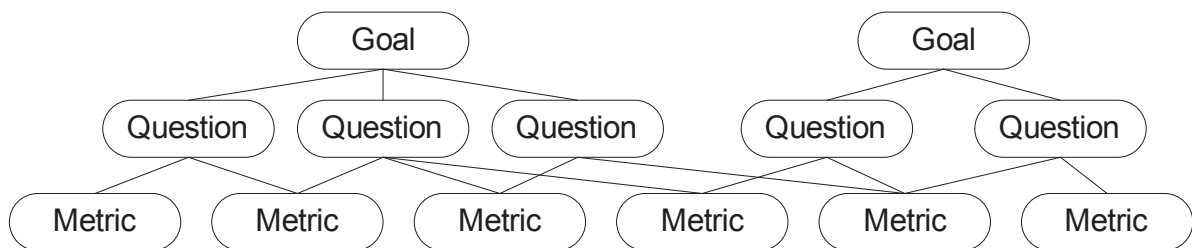


Abbildung 3.1.: Goal, Question, Metric Methode [BCR94]

Ein Ziel wird innerhalb von drei Dimensionen definiert (Sachverhalt, Objekt, Standpunkt) und dient einem bestimmten Zweck. Das Ziel dieser Studie wurde bereits in der Einleitung formuliert und lautet:

Empirische Bewertung (=Zweck) der Anwendbarkeit (=Sachverhalt) von Performance-Analyseverfahren für Software-Architekturen (=Objekt) aus der Sicht des Entwicklers (=Standpunkt).

Der Begriff "Empirische Bewertung" wird im Folgenden nach der Definition von Prechelt [Pre01] verwendet:

Definition (Empirische Bewertung): Empirische Bewertung steht in unserem Zusammenhang für die praktische Verwendung und Erprobung eines Werkzeugs, einer Methode oder eines Modells, um die tatsächlichen Eigenschaften dieser Artefakte zu verstehen und beschreiben zu können. Im Gegensatz dazu steht die spekulative

Bewertung, die auf Basis mehr oder weniger plausibler und größtenteils unausgesprochener Annahmen durch mehr oder weniger stringentes logisches Schließen die erwarteten Eigenschaften ohne Empirie herleitet.

Es folgen nun Ausführungen zu den aus dieser Zielsetzung definierten Fragestellungen.

3.2. Fragestellungen und Metriken

3.2.1. Validität der Vorhersagen

Die frühe Performance-Analyse von Software-Architekturen bereits während des Entwurfs erscheint den meisten Entwicklern zunächst als schwierig durchführbar. Viele vertreten die Ansicht, dass ohne eine Implementierung keine gültigen Vorhersagen für ein System auf Basis von Erwartungswerten oder Schätzungen gemacht werden können. Sie verfahren nach der bereits erwähnten "fix-it-later"-Mentalität und verlagern Performance-Analysen an das Ende der Entwicklung, wenn das fertige System mit Software-Werkzeugen ausgemessen werden kann. Interessant ist daher für solche Entwickler folgende Frage:

1. Wie hoch ist die Genauigkeit der Vorhersagen der Verfahren?

Zur Beantwortung der Frage müssen mehrere Metriken untersucht werden. Zunächst ist natürlich der Vergleich zwischen den Ergebnissen der Verfahren (etwa Antwortzeiten oder Ressourcenauslastungen) und den Messungen an einer Implementierung das augenscheinlichste Kriterium für die Genauigkeit der Verfahren (Metrik 3). Die Abweichung zwischen Vorhersage und Messung kann in Prozentzahlen angegeben werden.

Jedoch hängen die Ergebnisse als Ausgaben der Verfahren maßgeblich von den Eingaben der Verfahren ab. Dabei ist fraglich wie groß die Abweichung zwischen den Eingabedaten der Verfahren und den Ausgabedaten ist (Metrik 2). Eingabedaten für die Verfahren sind z.B. die Dauer bestimmter Einzelschritte im Kontrollfluss oder die Ressourcennutzung, die durch einzelne Aktion vorgenommen wird. Aufgrund der Heterogenität von Eingabe- und Ausgabedaten kann deren Abweichung nur schwierig in Zahlen ausgedrückt werden und soll daher auf einer groben Skala (hoch, mittel, niedrig) angegeben werden.

Eingabedaten für die Verfahren können auf verschiedene Weise bezogen werden, wenn noch keine Implementierung des Systems vorliegt. Es können Schätzungen angestellt werden, die auf Erfahrung mit früheren Projekten beruhen. Entwickler können kleinere Prototypen konstruieren oder dem zu entwickelnden System ähnliche Systeme untersuchen, um die Werte durch Messungen zu bestimmen. Die Werte können in Performance-Walkthroughs ermittelt oder durch die Angabe von oberen und unteren Schranken angenähert werden. In einem Experiment mit mehreren Teilnehmern kann untersucht werden, wie stark die Eingabedaten unterschiedlicher Teilnehmer voneinander abweichen, wenn diese von ihnen aufgrund gleicher Basisinformationen ermittelt werden (Metrik 1). Diese Abweichung kann z.B. durch die Spannweite der Dauer bestimmter Aktionen über alle Teilnehmer in Sekunden ausgedrückt werden. Dabei gilt es, den Teilnehmern eine solche Aufgabenstellung zu geben, die die jeweils normalen Methoden zur Ermittlung von Eingabedaten eines Verfahrens berücksichtigt.

Die Autoren der Verfahren zur frühen Performance-Analyse betonen oft, dass es nicht die Zielsetzung der Verfahren ist, genaue absolute Performance-Werte vorherzusagen. Im Vordergrund steht vielmehr die Bewertung unterschiedlicher Entwurfsalternativen aus Sicht der zu erwartenden Performance. Deshalb müssen in der Studie mehrere Entwurfsalternativen am Versuchsobjekt untersucht und die Versuchsteilnehmer um die Favorisierung einer Alternative gebeten werden. Die Performance der Alternativen muss dann auch an einer Implementierung gemessen werden. Danach kann prozentual angegeben werden, wie viele Versuchsteilnehmer die bzgl. der Performance beste Entwurfsalternative favorisiert haben (Metrik 4). Damit werden die Verfahren im Sinne ihrer Autoren verwendet. Sind die Vorhersagen der Teilnehmer so ungenau, dass jeweils unterschiedliche Alternativen gewählt werden, wären die Verfahren für normale Entwickler unbrauchbar und es müssten Performance-Experten hinzugezogen werden, die aufgrund ihrer Erfahrung zu genaueren Ergebnissen gelangen.

Die Metriken zur Beantwortung der ersten Frage finden sich zur besseren Übersicht in Tabelle 3.1 nochmals zusammengefasst. Diese Tabelle wird auch später für die Auswertung verwendet.

	Abweichung Eingabedaten verschiedener Teilnehmer (Metrik 1)	Abweichung Eingabedaten und Ausgabe- daten (Metrik 2)	Abweichung Ausgabedaten und Messdaten (Metrik 3)	Prozentsatz richtiger Entwurfent- scheidungen (Metrik 4)
Verfahren 1				
Verfahren 2				
...				
Verfahren n				

Tabelle 3.1.: Validität der Vorhersagen

3.2.2. Aufwand der Performance-Analyse

Ein Argument von Praktikern Performance-Analysen während des Entwurfsprozesses zu vernachlässigen ist nach wie vor der hohe Zeitaufwand, den die Anwendung dieser Verfahren mit sich bringt. Die zweite Frage soll daher diesen Sachverhalt näher untersuchen und lautet:

2. Wie hoch ist der Aufwand für die Performance-Analyse?

Performance-Analyseverfahren kommen in der Praxis nur dann zum Einsatz wenn der Aufwand für die Kosten gerechtfertigt ist. Ein Vergleich des Aufwands für die Performance-Analyse mit der eingesparten Zeit durch nachträgliche Entwurfsänderungen ist im Rahmen dieser Fallstudie leider nicht möglich. Dennoch können zumindest der Lernaufwand für die Verfahren (Metrik 5) und die Zeit für die Transformation einer Design-Spezifikation in ein Performance-Modell (Metrik 6) jeweils in Stunden gemessen werden.

3.2.3. Methodisches gegenüber intuitiven Vorgehen

Das Vorhandensein von Methoden zur Performance-Analyse im Design-Stadium bedeutet nicht automatisch, dass diese Methoden auch einen Vorteil gegenüber einem intuitiven Verfahren er-

bringen. Bewertungen von Entwurfsalternativen können auch von Personen vorgenommen werden, die sich noch nicht mit speziellen Performance-Analyseverfahren auseinander gesetzt haben. Sie können sich auf ihre Erfahrung berufen und versuchen mit einem unsystematischen Vorgehen eine Designentscheidung zu fällen. Auf diese Situation spielt die dritte Frage an:

3. Welche Vorteile bringen Performance-Analyseverfahren gegenüber einem intuitiven Vorgehen?

In der Studie wird daher mit einer ungeschulten Kontrollgruppe, die die gleiche Aufgabe bekommt, wie die eigentliche Versuchsgruppe, ein Vergleich zwischen methodischem und intuitivem Vorgehen gezogen (Metrik 7). Dabei können die Prozentzahlen für die empfohlenen Entwurfsalternativen, die bei Anwendung der Verfahren erzielt wurden, mit denen bei der intuitiven Bewertung verglichen werden.

3.2.4. Bewertung einzelner Eigenschaften

Die Qualität bzw. der Reifegrad der Verfahren wird mit einer weiteren Frage untersucht:

4. Wie sind ausgewählte Eigenschaften der Performance-Analyseverfahren zu bewerten?

Zur Beantwortung dieser allgemeinen Frage können eine Reihe von konkreten Kriterien herangezogen werden. Folgendes Kriterienschema wurde entwickelt (Metrik 8, z.T. angelehnt an [\[BMDI04\]](#)):

- Notation, benutzte Software- und Performance-Modelle
 - Mächtigkeit (Welche Konstrukte lassen sich ausdrücken, welche nicht?)
 - Strukturierung (Wie übersichtlich sind die Darstellungen, gibt es eine Modularisierung des Performance-Modells?)
 - Aufnahme von Performance-Indizes (In welcher Form gelangen Schätzungen oder Messungen in das Performance-Modell?)
- Werkzeugunterstützung
 - Benutzungsoberfläche, Ergonomie (Wie gut sind die Werkzeuge bedienbar?)
 - Robustheit gegenüber Fehleingaben (Wie robust sind die Werkzeuge gegenüber Benutzungsfehlern?)
 - Stabilität (Arbeiten die Werkzeuge fehlerfrei, und verhalten sie sich robust gegenüber Fehleingaben?)
 - Präsentation der Ergebnisse (Wie gut ist die Nachvollziehbarkeit für den Entwickler?)
 - Praxistauglichkeit (Sind die Software-Werkzeuge reif für die Anwendung in der Software-Industrie?)
- Eignung für verschiedene Domänen

- Auswertungsmethode (Welche Methoden werden benutzt, was sind die Vor- und Nachteile?)
- Prozessintegration (Gibt es ein Vorgehensmodell, wie gut sind die Verfahren in die Software-Entwicklung integrierbar?)

Die hier benötigten Antworten können teilweise aus der Beschreibung der Verfahren entnommen werden. Während der Studie können Beobachtungen der Testpersonen einzelne Punkte erweitern bzw. konkretisieren. Einige Kriterien sind durchaus auf subjektive Antworten ausgelegt (z.B. Verständlichkeit der Notation oder Bewertung der Benutzungsoberflächen), darauf wird bei der Präsentation der Ergebnisse in Kapitel 6 nochmals ausdrücklich hingewiesen.

3.2.5. Zusammenfassung

Die folgende Tabelle 3.2 ist angelehnt an [BCR94] und fasst die Fragestellungen und Metriken des GQM-Ansatzes dieser Ausarbeitung überblicksartig zusammen.

Ziel	Zweck Sachverhalt Objekt Standpunkt	Empirische Bewertung der Anwendbarkeit von Performance-Analyseverfahren aus der Sicht des Entwicklers.	Skala
Frage	F1	Wie hoch ist die Genauigkeit der Vorhersage der Verfahren?	
Metrik	M1	Abweichung Eingabedaten verschiedener Teilnehmer	Sekunden niedrig, mittel, hoch Prozentzahl Prozentzahl
	M2	Abweichung Eingabedaten und Ausgabedaten	
	M3	Abweichung Ausgabedaten und Messdaten	
	M4	Prozentsatz richtiger Entwurfsentscheidungen	
Frage	F2	Wie hoch ist der Aufwand für die Performance-Analyse?	
Metrik	M5	Lernaufwand	Stunden
	M6	Dauer Performance-Modellierung	Stunden
Frage	F3	Welche Vorteile bringen Performance-Analyseverfahren gegenüber einem intuitiven Vorgehen?	
Metrik	M7	Abweichung Prozentsätze für Entwurfsentscheidungen	Prozentzahl
Frage	F4	Wie sind die Eigenschaften der Performance-Analyseverfahren zu bewerten?	
Metrik	M8	Kriterienschema	

Tabelle 3.2.: Überblick GQM

3.3. Forschungsmethode

Nachdem die Fragestellungen und Metriken dieser Studie festgelegt wurden, muss nun eine geeignete Forschungsmethode zur Beantwortung der Fragen gewählt werden. Von Prechelt [Pre01] werden folgenden empirische Forschungsmethoden in der Softwaretechnik genannt: Kontrolliertes Experiment, Natürliches Experiment, Fallstudie, Feldstudie, Umfrage und Metastudie.

3.3.1. Kontrolliertes Experiment

Den höchsten Grad an Vertrauen in die Beobachtungen genießen kontrollierte Experimente, die sich meist sehr genau verstehen lassen und gut reproduzierbar sind. Diese Methode ist folgendermaßen definiert:

Definition (Kontrolliertes Experiment): Ein kontrolliertes Experiment ist eine Studie, bei der alle voraussichtlich für das Ergebnis relevanten Umstände (Variablen) konstant gehalten werden (Kontrolle), mit Ausnahme von einem oder wenigen, die den Gegenstand der Untersuchung bilden (Experimentvariablen). Die Beobachtungen (abhängige Variablen) für verschiedene gezielt ausgesuchte Werte der Experimentvariablen (unabhängige Variablen) werden miteinander verglichen, um so zu reproduzierbaren Aussagen zu kommen, die eine vor dem Experiment definierte Experimentfrage beantworten. Die Experimentfrage ist ein genügend enger Aspekt einer relevanten Forschungsfrage.

Kontrollierte Experimente vergleichen also zwei oder mehr Fälle, die sich nur in einem Merkmal unterscheiden. Möchte man verschiedene Performance-Analyseverfahren in einem kontrollierten Experiment vergleichen, dann stellen die Verfahren selbst dieses Unterscheidungsmerkmal dar. Alle anderen relevanten Variablen (z.B. zu untersuchende Architektur, Abschätzung der Performance-Daten, Qualifikation der Testperson, Eingabedaten) müssen konstant gehalten werden. Die Kontrolle dieser anderen Variablen erscheint im begrenzten Rahmen dieser Diplomarbeit und aufgrund der den Verfahren inhärenten Eigenschaften nicht vollständig machbar:

- **Geringe Teilnehmerzahl:** Die zumeist wichtigste zu kontrollierende Variable in der Softwaretechnik ist die individuelle Variation in der Arbeitsweise und Leistung der einzelnen Versuchsteilnehmer. Aus diesem Grund wird in kontrollierten Experimenten versucht, die Unterschiede zwischen den Teilnehmern dadurch auszugleichen, dass man die gleiche Aufgabenstellung von mehreren Personen jeweils einzeln durchführen lässt. Aus den Ergebnissen kann dann durch statistische Mittelwertbildung ein repräsentatives Ergebnis erstellt werden. Im Rahmen dieser Diplomarbeit können die Testpersonen jedoch nur aus dem studentischen Umfeld herangezogen werden, wobei mit einer so geringen freiwilligen Beteiligung (unter 20 Teilnehmer) gerechnet werden muss, dass die Ergebnisse nicht mehr statistisch auswertbar sind.
- **Eignung der Verfahren:** Da nur das Verfahren selbst die (änderbare) Experimentvariable darstellt, müssen u.a. auch die Eingabeparameter für alle Verfahren gleich sein. Dies ist bei den teilweise höchst unterschiedlichen Verfahren jedoch nicht realisierbar, da ganz verschiedene Notationen verwendet werden. Das 'Capacity Planning' beispielsweise

benötigt zur Erstellung eines Performance-Modells Messungen (z.B. in Form einer Logdatei der Benutzereingaben), während bei vielen SPE-Verfahren UML-Diagramme mit noch unsicheren Erwartungswerten oder Schätzungen verwendet werden, da im Regelfall keine Implementierung vorliegt. Bei einem Experiment könnte man also nicht sicherstellen, dass nur das Verfahren an sich bewertet wird, da die unterschiedlichen Eingabeparameter das Ergebnis der Performance-Analyse verfälschen.

- **Lernaufwand:** Die Verfahren sind nicht Teil des Lehrplans an Universitäten und sollten den Testpersonen weitgehend unbekannt sein. Daher müssen die Testpersonen vor dem eigentlichen Experiment in den Verfahren unterrichtet werden. Dies kann aufgrund der Komplexität der Verfahren nicht während des Experiments erfolgen. Da der Zeitaufwand für die selbständige Aneignung der vorliegenden Literatur über die Verfahren den Testpersonen nicht zumutbar ist, müssen für jedes Verfahren Trainingssitzungen durchgeführt werden. Damit stellt die Vermittlung der Verfahren durch den Experimentator eine weitere unkontrollierbare Störvariable dar. Es könnte nicht geklärt werden, ob etwaige Fehler der Testpersonen auf die Eigenheiten des Verfahrens oder auf die Form und den Inhalt der Trainingssitzung zurückzuführen sind.

3.3.2. Fallstudie

Aus diesen Gründen soll die Studie in Form einer vergleichenden Fallstudie durchgeführt werden. Eine allgemeine Definition für Fallstudien stammt von Yin [[Yin94](#)]:

Definition: A case study is an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident.

Speziell auf die Softwaretechnik bezogen definiert Prechelt [[Pre01](#)]:

Definition: Eine Fallstudie ist die Beschreibung und Bewertung eines Werkzeugs oder einer Methode anhand eines konkreten Anwendungsbeispiels, das eigens zu diesem Zweck unter künstlichen oder unter typischen Bedingungen ausgeführt wird. Fallstudien können auch mehrere Werkzeuge oder Methoden im Vergleich betrachten, sorgen dabei jedoch im Gegensatz zu kontrollierten Experimenten nicht dafür, dass alle übrigen Faktoren konstant gehalten werden.

Im Sinne dieser Definition soll die folgende Untersuchung durchgeführt werden, wobei als konkretes Anwendungsbeispiel ein experimenteller Webserver auf seine Performance-Eigenschaften untersucht wird. Vorteil dieser Forschungsmethode ist die relativ einfache und universelle Anwendbarkeit. Der Aufwand kann recht gut an die verfügbaren Ressourcen und die gewünschte Genauigkeit der Ergebnisse angepasst werden, so dass sich diese Methode besonders gut für den engen zeitlichen Rahmen einer Diplomarbeit eignet. Der große Nachteil von Fallstudien ist die oftmals schwache Basis für eine Verallgemeinerung. In der Studie wird nur ein einziges Beispiel (Webserver) untersucht, welches möglicherweise nicht repräsentativ ist. Ob die Erkenntnisse über die Verfahren ohne weiteres auf die Untersuchung anderer Architekturen übertragen werden können, ist unklar. Bei der Auswertung müssen Faktoren, die möglicherweise die Ergebnisse

verfälschen, genau herausgestellt werden. Yin [Yin94] nennt als weiteren Nachteil von Fallstudien die oftmals große Menge an Daten und die nur schwierig zu lesenden Fallstudien-Berichte. Dieser Problematik soll mit dem GQM-Ansatz (s.o.) entgegengewirkt werden.

Als Folgerung der Wahl der Fallstudie-Methode für diese Untersuchung kann ein statistischer Hypothesentest nicht durchgeführt werden. Die Ergebnisse stellen eine qualitative Bewertung der einzelnen Verfahren dar.

Die weiteren empirischen Forschungsmethoden kommen für die Thematik dieser Diplomarbeit nicht in Betracht. *Feldstudien* ähneln vom Aufbau den Fallstudien, untersuchen jedoch reale Softwareprojekte oder -artefakte. Solche Projekte sind jedoch für diese Ausarbeitung nicht verfügbar, zusätzlich ist auch hier eine Verallgemeinerung der Ergebnisse schwierig. Bei einer *Umfrage* beantworten mehrere Personen eine Reihe von sorgfältig von Forschern ausgewählten Fragen. Da die Performance-Analyseverfahren für Software-Architekturen noch kaum in der Praxis eingesetzt werden, wäre es sehr schwierig geeignete Personen zu finden, die entsprechende Fragen beantworten könnten. Eine *Metastudie* wiederum ist eine vergleichende und zusammenfassende Auswertung mehrerer früherer Studien zu einem Thema. Studien zu den Performance-Analyseverfahren liegen in den allermeisten Fällen aber nur von den Autoren selbst vor, so dass eine ausreichende Basis für eine Metastudie nicht gegeben ist.

3.4. Verwandte Arbeiten

In der Literatur finden sich keine Studien, die unterschiedliche Performance-Analyseverfahren für Software-Architekturen empirisch bewerten. Dennoch sollen hier einige Arbeiten aufgeführt werden, die dieser Fallstudie nahe kommen.

In [BMDI04] werden zwei unterschiedliche Performance-Analysetechniken in einer Feldstudie auf ein reales System angewandt. Untersucht wird die Performance von zwei Szenarien in einem Marinekommunikationssystem. Das System wird mit einer Stochastische Prozessalgebra modelliert und analytisch ausgewertet. Zusätzlich wird eine Modellierung mit Anwendungsfall-, Aktivitäts- und Verteilungsdiagrammen der UML vorgenommen, diese mit Performance-Messungen annotiert und dann mit einer Simulation Vorhersagen für das System bei einer höheren Auslastung gemacht. Anhand verschiedener Kriterien (z.B. Ableitung des Performance-Modells, Generalität, Skalierbarkeit, Feedback) werden die Verfahren verglichen. Anzumerken ist, dass die Autoren der Studie auch die Autoren der beiden verwendeten Verfahren sind.

Im Jahr 1993 wurde in [SW93] eine Fallstudie zum SPE-Verfahren von Smith/Williams von den Autoren selbst durchgeführt. Dabei werden verschiedene Entwurfsalternativen für ein Temperatursensorsystem für Schmelzöfen mit den SPE-Techniken untersucht. Hier gibt es keinen Vergleich der Ergebnisse des Verfahrens mit Messungen an dem System.

Einige Erfahrungsberichte und Feldstudien finden sich z.B. in [Dum02] oder den Proceedings zum Workshop for Software and Performance [Spe04]. Dabei werden aber meistens Einzelfallbetrachtungen durchgeführt und keine strukturierten Verfahren angewandt. Die Erstellung von Performance-Modellen erfolgt in diesen Studien meist manuell und ist auf andere Anwendungsfälle kaum zu übertragen.

Ein ähnliches Vorgehen in der Performance-Analyse wie in dieser Fallstudie und einen Vergleich von Vorhersagen und Implementierung eines Verfahren liefern Gorton et. al. [GL03,

[CLGL04](#)]. Dabei werden speziell Entwurfsalternativen für Komponentenarchitekturen untersucht, die mit Enterprise Java Beans realisiert wurden.

Eine Reihe von vergleichenden Fallstudien, die der Struktur dieser Arbeit ähneln, sind bereits zu anderen Themen durchgeführt worden. Verschiedene Techniken zur Spezifikation von Softwaresystemen (wie z.B. UML oder Z) wurden in [\[vKKR⁺98\]](#) verglichen. Dazu wurde ein Klassifikationsschema aufgestellt, in dem die Eigenschaften der Notationen erläutert wurden. Jedoch wendete hier immer nur eine Testperson eines der Verfahren an, daher ist Qualifikation der Versuchsteilnehmer ein beeinflussender Faktor. Eine ähnliche Arbeit zum Vergleich von Softwarespezifikationstechniken liegt in Form einer Diplomarbeit vor [\[Feh99\]](#).

In [\[Pre00\]](#) untersucht Prechelt sieben Programmiersprachen in einer empirischen Studie auf verschiedene Kriterien wie Laufzeit, Speicherverbrauch, Programmlänge, Zuverlässigkeit und Produktivität.

Ein kontrolliertes Experiment zur Erforschung der Beziehung zwischen den Schnittstellen und der Funktionalität von Softwarekomponenten wurde von Kratz [\[Kra03\]](#) im Rahmen einer Diplomarbeit durchgeführt. Bei dieser Arbeit wurde ähnlich wie in der folgenden Fallstudie versucht, die äußeren Einflüsse auf das Experiment möglichst gering zu halten.

4. Auswahl und Vorstellung der untersuchten Verfahren

4.1. Überblick Performance-Analyseverfahren

Die verschiedenen, bekannten Performance-Analyseverfahren können im Rahmen dieser Diplomarbeit nicht detailliert vorgestellt werden. Jedoch soll der folgende Überblick über die Verfahren die spätere Fallstudie in einen breiteren Kontext einbetten und den Hintergrund für die Motivation liefern, warum gerade die drei ausgewählten Verfahren exemplarisch untersucht werden. Ein tabellarischer Überblick über die Verfahren findet sich in Anhang A.

V1: Das Verfahren von Smith und Williams [Smi90, Smi02] verwendet ein Software-Modell basierend auf Ausführungsgraphen (engl. Execution Graphs) und ein System-Modell basierend auf QNs. Das Software-Modell wird in diesem Verfahren manuell aus erweiterten Sequenzdiagrammen der UML abgeleitet, während das Systemausführungsmodell automatisch durch ein Werkzeug (SPE-ED [SW97]) erstellt und ausgewertet werden kann.

V2: Beim Verfahren PRIMA-UML [CM02] wird ein QN direkt aus annotierten UML-Diagrammen abgeleitet, in einer weiteren Version für mobile Software-Architekturen [GM02] werden erweiterte Ausführungsgraphen und EQNs verwendet.

V3: In [CDI01] leiten Cortellessa et. al. LQNs aus Klassen- und Sequenzdiagrammen ab. Dazu werden eine Reihe von Informationen (wie Konfiguration, Ressourcenkapazität und Benutzerprofil) identifiziert, die das Verfahren benötigt, um das Performance-Modell zu erstellen. Als Zwischenschritt wird ein erweiterter Ausführungsgraph erstellt.

V4: Ein weiterer Ansatz [GM00, GM01] untersucht das Zusammenspiel von Komponenten bei Client/Server-Systemen und erstellt auf Basis von Klassen- und Kollaborationsdiagrammen ein EQN. Dies war das erste Verfahren, dass sich speziell mit komponentenbasierten Software-Architekturen [SGM02] beschäftigte.

V5: Petriu et. al. stellen drei ähnliche Verfahren [PW99, PS02, GP02] vor, bei denen durch Architekturmuster (z.B. Client/Server, Pipe-and-Filters, Broker, Layers) beschriebene Software-Architekturen [SG96] mittels Graphen-Transformation in ein LQN umgewandelt werden. Dabei kommt auch das in Abschnitt 2.4 erläuterte UML Performance Profile zum Einsatz. Der dritte dieser Ansätze verwendete die eXtensible Stylesheet Language Transformations (XSLT) [Rec99] zur Erstellung der LQNs aus UML-Modellen, die im XML Metadata Interchange Format (XMI) [OMG02] gespeichert werden. Die Auswertung der LQNs kann durch existierende LQN-Werkzeuge vorgenommen werden.

V6: Das CLISSPE-Verfahren (CLient/Server Software Performance Evaluation) beinhaltet eine Sprache, mit der Client, Server und Netzwerke beschrieben werden können [MG98, MG00]. Ein Compiler generiert daraus QNs und berechnet deren Lösung. Das Verfahren benutzt Anwendungsfall- und Klassendiagramme der UML für statische Strukturen und Kollaborationsdia-

gramme, um dynamische Aspekte einer Software-Architektur abzubilden.

V7: Beim Verfahren von Andolfi et. al. [[AABI00](#)] werden MSCs oder Labeled Transition Systems (LTS) [[ABI01](#)] als Eingabe für einen Algorithmus benutzt, der automatisch QNs erstellt. Dieses Verfahren untersucht die Sequenzen von Nachrichten, die zwischen Komponenten ausgetauscht werden, um den Grad der Parallelisierung zwischen den Komponenten und ihre dynamischen Abhängigkeiten zu analysieren.

V8: Von Woodside et al. [[WHSB01](#)] stammt eine Methode zur Ableitung von LQNs aus einer kommerziellen Software-Entwurfsumgebung namens ObjecTime Developer mittels eines prototypischen Software-Werkzeugs namens Performance Analysis Model Builder (PAMB). PAMB zeigt die Ergebnisse der Performance-Analyse in Form von annotierten MSCs an. In [[PW02a](#)] stellen Woodside et al. einen Ansatz vor, um LQNs aus Use Case Maps abzuleiten. Dazu wurde das Werkzeug UCM2LQN implementiert, das den Transformationsprozess automatisiert.

V9: Kähkipuro [[Käh01](#)] stellte noch vor Veröffentlichung des UML Performance Profiles einen Ansatz vor, um UML-Diagramme mit Performance-Attributen zu ergänzen. Diese Repräsentationen werden dann in eine textuelle Beschreibung transformiert, die ausschließlich die Performance-Aspekte des Systems enthält. Diese wiederum wird auf ein EQN abgebildet, welches durch Analyse oder Simulation ausgewertet werden kann. Zu diesem Verfahren existiert ein prototypisches Werkzeug namens OAT (Object-oriented performance modelling and Analysis Tool).

V10: Lindemann et. al. [[LTK⁺02](#)] verwenden zur Performance-Modellierung stochastische Prozesse bei denen die Markowschen Eigenschaften nur zum Teil erfüllt sind. Diese werden automatisch aus erweiterten Zustands- und Aktivitätsdiagrammen der UML generiert. Problem bei diesem Ansatz sind die nicht standardkonformen UML-Erweiterungen und die aufgrund von Zustandstandsexplosion unter Umständen nicht analytisch auswertbaren Zustandsgraphen.

V11: Beim Verfahren von de Miguel et. al. [[MLH⁺00](#)] werden UML-Diagramme als Software-Modell verwendet und mit Stereotypes, Tagged Values und Constraints zur Aufnahme von Zeiten und Ressourcennutzungen versehen. Mittels des 'Simulation Model Generators' werden aus diesen Diagrammen Simulationsmodelle erstellt. Dieser Ansatz beinhaltet auch einen Feedback-Mechanismus der Performance-Ergebnisse, diese werden in Tagged-Values der UML-Diagramme geschrieben.

V12: Arief und Speirs [[AS99](#)] stellen ein Verfahren vor, bei dem UML-Spezifikationen in eine prozessorientierte Simulation transformiert werden, die als XML-Datei in der 'Simulation Modelling Language' SimML repräsentiert wird. Durch dieses Format wird das Simulationsmodell unabhängig von der Implementierung der Simulation gemacht. Die Autoren stellen zwei Back-Ends vor, die die SimML-Notation in Simulationsprogramme in Java oder C++ umwandeln.

V13: Auch der Ansatz von Marzolla et. al. [[Mar04](#)] basiert auf Simulationen. Die Software-Architektur wird mittels Anwendungsfall-, Aktivitäts und Verteilungsdiagrammen in UML modelliert, mit dem UML Performance Profile annotiert und anschließend mit dem Werkzeug uml-PSI automatisch in eine diskrete Simulation transformiert.

V14: Von Hennig et. al. [[HRP03](#)] stammt ein Simulationsframework, mit dem Sequenz-, Kollaborations- und Verteilungsdiagramm ausgewertet werden können. Die Software-Modelle werden zunächst in ein XML-Dokument übertragen, bevor dann Code für ein Simulationsprogramm generiert wird. Der Simulator basiert auf dem OMNet++-Paket und enthält einige SPE-Erweiterungen. Die Ergebnisse der Simulation werden in Tagged-Values der ursprünglichen

UML-Diagramme eingetragen.

V15: King und Pooley [KP00] benutzen kombinierte Kollaborations- und Zustandsdiagramme, um ein generalisiertes Stochastisches Petri-Netz manuell zu erstellen. Dabei werden die Zustände und Transitionen der Zustandsdiagramme auf Stellen und Transitionen der Petri-Netze abgebildet. Die Auswertung der Petri-Netze kann dann mit dem Werkzeug SPNP vorgenommen werden.

V16: Ein weiteres Verfahren zur Erstellung von generalisierten Stochastischen Petri-Netzen aus UML-Diagrammen stammt von Bernardi et. al. [BDM02]. Dabei werden als Ausgangsbasis Zustands- und Sequenzdiagramme verwendet. Jeder Zustand eines Zustandsdiagramms mit allen abgehenden Transitionen wird zunächst in ein einzelnes Petri-Netz übertragen, danach werden die resultierenden Petri-Netze über gleichnamige Stellen zusammengesetzt. Bei den Sequenzdiagrammen wird in ähnlicher Weise jede Nachricht zunächst in ein Subsystem eines GSPN abgebildet, und Subsysteme anschließend verbunden. Das endgültige Modell wird durch die Kompositionen der beiden resultierenden GSPNs erstellt.

V17: Bernardo et. al. [BCD00] stellen eine Architekturbeschreibungssprache namens AEMPA vor, deren Spezifikationen automatisch in eine Stochastische Prozessalgebra (EMPA) übersetzt werden können. Dabei kommt zur Analyse der Prozessalgebra das Werkzeug "TwoTowers" zum Einsatz.

V18: Das Projekt "Performance Modelling for ATM Based Applications and Services" (PER-MABASE) [WLA⁺01] beschäftigt sich mit der Transformation von UML-Modellen in eine Zwischendarstellung (Composite Model Data Structure, CMDS), aus der automatisch Performance-Modelle generiert werden können. Eine Rückabbildung der Performance-Ergebnisse in die Zwischendarstellung ist vorgesehen.

V19: Hoeben benutzt Klassen- und Komponentendiagramme um Informationen über Performance-Charakteristiken abzubilden [Hoe00]. Das Verhalten eines System wird mit Sequenz- und Kollaborationsdiagrammen modelliert, zusätzlich stellen Verteilungsdiagramme Server und Netzwerkverbindungen im System dar. Das aus diesen Spezifikationen abgeleitete Performance-Modell basiert auf Warteschlangennetzen.

V20: Beim "Capacity Planning" nach Menasce et.al. [MAD94] werden keine Software-Modelle verwendet, es wird nur die Systemarchitektur mit Warteschlangenmodellen abgebildet. Die von den Autoren zur Verfügung gestellten Spreadsheets zur Auswertung von Warteschlangenmodellen müssen mit Messdaten parametrisiert werden. Näheres zu diesem Verfahren folgt in Abschnitt 4.5.

4.2. Kriterien für die Auswahl

Aufgrund der begrenzten Zeit von 6 Monaten für die Anfertigung der Diplomarbeit und der teilweise hohen Komplexität der Verfahren müssen aus den bekannten Performanc-Analyseverfahren möglichst repräsentative Vertreter ausgewählt werden. Die Auswahl richtet sich danach, wie die gestellten Forschungsfragen aus Kapitel 3 an die Verfahren am besten beantwortet werden können. Da die Fallstudie bei den Testpersonen auf Studenten angewiesen ist, ergeben sich für jedes Verfahren mehrere Mindestanforderungen:

- **hoher Reifegrad:** Das Verfahren muss möglichst weit ausgearbeitet sein, es muss genügend

Dokumentation darüber zur Verfügung stehen. Idealerweise sind kleinere Fallstudien vorhanden.

- **Werkzeugunterstützung:** Ein Software-Werkzeug sollte vorhanden sein. Dies sollte es den Testpersonen erleichtern, die Verfahren anzuwenden. Gleichzeitig kann die Erfassung der Ergebnisse teilweise automatisch erfolgen.
- **Standardnotationen:** Damit das Verfahren in der Kürze der zur Verfügung stehenden Zeit den Testpersonen vermittelt werden kann, muss es auf Standardnotationen zurückgreifen, mit denen die Testpersonen vertraut sind. Die meisten Verfahren verwenden für ihr Software-Modell UML. Diese Modellierungssprache wird im Informatik-Studium in Pflichtveranstaltungen behandelt und sollte den Testpersonen daher vertraut sein. Deshalb bieten sich Verfahren an, die die UML einsetzen.

Am weitesten ausgearbeitet unter den SPE-Verfahren erscheint der Ansatz von Smith/Williams (V1), die bereits auf über 20 Jahre Erfahrung in diesem Bereich zurückblicken können. Von den Autoren existiert ein Buch [Smi02], in dem die Methode ausführlich vorgestellt wird. Es enthält einige erklärende Fallstudien und wird bei SPE-Lehrveranstaltungen gerne als Grundlagen-Literatur verwendet [Dug04]. Für das Verfahren existiert weiterhin ein Werkzeug (SPE-ED [SW97]), das in einer (für die Fallstudie ausreichenden) Demonstrations-Version frei verfügbar ist. Die Performance-Modelle können aus einer erweiterten Version der UML-Sequenzdiagramme abgeleitet werden und basieren somit auf Standard-Notationen. Da die Autoren den Begriff SPE geprägt haben, wird im folgenden das Verfahren von Smith/Williams als "SPE-Verfahren" bezeichnet.

Als zweites Verfahren wurde die simulationbasierte Performance-Modellierung mit dem umlPSI-Werkzeug von Marzolla [Mar04] (V13) als Vertreter für die Klasse der simulationsbasierten Verfahren ausgewählt. Somit können in der Fallstudie die Eigenschaften von Simulation gegenüber analytischen Verfahren untersucht werden. Für das Verfahren ist ausreichend Dokumentation in Form einer Dissertation und einiger erklärender Fallstudien vorhanden. Weiterhin existieren mit umlPSI (UML Performance Simulator) und der Simulationsbibliothek libcppsim quelloffene Werkzeuge, mit denen ein UML-Modell automatisch in eine Simulation umgewandelt und die Simulation dann auch ausgeführt werden kann. Als einziges bekanntes Simulationsverfahren verwendet diese Methode das UML Performance Profile, dessen Tauglichkeit ebenfalls in der Fallstudie untersucht werden soll.

Aus der Thematik des Capacity Plannings sollen als weiteres Verfahren die Methoden von Menasce und Almeida (V20) exemplarisch untersucht werden, wie sie in [MAD04] und [MA02] vorgestellt wurden. Durch die fünf vorliegenden Bücher und die langjährige Forschung kann diesem Verfahren ein hoher Reifegrad bescheinigt werden. Im aktuellen Buch von Menasce/Almeida finden sich eine Reihe von erklärenden Fallstudien und Beispielen, an denen sich diese Arbeit orientieren kann. Als Werkzeuge liegen zumindest mehrere Excel-Spreadsheet-Dateien vor, mit denen Cluster-Analysen der Systemlast durchgeführt werden und die Antwortzeiten, Auslastungen und Warteschlangenlängen von Warteschlangenmodellen berechnet werden können. Bei diesem Verfahren werden keine UML-Diagramme zur Modellierung der Software-Architektur verwendet. Die verschiedenen Typen von Warteschlangenmodellen und deren Analyse muss den Teilnehmern der Studie vorher vermittelt werden.

Die ausgewählten Verfahren werden in den folgenden Abschnitten soweit vorgestellt, wie es für das Verständnis der Fallstudie erforderlich ist. Für weitere Details zu den Verfahren wird auf die Folien der Übungen im Anhang und die genannte Literatur verwiesen.

4.3. Verfahren von Smith/Williams (SPE)

Smith/Williams geben zu ihrer SPE-Methode ein ausführliches, schrittweises Vorgehensmodell (Abbildung 4.1) zur Integration in den Software-Entwicklungsprozess an.

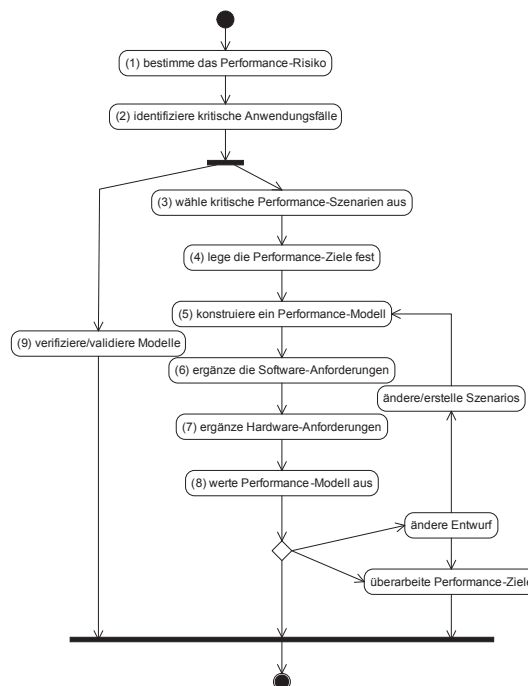


Abbildung 4.1.: SPE-Prozess [Smi02]

Der SPE-Prozess konzentriert sich auf Performance-kritische Anwendungsfälle und die Szenarien, die den zugehörigen Kontrollfluss durch die Architektur abbilden. Zu Beginn eines Software-Entwicklungsprojekts wird zunächst das allgemeine Performance-Risiko festgestellt (1), um die notwendige Komplexität der Performance-Modelle planen zu können. Für weniger Performance-kritische Anwendungen reichen z.B. einfache Modelle aus. Danach werden Performance-kritische Anwendungsfälle gesucht, die für den Betrieb des Systems oder die vom Benutzer wahrgenommene Performance besonders wichtig sind (2). Jeder dieser relevanten Anwendungsfälle wird anschließend (3) mit einer erweiterten Form der UML-Sequenzdiagramme modelliert. Die Erweiterungen sind angelehnt an die Eigenschaften von Message Sequence Charts [Sta99] und erlauben die Verwendung von Schleifen, Kontrollflussverzweigungen, optionalen Abschnitten, parallelen Aktionen und Referenzen auf andere Sequenzdiagramme. Anschließend (4) werden konkrete Performance-Ziele wie Antwortzeiten, maximale Ressourcenbelastung und Durchsatz auf Basis der Kundenanforderungen festgelegt. Beispielsweise darf die maximale Antwortzeit für eine Benutzeranfrage höchstens fünf Sekunden betragen, oder es müssen mindestens zehn

Anfragen pro Minute verarbeitet werden können.

Aus den Sequenzdiagrammen wird dann (5) das Softwareausführungsmodell in Form von Ausführungsgraphen (engl. Execution Graphs) erstellt. Ausführungsgraphen ähneln den Zustandsdiagrammen der UML und können alle Eigenschaften der erweiterten Sequenzdiagramme abbilden. Sie eignen sich wegen ihrer Übersichtlichkeit besser für die Aufnahme von Performance-Attributen. In diese Graphen fließen zusätzlich Informationen über die Ausführungshäufigkeit von Schleifen und die Wahrscheinlichkeit einzelner Aktionen ein (6). So bestellt beispielsweise ein Kunde bei einem Web-Shop durchschnittlich drei Artikel oder tätigt bei 40 Prozent seiner Online-Banking-Sitzungen eine Überweisung. Diese Werte werden entweder statistisch bestimmt, aufgrund von Erfahrung abgeschätzt oder in *Performance-Walkthroughs* ermittelt. Performance-Walkthroughs sind informelle Reviews zur Sammlung von Informationen für die Konstruktion eines Performance-Modells, bei denen ein Mitglied des Entwicklungsteams die Teilnehmer durch ausgewählte Teile der Systemarchitektur oder der Implementierung führt. Jedem Knoten eines Ausführungsgraphen werden die Software-Ressourcen zugeordnet, die bei der entsprechenden Aktion gebraucht werden. Dies kann beispielsweise die Dauer der Rechenzeit auf dem Prozessor oder die Anzahl von Datenbankabfragen und Aufrufen an externe Systeme sein. Da zu Beginn der Entwicklung die genauen Werte hier nur schwierig abgeschätzt werden können, arbeitet man häufig mit 'best-case' und 'worst-case'-Werten. Beispielsweise braucht eine Aktion im günstigsten Fall eine Netzwerknachricht und zwei Datenbankzugriffe.

Die Software-Anforderungen werden im nächsten Schritt in einer Tabelle auf die Computer-Anforderungen abgebildet (7). So braucht z.B. ein Datenbankzugriff auf dem Zielsystem zur Ausführung 500.000 CPU-Instruktionen (beinhaltet das Öffnen und Schließen der Datenbankverbindung) und zwei Festplattenzugriffe. Die Service-Zeiten, die z.B. das Abarbeiten einer CPU-Instruktion oder ein Festplattenzugriff verursacht, können aus den Hardware-Spezifikationen abgeleitet oder von Performance-Experten zur Verfügung gestellt werden. Mit diesen Werten kann man nun für einen Knoten im Ausführungsgraphen die Antwortzeit ausrechnen. Dies geschieht anschließend sukzessive für alle Knoten eines Ausführungsgraphen, wobei die vorher angegebenen Wahrscheinlichkeiten und Ausführungshäufigkeiten für bestimmte Aktionen berücksichtigt werden. Auf diese Weise werden die 'best/worst-case' Antwortzeit für einen kompletten Ausführungsgraphen und somit für ein Szenario berechnet.

Automatisiert man die Berechnung mit dem SPE-ED-Werkzeug, können schnell verschiedene Entwurfsalternativen getestet werden (8). Liegen bei der Auswertung der Ausführungsgraphen die berechneten Antwortzeiten bzw. Ressourcenauslastung nicht innerhalb der in Schritt (4) bestimmten Performance-Ziele, gibt es zwei Möglichkeiten. Entweder die Software-Architektur und somit das Szenario kann so modifiziert werden, dass die Performance-Ziele nach erneuter Auswertung eingehalten werden können oder die Performance-Ziele müssen mit dem Auftraggeber neu verhandelt werden. Dies erscheint zwar nicht erwünscht, ist in diesem frühen Stadium der Entwicklung aber noch relativ leicht durchführbar. Falls das System bereits implementiert wurde und anschließend bemerkt wird, dass die Performance-Ziele nicht eingehalten werden können, ist dies nicht mehr machbar.

Sind die Performance-Modelle auf Basis von Ausführungsgraphen erstellt, so müssen sie während des Software-Entwicklungsprozesses möglichst oft mit dem aktuellen Stand der Implementierung verglichen werden. Die anfangs oftmals noch geschätzten Performance-Werte können im Verlauf der Entwicklung durch Messungen ersetzt werden. Weiterhin existieren Kon-

strukturen, die die Modellierung von synchroner bzw. asynchroner Kommunikation in verteilten Systemen ermöglichen und somit die Modelle verfeinern. Auf diese Weise wird sicher gestellt, dass tatsächlich das System modelliert wurde, das auch implementiert wird.

Dieses Softwareausführungsmodell ist ein einfaches Modell und berücksichtigt noch nicht, dass meist mehrere Benutzer oder Prozesse gleichzeitig um begrenzte Ressourcen (wie CPU, Datenbanken, Netzwerkverbindungen) konkurrieren. Falls mit diesem Modell keine Performance-Probleme mehr gefunden werden, wird deshalb danach ein weiterführendes Systemausführungsmodell erstellt. Dieses verwendet zur Modellierung von konkurrierenden Ressourcen Warteschlangenmodelle, wobei deren Eingabeparameter aus dem Softwareausführungsmodell bezogen werden können. Damit kann dann z.B. die durchschnittliche Zeit berechnet werden, die ein Prozess in einer Warteschlange auf seine Bearbeitung wartet. Auf diese Weise können Flaschenhälse in der Hardware-Architektur identifiziert werden. Die Erstellung des Systemausführungsmodells erfolgt mit dem SPE-ED-Werkzeug mit Hilfe der vorher modellierten Ausführungsgraphen automatisch, zusätzlich muss die Ankunftsrate der Anfragen (offenes QN) bzw. die Anzahl der Benutzer im System (geschlossenes QN) angegeben werden.

Es existieren einige erläuternde Fallstudien, die die Anwendung des Verfahrens z.B. auf Web-Applikationen oder Realzeitsysteme demonstrieren. Indikator für den Reifegrad des Verfahrens sind die vorgestellten Performance-Prinzipien und die Performance-Patterns/Antipatterns. Patterns kapseln bewährte Lösungen für häufig auftretende Entwurfsprobleme und stellen "best-practices" dar. Die Patterns im Performance-Kontext sind angelehnt an die Design Patterns für objektorientierte Systeme nach Gamma et. al. [GHJV95]. Sie stellen bewährte Lösungen für Performance-Probleme von Software dar, bzw. zeigen als Antipatterns typische Architekturmängel, die zu einer schlechten Performance des Systems führen.

4.4. Verfahren von Marzolla/Balsamo (umlPSI)

Das recht neue Verfahren von Marzolla/Balsamo vom Februar 2004 beruht auf ausführbaren Simulationen. Dabei imitiert die Simulation das spätere Verhalten des Systems und kann so Messungen von Performance-Werten durchführen. Das Verfahren lässt sich aufgrund der Nutzung von UML-Diagrammen und der Verfügbarkeit eines Werkzeugs zur automatischen Erstellung der Simulationen leicht in den Software-Entwicklungsprozess integrieren. Ein manuelles Erstellen der Performance-Modelle ist nicht notwendig.

Ausgangspunkt des Verfahrens sind Anwendungsfall-, Verteilungs- und Aktivitätsdiagramme, mit denen die zu untersuchende Architektur unter Benutzung eines CASE-Werkzeugs (ArgoUML [Eng04] oder Poseidon [Gen04]) modelliert wird. Anwendungsfalldiagramme werden verwendet, um die Systemlast bzw. die Anzahl der Benutzer zu modellieren. Anstatt Sequenzdiagramme werden in diesem Verfahren Aktivitätsdiagramme für die Abbildung des Kontrollflusses verwendet, da sich mit ihnen Verzweigungen im Kontrollfluss bzw. parallele Aktivitäten darstellen lassen. Jeder Aktivität wird eine Ressource aus dem Verteilungsdiagramm zugeordnet, welches die Systemarchitektur modelliert. Alle Diagramme werden mit Hilfe des UML Performance Profiles [OMG03b] (siehe auch Abschnitt 2.4) mit Performance-Angaben (wie Schätzungen für Ausführungszeiten, Ankunftsraten, Benutzerzahlen, Wahrscheinlichkeiten für Transitionen) versehen. Die anschließend im Datenformat XMI [OMG02] vorliegenden Diagramme dienen dem

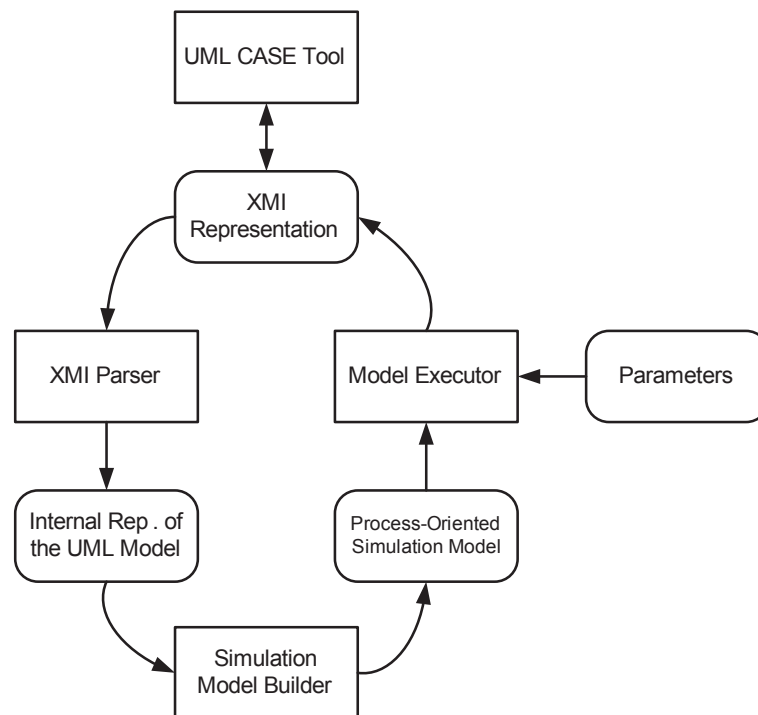


Abbildung 4.2.: Simulation von UML-Modellen [Mar04]

Simulationswerkzeug UML-Performance-Simulator (umlPSI) als Eingabe. Dieses erzeugt automatisch eine diskrete Simulation (im Gegensatz zu einer kontinuierlichen wechseln hier die Zustände nur an diskreten Zeitpunkten), für die noch die Anzahl der Jobs im System sowie weitere Simulationsparameter angegeben werden müssen.

Die Ausführung der Simulation mit umlPSI berechnet für alle Anwendungsfälle die mittleren Antwortzeiten und für Ressourcen Durchsätze und Auslastungen. Das Verfahren verfügt darüber hinaus über einen Feedback-Mechanismus für die Performance-Ergebnisse. Mittels des UML Performance Profiles werden die gemessenen Werte direkt in die XMI-Dateien zurückgeschrieben werden, da dieses für die Ergebnisse eigene 'Tagged-Values' vorsieht. Öffnet der Entwickler die XMI-Datei erneut mit seinem CASE-Werkzeug, kann er sofort Performance-Probleme seiner Architektur konkreten Aktivitäten und Ressourcen zuordnen und schnell verschiedene Alternativen testen.

Für dieses Verfahren liegen zwei erklärende Fallstudien der Autoren vor [BM03, BMDI04].

4.5. Verfahren von Menasce/Almeida (CP)

Das Vorgehensmodell des 'Capacity-Plannings' findet sich in Abbildung 4.3. Zunächst werden die Performance-Ziele in Form von Service Level Agreements (SLA) bestimmt. Dies sind feste, mit dem Auftraggeber des Systems ausgehandelte Performance-Vorgaben wie z.B. garantierte maximale Antwortzeiten oder Anzahl der Benutzer im System. Anschließend folgt die quantitative Analyse des Systems. In Schritt (1) muss ein allgemeines Verständnis für das zu untersuchende System entwickelt werden. Welche Typen von Software (Betriebssystem, DBMS, Applikatio-

nen usw.) werden eingesetzt, welche Hardware-Elemente (CPU, Festplatten, Netzwerke usw.) sind vorhanden? Daraus ergibt sich eine systematische Beschreibung der Systemarchitektur. Als nächstes (2) wird ein so genanntes Systemlast-Modell (engl. Workload-Model) erstellt, das die Typen und Anzahl der Eingaben an das System möglichst genau charakterisiert. Dabei unterscheidet man zwischen natürlichen (z.B. aus der Log-Datei eines realen Servers entnehmbaren) und künstlichen (Programme zur künstlichen Generierung von Systemeingaben) Modellen. Die Systemlast wird idealerweise zur Verfeinerung der Performance-Ergebnisse in mehrere Klassen eingeteilt, so dass z.B. für teure Datenbankabfragen und schnelle Cache-Zugriffe jeweils eigene Performance-Analysen vorgenommen werden. Für diese Einteilung können Cluster-Analysen der Eingabesequenzen vorgenommen werden.

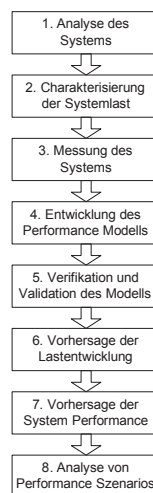


Abbildung 4.3.: Vorgehensmodell beim Capacity Planning [MAD04]

Bei der Messung des Systems (3) wird nun versucht, durch verschiedene Werkzeuge wie z.B. Performance-Monitore möglichst viele Performance-Daten des Systems zu erfassen. Diese Daten können anschließend (4) den Performance-Modellen als Eingabeparameter dienen. Die Performance-Modelle beim Capacity Planning basieren ähnlich wie viele Verfahren des SPE auf Warteschlangenmodellen. Menasce und Almeida benutzen verschiedene Konstrukte für Warteschlangenmodelle, um z.B. mehrere Benutzerklassen, unterschiedliche Typen von Ressourcen, begrenzte Warteschlangen, Ressourcenkonkurrenz und gleichzeitige Ressourcennutzung zu modellieren. Mittels der bekannten Gesetzmäßigkeiten für Warteschlangenmodelle (siehe Abschnitt 2.3.2) können aus den gemessenen Performance-Daten die benötigten Eingabeparameter wie Ankunftsraten und Ressourcennutzungszeiten berechnet werden.

Wichtig ist die Verifikation und Validation (5) der konstruierten Workload- und Performance-Modelle an realen Systemen. Dabei gilt z.B. ein Performance-Modell als valide wenn die berechneten Werte für Auslastungen und Durchsatz weniger als 10% und für Antwortzeiten weniger als 20% von den gemessenen Werten abweichen. Die Modelle müssen so lange kalibriert werden, bis die notwendige Genauigkeit erreicht ist.

Da die Systemlast bei den meisten Systemen im Verlauf der Zeit nicht konstant bleibt, werden durch statistische Methoden Vorhersagen für die zukünftige Entwicklung der Systemlast getroffen (6). Beispielweise könnte man 3 Monate nach Eröffnung eines Web-Portals durch den

steigenden Bekanntheitsgrad eine doppelt so hohe Benutzerzahl erwarten wie im ersten Monat. Mittels des Performance-Modells und der erwarteten Systemlast kann dann die zukünftige System-Performance vorhergesagt werden (7). Auf diese Weise kann festgestellt werden, bei welcher Systemlast bestimmte Ressourcen überlastet werden. Für verschiedene Entwurfsalternativen und Systemarchitekturen (Aufrüstung von Servern, schnellere Netzwerke, Veränderungen am Software-System usw.) können dann mit Hilfe des Performance-Modells die erwarteten Antwortzeiten und Ressourcenauslastungen vorhergesagt werden (8).

Das 'Capacity Planning' ist nicht wie der SPE-Ansatz nach Smith/Williams eng in den Software-Entwicklungsprozess integriert. Auch das Vorgehensmodell ist nicht so straff durchführbar wie beim SPE. Menasce/Almeida stellen in ihren Publikationen viele Beispiele vor, an denen verschiedene Aspekte der Performance-Modellierung demonstriert werden. Das Verfahren besteht aus einer Vielzahl von kleineren Techniken und Methoden, die in ihrer Gesamtheit in dieser Fallstudie nicht untersucht werden können. Jedoch können einzelne Aspekte wie die Systemlast-Modellierung und die Konstruktion eines einfachen Warteschlangen-basierten Performance-Modells exemplarisch untersucht werden.

5. Entwurf und Durchführung der Fallstudie

Kern dieser Fallstudie ist die experimentelle Untersuchung eines experimentellen Webservers mit drei ausgewählten Performance-Analyseverfahren. Auch wenn der Aufbau der Fallstudie einem Experiment ähnelt, so können, wie in Kapitel 3 erläutert, nicht alle relevanten Störvariablen kontrolliert werden.

5.1. Randbedingungen

Außer der Auswahl einer kleinen Anzahl von Verfahren (siehe Kapitel 4) mussten weitere Randbedingungen beim Entwurf der Fallstudie berücksichtigt werden. Für die Auswahl der Versuchsteilnehmer bot sich die Einbeziehung von Teilnehmern der Vorlesung "Komponentenbasierte Softwareentwicklung" im Sommersemester 2004 an. Für die erfolgreiche Teilnahme an dieser Vorlesung musste außer einer mündlichen Prüfung eine Vorleistung erbracht werden, die entweder in Form einer Seminaarausarbeitung oder durch die Teilnahme an dieser Studie geleistet werden konnte. Dabei setzte sich die Benotung der Vorlesung zu 70% aus der mündlichen Prüfung und zu 30% aus der erfolgreichen Erbringung der Vorleistung zusammen. Auf diese Weise war für die Studenten eine gewisse Motivation gegeben, an der Studie teilzunehmen, da sie sich außer dem Kennenlernen von Performance-Analyseverfahren für Software-Architekturen diese Studie auch als Teil ihrer Studienleistung anrechnen lassen konnten. Damit konnte auf weitere Teilnahmeanreize (wie z.B. monetäre Vergütung) verzichtet werden.

Um den Aufwand für die Seminaarausarbeitung bzw. für die Teilnahme an der Studie ungefähr gleich zu gewichten, wurde vereinbart, dass für die Studie ca. sechs jeweils zweistündige Termine plus die Anfertigung der Lösung von Übungszetteln veranschlagt werden konnte. Erwartet werden konnten aufgrund der Besucherzahl der Vorlesung im vorherigen Jahr etwa 20 Studenten, tatsächlich beteiligten sich schließlich 24 Personen an der Studie. Einige statistische Daten zur näheren Charakterisierung der Teilnehmer finden sich in Abschnitt 5.3. Daraufhin wurde folgender Aufbau für die Fallstudie gewählt:

5.2. Kurzüberblick

Zunächst mussten die Verfahren den Studenten vorgestellt werden. Die selbständige Aneignung über die Literatur erschien aufgrund des Umfangs der benutzten Bücher und deren mangelnder Verfügbarkeit in der Universitätsbibliothek als nicht durchführbar. Deshalb wurden die Verfahren in jeweils zweistündigen Trainingssitzungen vorgestellt, wozu entsprechende Foliensätze anhand der Literatur erstellt wurden (siehe Anhang B). Eine vollständige Darstellung der einzelnen Verfahren war in nur zwei Stunden nicht machbar, jedoch konnten die wichtigsten und für das spätere Experiment relevanten Elemente vermittelt werden. Am Ende jeder Trainingssitzung wurde ein Übungszettel mit mehreren Aufgaben ausgegeben, damit die Studenten die

5. Entwurf und Durchführung der Fallstudie

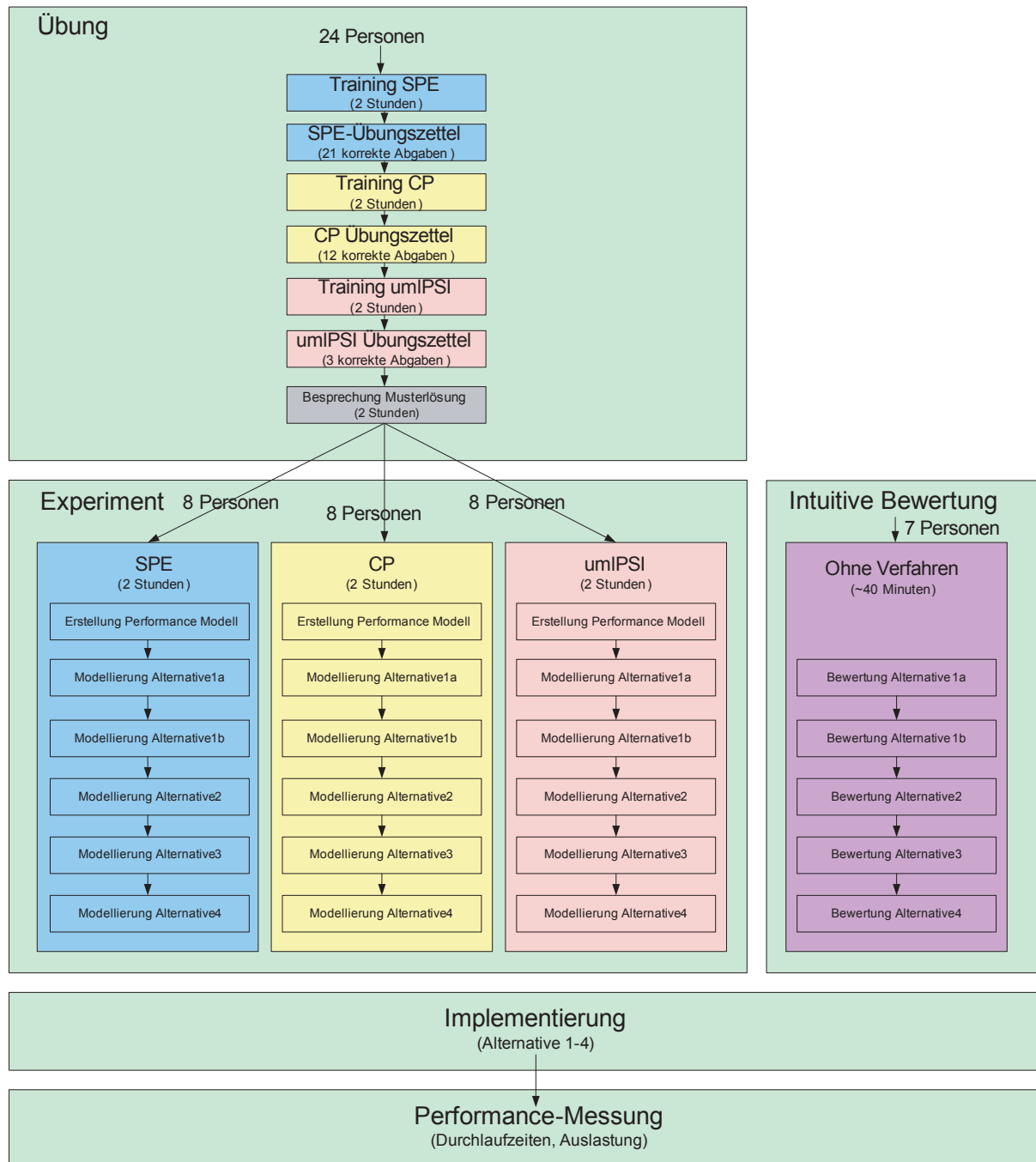


Abbildung 5.1.: Aufbau der Fallstudie

Verfahren auch aktiv anwendeten und sich für das spätere Experiment bereits mit den entsprechenden Werkzeugen vertraut machten. Die Lösungen der Übungszettel wurden eingesammelt und den Teilnehmern korrigiert zurückgegeben. Nach den drei Übungen wurde in einem weiteren zweistündigen Termin eine Musterlösung zu den Aufgaben vorgestellt. Die Aufgabenstellung der Übung und die Auswertung der Lösung werden in Abschnitt 5.4 vorgestellt.

Für das eigentliche Experiment wurde die Versuchsgruppe dann dreigeteilt, so dass jeweils 8 Personen mit genau einem Verfahren die Architektur des Webservers untersuchten. Ein Experimentaufbau, bei dem alle 24 Teilnehmer jeweils alle Verfahren auf die Architektur angewendet hätten, hätte zwar mehr Datenpunkte geliefert, jedoch den Aufwand für die Versuchsteilnehmer verdreifacht. Zusätzlich wäre es zu nicht kalkulierbaren Lerneffekten gekommen. Die Ergebnisse, die eine Testperson mit dem ersten Verfahren erzielt hätte, hätten mit hoher Wahrscheinlichkeit ihre Ergebnisse in einem der anderen Verfahren beeinflusst.

Um eine balancierte Einteilung der drei Gruppen bezüglich der Qualifikation und Leistung der Teilnehmer zu gewährleisten, wurden die Übungslösungen ausgewertet. Dabei wurden die Gruppen so eingeteilt, dass sie jeweils etwa gleich viele Teilnehmer enthielten, die eine überdurchschnittliche, mittlere und unterdurchschnittliche Leistung mit einem Verfahren erzielt hatten. Auf diese Weise wurde versucht zu verhindern, dass die Qualifikation einer Gruppe die Ergebnisse eines Verfahrens maßgebend beeinflusste, wenn z.B. in einer Gruppe wesentlich höher qualifizierte Teilnehmer gewesen wären als in einer anderen.

Das Experiment fand in einem Rechnerraum unter kontrollierten Bedingungen statt, so dass sich die Versuchsteilnehmer nicht untereinander beeinflussen konnten. Die Teilnehmer hatten für das Experiment zwei Stunden Zeit und untersuchten jeweils immer die gleiche Software-Architektur eines Webservers, wobei die Eingabeparameter der Verfahren leicht variierten. Zusätzlich bewerteten sie mit Hilfe der Verfahren fünf unterschiedliche Entwurfsalternativen für den Webserver auf ihre Performance-Eigenschaften und empfahlen jeweils eine der Alternativen für die Implementierung. Details über die Aufgabenstellung und die Durchführung des Experiments finden sich in Abschnitt 5.5. Anmerkungen zur inneren und äußeren Gültigkeit des Aufbaus und zur Kontrolle von Störvariablen des Experiments folgen in Abschnitt 5.7.

Das methodische Vorgehen mit den Verfahren wurde zusätzlich einem intuitiven Vorgehen ohne Verfahren gegenübergestellt. Dazu bearbeiteten sieben weitere Personen, die am Experiment nicht beteiligt waren, die Aufgabenstellung, ohne jedoch die Performance-Analyseverfahren zu kennen. Die von diesen Teilnehmern gelieferten Entwurfsempfehlungen werden in Abschnitt 6.3 mit den Empfehlungen der Teilnehmer des Experiments verglichen.

Alle im Experiment untersuchten Entwurfsalternativen wurden separat für den Webserver implementiert. Durch den komponentenorientierten Aufbau des Servers ließen sich die einzelnen Alternativen relativ unproblematisch umsetzen, indem z.B. eine Cache-Komponente hinzugefügt wurde oder für das Clustering ein Scheduler vor den Webserver geschaltet wurde. Mit diesen Implementierungen war es möglich, die Vorhersagen der Performance-Analyseverfahren mit realen Messungen zu vergleichen. Die Ergebnisse werden in Abschnitt 6.1.4 vorgestellt.

5.3. Versuchsteilnehmer

Häufiger Kritikpunkt an Experimenten in der Softwaretechnik ist, dass die Versuchsteilnehmer oftmals "nur" Studierende sind, und die Ergebnisse somit nicht auf die reale Softwareentwicklung übertragbar seien. Prechelt [Pre01] betont jedoch, dass die Unterschiede zwischen Profis und Studierenden meist gar nicht so groß sind wie zunächst angenommen. Einerseits fallen Studierende meist kurze Zeit später selbst in die Kategorie Profi, andererseits ist die Erfahrung einer Versuchsperson meist gar nicht ausschlaggebend für das Ergebnis des Experiments, sofern die Arbeitsweise der Teilnehmer gleich ist. Dabei muss das Experiment so entworfen werden, dass die Testpersonen nicht überfordert werden.

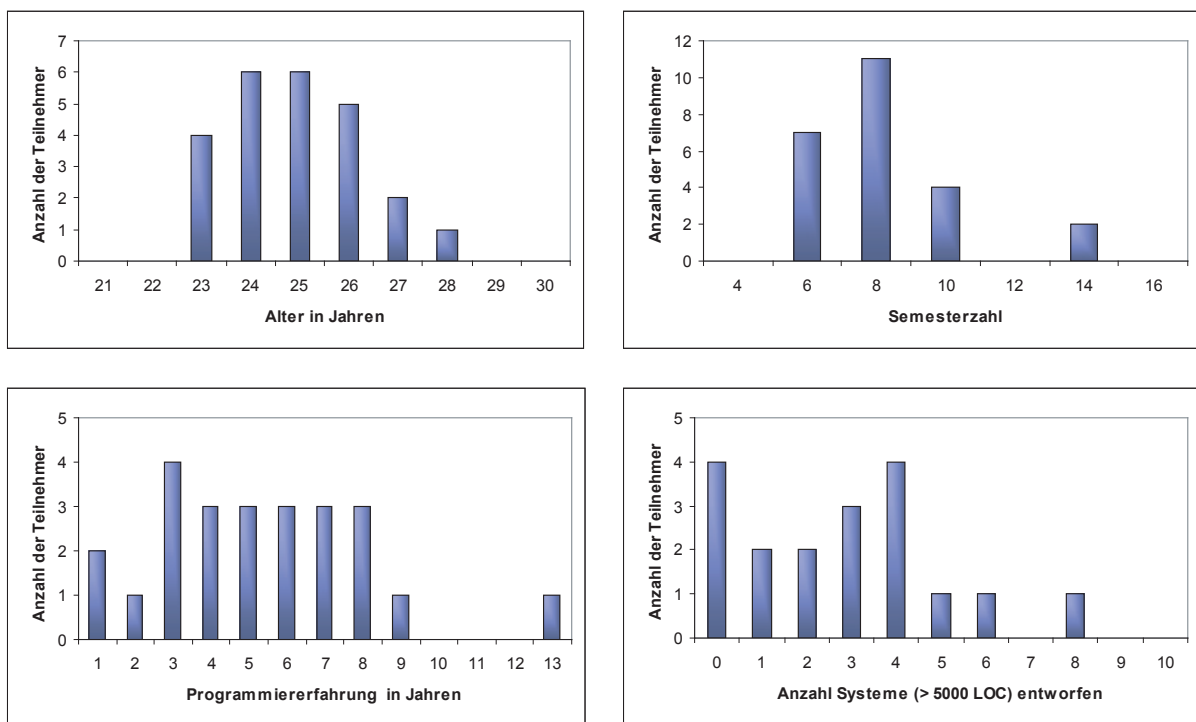


Abbildung 5.2.: Informationen über die Versuchsteilnehmer

In dieser Fallstudie waren die Teilnehmer keine Anfänger mehr, sondern hatten alle ihr Informatik-Vordiplom abgeschlossen und bereits Entwurfs- und Programmiererfahrung in einem erfolgreich absolvierten Software-Praktikum erworben. Um die Teilnehmer weiter zu charakterisieren wurde beim Experiment ein Fragebogen zur Erfassung von Alter und Programmiererfahrung verteilt. Die Ergebnisse (Abbildung 5.2) zeigen, dass die Studenten

- zwischen 23 und 28 Jahre alt sind,
- im Hauptstudium studieren,
- bereits mehrjährige Programmiererfahrung besitzen und
- zum großen Teil schon mehrere Systeme mit mehr als 5000 Zeilen Programmiercode entworfen haben.

Somit sind sie mit Entwürfen in UML vertraut und sollten in der Lage sein, aufgrund ihrer Informatik-Erfahrung gewisse Performance-Werte abzuschätzen. 80 Prozent der Teilnehmer gaben an, noch überhaupt keine Erfahrung mit Performance-Analysen gemacht zu haben. Vier Personen hatten bereits kleinere Systeme auf Performance-Probleme untersucht. Ein Teilnehmer hatte in seiner Nebentätigkeit größere Systeme untersucht, gab jedoch auch an, dass er die untersuchten Performance-Analyseverfahren dabei nicht verwendet hatte. Der Kenntnisstand bzgl. der Verfahren war also bei allen Teilnehmern vor Beginn der Übungen gleich hoch.

Aufgrund ihres fortgeschrittenen Informatikstudiums und der langjährigen Programmiererfahrung, kann somit durchaus von einer soliden Grundkompetenz der Teilnehmer gesprochen werden, die der Qualifikation von Profis in der Praxis schon recht nahe kommt.

Eine Bildung von Teams aus z.B. 2-3 Teilnehmern für die Performance-Analysen wurde als nicht sinnvoll erachtet, da dies die ohnehin schon geringe Anzahl der Datenpunkte noch weiter reduziert hätte.

5.4. Übung: (Einführung und Vortest)

Ein Vortest ist eine Aufgabe, die alle Teilnehmer vor dem eigentlichen Kernexperiment absolvieren, so dass sowohl die Aufgabenstellung des Experiments als auch die Leistung der Teilnehmer getestet werden kann. Auf diese Weise werden Fehler in der Aufgabenstellung vermieden und Lerneffekte während des Experiments reduziert. Ein Vortest mit exakt der gleichen Aufgabenstellung wie im Experiment war in dieser Studie nicht durchführbar, da dann die Teilnehmer des Vortests nicht mehr am Experiment hätten teilnehmen können und sich die Anzahl der Ergebnisse reduziert hätte. Das Einbeziehen von weiteren Testpersonen für einen Vortest erschien aufgrund des nicht unerheblichen Lernaufwands für die Verfahren als nicht realistisch. Daher stellte in dieser Fallstudie die Bearbeitung der Übungszettel den Vortest für das Experiment dar. Die Aufgabenstellungen auf den Übungszetteln waren den Experimentaufgaben recht ähnlich und somit konnten aus den Ergebnissen Aussagen bzgl. des Umfangs und der Art der Aufgabenstellungen abgeleitet werden. Zusätzlich wurden Punkte für die Lösungen vergeben, womit die Leistung der Teilnehmer für die spätere Gruppeneinteilung beim Experiment bewertet wurden.

5.4.1. Aufgabenstellung

Die kompletten Übungszettel können im Anhang C eingesehen werden.

Für das SPE-Verfahren sollte nach der Durcharbeitung des Handbuchs für das Werkzeug SPE-ED (Aufgabe 1) der einfache Entwurf eines Bibliothekssystem analysiert werden (Aufgabe 2). Das System ähnelte in seiner Architektur bereits dem Webserver, welcher später im Experiment untersucht wurde. Um eine zu hohe Abweichung der Ergebnisse zu vermeiden, wurden Vorgaben bzgl. der Service-Zeiten von Hardware-Elementen gemacht (der "Processing Overhead" wurde vorgegeben). Für die Modellierung mit dem SPE-ED Werkzeug wurde ein erweitertes Sequenzdiagramm vorgegeben, das Schleifen und Ausführungsalternativen enthielt. Die Teilnehmer sollten nach der Performance-Analyse des Systems auch eine Erweiterung durch eine Dual-CPU und zwei Festplatten testen. Somit sollten bei sorgfältiger Lösung dieser Aufgaben Kenntnisse über das SPE-ED Werkzeug, über die Erstellung von Ausführungsgraphen aus den

erweiterten Sequenzdiagrammen, über die Abschätzung von Performance-Werten und die Bewertung von Entwurfsalternativen erlangt worden sein, so dass die Teilnehmer genau auf die Experimentaufgabe vorbereitet waren.

Beim CP-Verfahren wurden drei Aufgaben zur Analyse eines Datenbankservers gestellt, die analog zu dem auf den Folien dargestellten Beispiel waren. Die Aufgabenstellung wurde aus [MAD04] Kapitel 5 entnommen und diente dazu, die Teilnehmer mit der Verwendung einer Tabellenkalkulation, der Analyse der Systemlast sowie der Berechnung der Eingabeparameter für Warteschlangenmodelle anhand der vorgestellten Formeln vertraut zu machen.

Für das umlPSI-Verfahren wurde ein Übungszettel erstellt, bei dem die Teilnehmer ein E-Commerce System zunächst aus einer textuellen Beschreibung heraus mit UML modellieren sollten. Dabei sollten die Teilnehmer das Werkzeug Poseidon einsetzen, welches auch später während des Experiments verwendet wurde. Hierbei musste Poseidon in der schon zur Durchführung des Experiments nicht mehr aktuellen Version 1.41 verwendet werden, da umlPSI in der vorliegenden Version nur solche XMI-Dateien auswerten kann, die genau von diesem Werkzeug erstellt wurden. In den späteren Versionen wurde der XMI-Export verändert, diese Dateien können von umlPSI nicht geparkt werden. Ebenfalls wäre die Verwendung des (freien) Werkzeugs ArgoUML 0.12 möglich gewesen, welches auch kompatibel mit umlPSI ist. Dies wurde allerdings verworfen, da sich hier die Eingabe von Stereotypes und Tagged-Values als umständlich und fehlerhaft erwies. Nach der Modellierung der Architektur des E-Commerce Systems mit Anwendungsfall-, Aktivitäts- und Verteilungsdiagrammen sollten diese in der zweiten Aufgabe anhand des UML Performance Profiles mit Stereotypen und Tagged Values versehen werden. Die Dauer für einzelne Aktionen musste hier abgeschätzt werden. Abschließend sollte in der dritten Aufgabe umlPSI gestartet werden und die Modellierung simuliert werden.

5.4.2. Auswertung der Übungslösungen

Insgesamt wurden von 24 Teilnehmern Lösungen der Übungszettel eingereicht. Bei einigen Lösungen waren Teilaufgaben nicht bearbeitet worden, teilweise wurden identische Lösungen abgegeben, obwohl gefordert war, dass die Zettel von Einzelpersonen bearbeitet werden sollten. Einige Lösungen wurden mit großer Sorgfalt angefertigt, andere waren aufgrund von Unvollständigkeiten kaum verwertbar. Trotzdem können einige Beobachtungen gemacht werden:

SPE

Für die Architektur des Bibliothekssystems wurden die Durchlaufzeiten und Ressourcenauslastungen mit dem SPE-ED-Tool gemessen. Laut Aufgabenstellung sollte ein Anwendungsszenario modelliert werden, bei dem Benutzer nacheinander 10 Suchanfragen an zwei verschiedene Datenbanksysteme stellen und sich danach zu einem der Suchergebnisse detaillierte Informationen anzeigen lassen. Bei den abgegebenen Lösungen fällt zunächst die große Spannweite der gemessenen Durchlaufzeiten auf (Abbildung 5.3, oben links). Viele Teilnehmer haben anscheinend nur die Dauer für die Einzelschritte des Kontrollflusses abgeschätzt, aber nicht die Gültigkeit des Gesamtergebnisses kontrolliert. So kommen bei einigen Modellierungen Durchlaufzeiten von unter 10 Sekunden zustande, in denen kein Benutzer realistischerweise 10 Suchanfragen eintippen könnte. Einigen Teilnehmern war die Bedeutungen der Software-Ressourcen (CPU, Disk, Delay,

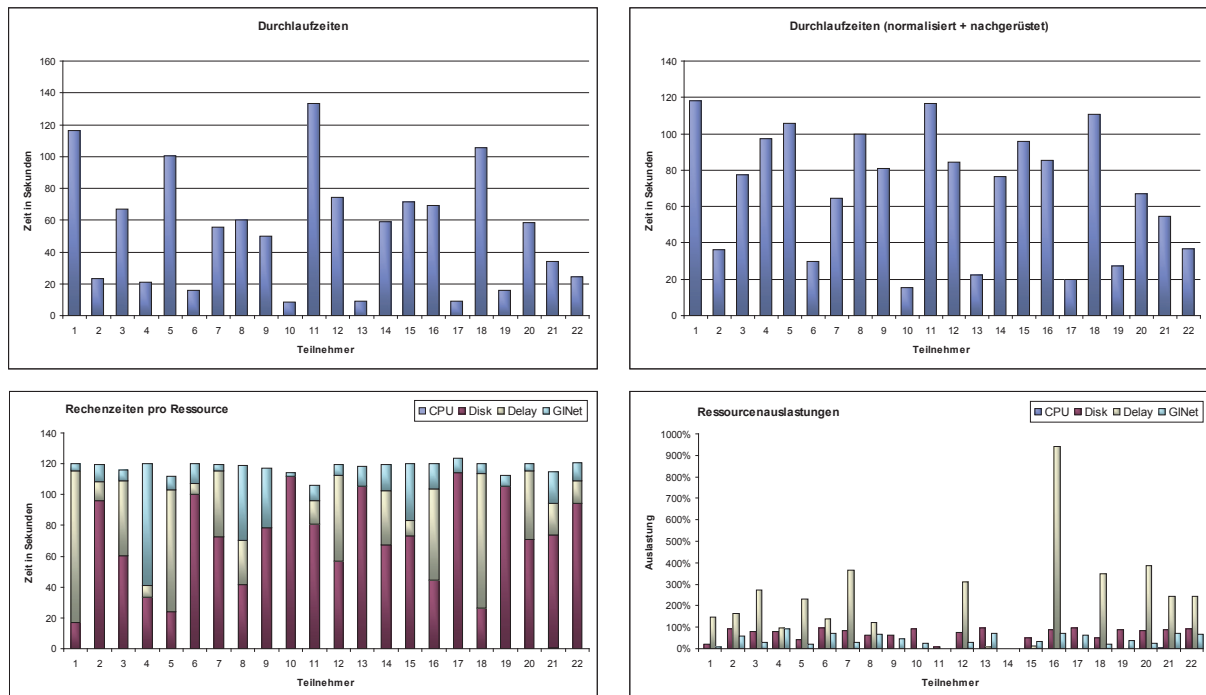


Abbildung 5.3.: SPE-Ergebnisse

GINet) noch unklar, obwohl diese in der Übung gesondert erklärt worden waren. Insbesondere das Abschätzen der Dauer der Benutzereingaben, für die die Software-Ressource "Delay" verwendet werden sollte, fiel den meisten Teilnehmern offensichtlich sehr schwer.

Um die Ergebnisse zu normalisieren, wurde bei der Auswertung der Lösungen die Ankunftsrate der Benutzer für dieses Szenario so eingestellt, dass sich die Gesamtdurchlaufzeit aufgrund der konkurrierenden Ressourcennutzung auf ca. 120 Sekunden steigerte (Abbildung 5.3, unten links). Hier fällt auf, dass die Teilnehmer die Rechenzeit für die unterschiedlichen Ressourcen sehr unterschiedlich abgeschätzt haben. Bei einigen Modellierungen dominieren Zugriffe auf die Festplatte die Durchlaufzeit, bei anderen wird bei den Benutzereingaben die meiste Zeit verbraucht. Dies spiegelt sich auch in den Ressourcenauslastungen (Abbildung 5.3, unten rechts) wieder.

Als letzter Teil der Aufgabe sollte untersucht werden, wie sich eine Änderung der Hardware (Einbau einer zweiten CPU und einer zweiten Festplatte) auf die Performance auswirkt. Dazu musste lediglich der "Processing Overhead" im SPE-ED Werkzeug geändert werden. Die Ergebnisse sind in (Abbildung 5.3, oben rechts) zu sehen. Bei einigen Lösungen, nämlich solchen bei denen diese Ressourcen besonders stark belastet wurden, zeigte die Änderung der Hardware eine große Auswirkung. Bei anderen Lösungen, bei denen die Benutzereingaben die Durchlaufzeit des Szenarios dominierten, führten diese Änderungen hingegen kaum zu Verbesserungen.

Teilweise enthielten die Lösungen der Teilnehmer auch offensichtlich falsche Werte, wobei z.B. die Dauer der Abfrage der lokalen Datenbank als länger abgeschätzt wurde, als die Abfrage der über das Internet angesprochenen Datenbank.

Für die gegebene Architektur wurden entsprechend der Aufgabenstellung eine Reihe von Ver-

besserungsvorschlägen gemacht. Darunter waren z.B. die Aufrüstung der Hardware, die Parallelisierung von Datenbank-Abfragen, die Verwendung von Caches sowie die Verwendung von Cookies zur Beschleunigung des Logins. Hier waren die Teilnehmer teilweise sehr ideenreich und zeigten, dass sie trotz ihrer Unerfahrenheit mit Performance-Analysen durchaus eine Reihe von Vorschlägen zur Geschwindigkeitsverbesserung machen konnten.

Capacity Planning

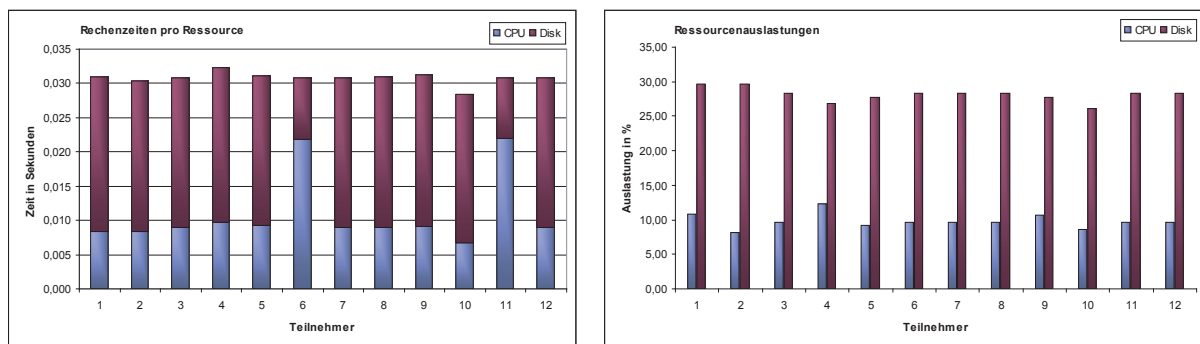


Abbildung 5.4.: CP-Ergebnisse

Für den Übungszettel zum "Capacity Planning" waren nur 12 der abgegebenen 23 Lösungen vollständig korrekt. Bei den meisten der fehlerhaften Lösungen wurde vergessen, die Ressourcenauslastungen für die einzelnen Systemlastklassen analog zum Beispiel auf Übungsfolien zu faktorisieren.

Die Lösungen für die Rechenzeiten und Auslastungen liegen sehr eng beieinander (Abbildung 5.4), weil die Berechnung in diesem Verfahren so gut wie gar nicht von Schätzungen beeinflusst wird. Lediglich die Mittelpunkte der Systemlastklassen sollten abgeschätzt werden, dabei waren die Ergebnisse der Teilnehmer bei den abgegebenen Lösungen fast identisch, da sich die Werte aus einem Diagramm ablesen ließen. Alle anderen Ergebnisse bei diesem Verfahren konnten aus den Eingabeparametern berechnet werden. Eine Musterlösung befindet sich im Anhang.

umlPSI

Dieses Aufgabenblatt wurde von 21 Studenten abgegeben. Die Modellierung des E-Commerce Systems in UML mit Poseidon (Aufgabe 1) gelang dabei allen Teilnehmern zufrieden stellend. Die Annotation der UML-Diagramme mit Stereotypen und Tagged Values (Aufgabe 2) war jedoch bei fast allen Teilnehmern fehlerhaft. Dabei kamen sehr häufig Syntaxfehler vor, weil sich die Teilnehmer nicht exakt an die Vorgaben auf den Folien hielten, obwohl es dort viele Beispiele gab. Bei der Abschätzung von Performance-Daten wurden teilweise sehr hohe und unrealistische Werte angegeben. Einige Lösungen enthielten jedoch durchaus realistische Werte.

Auf dem Rechnerpool der Universität konnte umlPSI nicht kompiliert werden, da es das Betriebssystem Linux voraussetzt und auf den dortigen Rechnern FreeBSD verwendet wird. Deshalb wurden die Teilnehmer gebeten, dieses im Internet frei verfügbare Werkzeug privat auf ihren eigenen Rechnern zu installieren. Falls dies nicht möglich sei, sollten die Teilnehmer zumindest ihre Poseidon-Projektdateien einschicken, damit diese Lösungen nachträglich simuliert werden

konnten. Unerwartet war, dass die Installation und Ausführung des umlPSI-Werkzeugs unter Linux keinem (!) der Teilnehmer gelang, obwohl dies beim Entwurf der Aufgabenstellung auf mehreren Rechnern getestet worden war. Trotzdem reichten 21 Teilnehmer Poseidon-Projektdateien ein.

Deren Simulation wurde nach Abschluss der Übung auf einem privaten Rechner durchgeführt. Wegen der oben angesprochenen Syntaxfehler bei der Beschriftung der UML-Diagramme, konnten jedoch die meisten Lösungen nicht von umlPSI geparkt werden. Auch bei nachträglich korrigierten und erfolgreich geparkten Lösungen lieferten die Werkzeuge eine Reihe von Fehlern, deren Ursache nicht gefunden werden konnte. Insgesamt konnten nur drei der abgegebenen Lösungen simuliert werden.

5.4.3. Schlussfolgerungen aus den Übungen

Aus der Korrektur der Übungen konnten einige Erkenntnisse gezogen werden. Die Schätzungen beim SPE-Verfahren fielen dermaßen unterschiedlich aus, dass für das Experiment mehr Vorgaben bzgl. der Performance-Daten gemacht werden mussten. In der Praxis wird man bei solchen Performance-Analysen ebenfalls versuchen, möglichst wenige Werte abzuschätzen und möglichst viele Werte mit Messungen zu bestimmen. So gesehen sollten mehr Vorgaben die Anwendung des Verfahrens nicht verfälschen. Die Teilnehmer mussten in der Aufgabenstellung des Experiments nochmals explizit dazu aufgefordert werden ihre Ergebnisse auf Validität zu prüfen, damit die Inkonsistenzen und offensichtliche Fehler in den Ergebnissen reduziert werden. Weiterhin wurden in der Aufgabenstellung die Bedeutung der einzelnen "Software Ressourcen" genau erklärt, damit hier keine Unklarheiten entstehen konnten.

Beim "Capacity Planning" stimmten zwar nur bei 12 Abgaben die Endergebnisse mit der Musterlösung vollständig überein, jedoch waren bei den fehlerhaften Abgaben meist nur kleinere Rechenfehler zu bemängeln. Da das Verfahren eine Reihe von aufeinander folgenden Rechenschritten beinhaltet, pflanzten sich diese Fehler bis zum Endergebnis fort, obwohl die späteren Schritte jeweils richtig bearbeitet wurden. Auf die Rechenfehler und anderen kleineren Mängel wurde bei der Vorstellung der Musterlösung deutlich hingewiesen.

Als Folgerung der Probleme mit dem umlPSI-Werkzeug musste auf eine Simulation der annotierten UML-Diagramme während des Experiments verzichtet werden. Diese konnte erst nach Abschluss des Experiments erfolgen, weil für die anderen Verfahren Windows-Rechner benötigt wurden und eine ausreichende Zahl von zusätzlichen Linux-Systemen nicht zur Verfügung stand. Daraus resultierend konnte auch die Entscheidung für eine Entwurfsalternative erst nach Ende des Experiments getroffen werden, als die Ergebnisse der Simulationen vorlagen. Weiterhin wurde die Modellierung der Architektur in UML in der Aufgabenstellung vorgegeben, um den Zeitaufwand während des Experiments zu verringern. Die Teilnehmer mussten lediglich für die Modellierung der Entwurfsalternativen leichte Veränderungen an den Diagrammen vornehmen. Um Syntaxfehler bei der Eingabe der Tagged-Values wie bei den Übungen zu vermeiden, wurden diese mit einer Dauer von jeweils Null Sekunden vor Beginn des Experiments in die Diagramme eingetragen. Die Teilnehmer mussten dann nur noch die Dauer einzelner Aktionen abschätzen und die Werte in die vorbereiteten Diagramme eintragen.

Die Einteilung der Gruppen des Experiments erfolgte anhand der Punktevergabe für die Übungszettel. Für jeden Übungszettel wurden maximal 7,5 Punkte vergeben, wobei nicht die

Schätzungen der Teilnehmer bewertet wurden, sondern die korrekte Anwendung der Verfahren.

SPE	CP	umlPSI
7,5	7,5	7,5
7,5	7,5	7,5
7,5	7,5	7,5
6	7,5	6
5	7	6
5	6	5
4	6	5
2	5	3,5

Tabelle 5.1.: Punkte der Teilnehmer des Experiments in der Übung

Die Gruppen wurden bewusst so eingeteilt, dass ungefähr genauso viele Teilnehmer mit über- bzw. unterdurchschnittlichen Leistungen in der Übung in einer Gruppe waren. Hätte man die Gruppen so eingeteilt, dass nur die Teilnehmer einem Verfahren zugeordnet werden, die in diesem Verfahren in den Übungen eine besonders gute Leistung erbracht haben, so wären die Ergebnisse des Experiments nicht mehr repräsentativ für alle Entwickler. Die Punktzahl bei den CP-Teilnehmern ist im Durchschnitt höher als bei den anderen Verfahren, da es bei dieser Übung bei den meisten Teilnehmern nur einen niedrigen Punktabzug gab. Bei der umlPSI-Übung erhielten auch solche Teilnehmer die volle Punktzahl, die die Lösung zwar nicht selbst simuliert hatten, aber zumindest eine korrekte Poseidon-Projektdatei abgegeben hatten.

5.5. Experiment

5.5.1. Webserver-Architektur

Untersuchungsobjekt im Experiment war ein experimenteller Webserver, der während einer studentischen Nebentätigkeit in der Palladio-Gruppe zu Testzwecken entwickelt worden war. Die Implementierung war in einer ersten Version des Servers rein objektorientiert auf Basis des .NET-Frameworks unter dem Microsoft Visual Studio durchgeführt worden. Im Rahmen eines Individuellen Projekts (einer viermonatigen Studienarbeit) wurde der Server in mehrere austauschbare Komponenten zerlegt und Schnittstellen zur Anbindung von externen Applikationen und Datenbanken geschrieben [Tei04]. Damit konnte der Webserver als Beispielsystem für ein komponentenorientiertes System von der Palladio-Gruppe für Testzwecke genutzt werden.

Der Server nimmt Client-Anfragen nach dem HTTP-Protokoll entgegen und liefert die geforderten Seiten (z.B. HTML- oder Bilddateien) zurück. Dabei ist er multi-threaded, d.h. er spaltet für jede Anfrage einen neuen Programmthread ab, so dass mehrere Dateien parallel bezogen werden können. HTML-Dateien können vom Server auch dynamisch erzeugt werden, um beispielsweise benutzerindividuelle Informationen einzubetten. Alle Seitenanfragen werden in einer Logdatei gespeichert, so dass Analysen über die Benutzer bzw. Zugriffsprofile erstellt werden können.

Der Webserver umfasst ca. 4000 Zeilen Quellcode in der Programmiersprache C# [Lib03]. Bei der Entwicklung wurde speziell auf die spätere Erweiterbarkeit der Architektur geachtet.

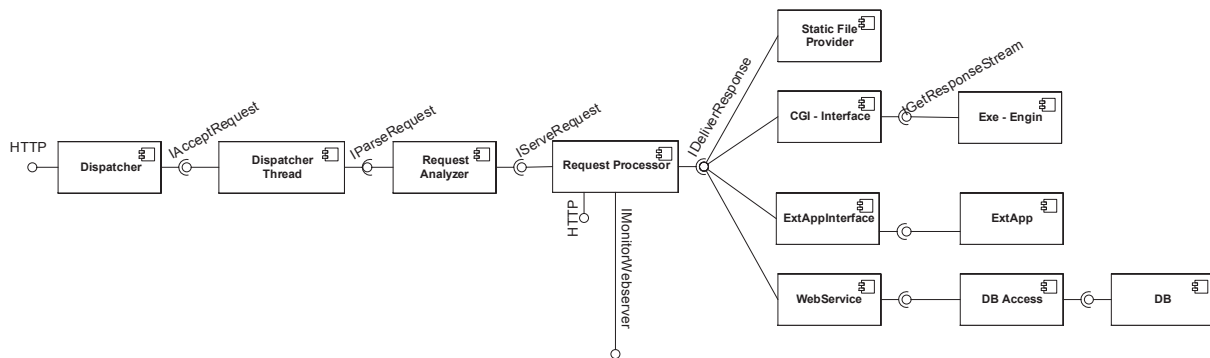


Abbildung 5.5.: Komponentendiagramm des Webservers

Somit können zusätzliche Protokolle und Komponenten zur Generierung von Dateien leicht nachträglich in den Server integriert werden. Die Komponenten des Webservers (Abbildung 5.5) sind im Einzelnen:

- **Dispatcher:** nimmt Client-Anfragen an einem offenen Socket entgegen
- **DispatcherThread:** startet für jede Anfrage einen neuen Thread
- **RequestAnalyser:** überprüft die Korrektheit der Anfrage, bestimmt das entsprechende Protokoll (HTTP, FTP, ICMP), parst die Anfrage, bestimmt die angefragte Seite
- **RequestProcessor:** sorgt für das Beziehen der Seite vom Server, sendet Rückantwort an den Client, enthält Klassen zur Fehlerbehandlung und zur Überwachung des Servers
- **FileProvider:** bezieht eine Datei aus dem lokalen Dateisystem
- **CGIInterface:** stellt eine Schnittstelle für externe Applikationen zur Verfügung, die mit dem CGI kommunizieren können
- **ExeEngine:** startet bei Anfragen für dynamische Seiten externe Applikationen auf dem Server, die als ausführbare Dateien vorliegen
- **Internal-API-Application:** stellt eine Schnittstelle für weitere Programme dar, die auf dem Server gestartet werden können, entspricht jedoch keinem Standard
- **Webservice:** ermöglicht die Kommunikation mit Webservices nach dem SOAP-Protokoll
- **DBAccess:** stellt eine Schnittstelle für Datenbanken dar

Anfragen werden vom **RequestProcessor** über das **IDeliverResponse**-Interface an die weiteren Komponenten zur Beantwortung übergeben. Dabei sind diese Komponenten nach dem Entwurfsmuster "Chain of Responsibility" [GHJV95] angeordnet und behandeln die Anfrage sofern möglich selbst oder leiten sie andernfalls an die nächste Komponente in einer Kette weiter. Kann keine Komponente die Anfrage beantworten, so liefert der **RequestProcessor** eine Fehlermeldung zurück. Näheres zum Webserver findet sich in [Tei04]. Die Ergebnisse der Performance-Analysen der Implementierung können in Abschnitt 6.1.4 eingesehen werden.

5.5.2. Aufgabenstellung

Für das Experiment musste nun eine Aufgabenstellung gefunden werden, die zur Beantwortung der in Kapitel 3 formulierten Forschungsfragen geeignet war. Dazu sollte als typischer Anwendungsfall der Performance-Analyseverfahren die Bewertung von Entwurfsentscheidungen im Vordergrund stehen. Die absolute Performance-Messung des Webservers ist mit den untersuchten Verfahren zwar auch möglich, in diesem Experiment stand die Implementierung den Teilnehmern jedoch nicht zur Verfügung, da gerade die frühe Performance-Analyse zum Entwurfszeitpunkt untersucht werden sollte.

Daher orientierte sich die Aufgabenstellung an den zur Verfügung stehenden Entwurfsdokumenten des Webservers. Es lag für jede Entwurfsalternative ein UML-Komponentendiagramm vor. Diese Diagramme mussten nun noch zur Verwendung im Experiment so aufbereitet werden, dass sie sich direkt als Eingabeparameter für die Verfahren eigneten und die Teilnehmer des Experiments möglichst wenig Routineaufgaben zu erledigen hatten. Folgende Artefakte wurden dazu erstellt:

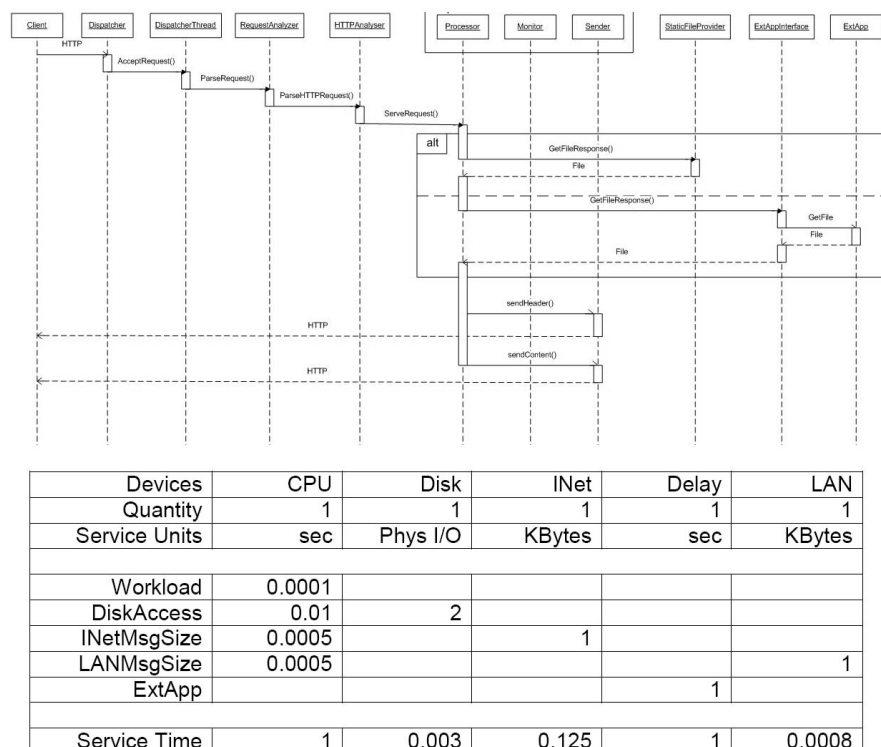


Abbildung 5.6.: Eingabeparameter SPE

- **SPE:** auf Basis der Komponentendiagramme und des Quellcodes wurden für jede der Entwurfsalternativen die von Smith und Williams eingeführten erweiterten Sequenzdiagramme erstellt (Abbildung 5.6). Diese enthielten Schleifen und Ausführungsalternativen für verschiedene Möglichkeiten des Kontrollflusses durch den Webserver. Damit ließen sich die benötigten Ausführungsgraphen mit dem SPE-ED Werkzeug leicht erstellen. Die "Software Ressourcen" wurden vorgegeben und aufgrund der Erfahrung bei den Übungen

nochmals ausführlich erklärt. Der "Processing Overhead" (eine Tabelle, in der die Anforderung der Software-Ressourcen auf Hardware-Ressourcen abgebildet werden) wurde ebenfalls vorgegeben. Die Vorgabe dieser Werte entspricht der gängigen Praxis, denn im Verfahren ist vorgesehen, dass diese Werte von Performance-Experten geliefert werden. Die Werte wurden durch Messungen bestimmt bzw. für die Netzwerkverbindungen mit typischen Werten (ISDN-Verbindung zum Client, 10 MBit-Anbindung zwischen Webserver und externem Server) belegt. Diese Hardwareumgebung wurde dann später auch bei den Performance-Messungen am Webserver nachgebildet. Zusätzlich wurde ein Warteschlangenmodell zur Verteilung der Ressourcen angegeben, das allerdings nur der Illustration und dem Verständnis diente und im SPE-ED Werkzeug nicht verwendet werden konnte.

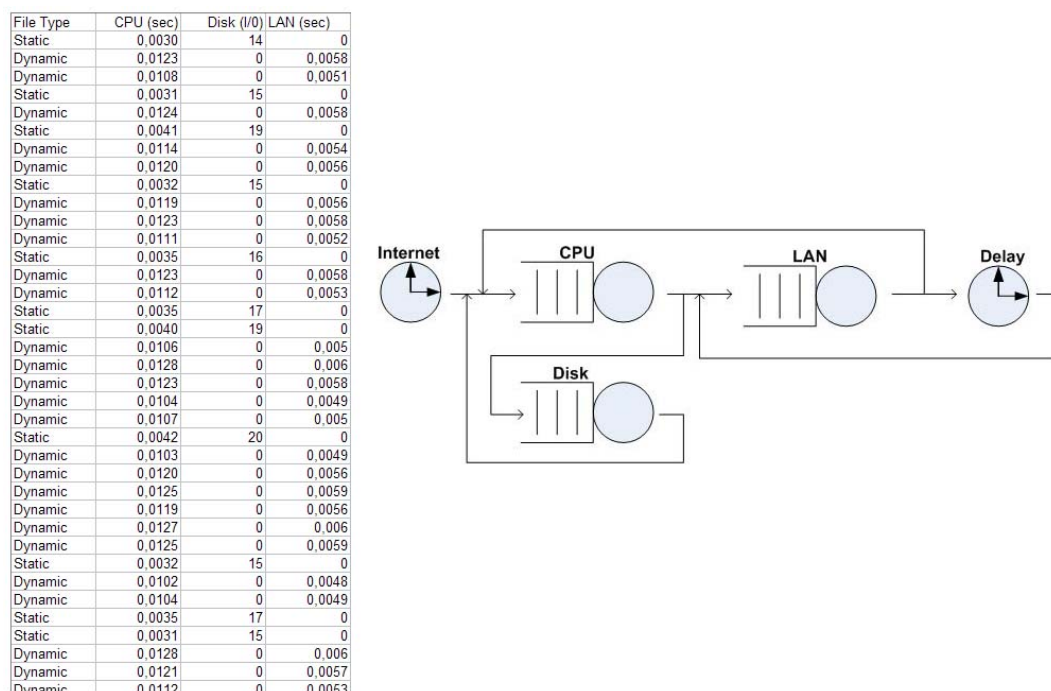


Abbildung 5.7.: Eingabeparameter Capacity Planning

- **Capacity Planning:** Es lag kein realistisches Benutzungsprofil vor, das als Startpunkt für das Capacity-Planning Verfahren hätte dienen können. Daher wurde für diese Methode mit Microsoft Excel "künstlich" eine Logdatei erzeugt, die die CPU-Zeit, die Anzahl der Festplattenzugriffe und die Bearbeitungszeiten des LANs für 1000 Zugriffe auf den Webserver dokumentierte (Abbildung 5.7). Die Werte wurden abgeschätzt, wobei auf ein realistisches Verhältnis der Performance der Ressourcen geachtet wurde. Somit sollten sich die Berechnungen, die auf dieser Basis angestellt werden, für das Füllen einer Entwurfsentscheidung ausreichende Ergebnisse liefern. Dabei konnte jedoch erwartet werden, dass die absoluten Werte von den späteren Messungen an der Implementierung abweichen. Zusätzlich wurden die Auslastungen für die Ressourcen CPU, Festplatte und LAN sowie die Übertragungszeit zum Client und die Berechnungsdauer des externen Server vorgegeben. Die Auslastungen wurden anhand von Messungen bei der Ausführung des Prototyps abgeschätzt. Die Vorgabe dieser Werte war für die späteren Berechnungen notwendig und

entspricht den Beispielen aus [MAD04].

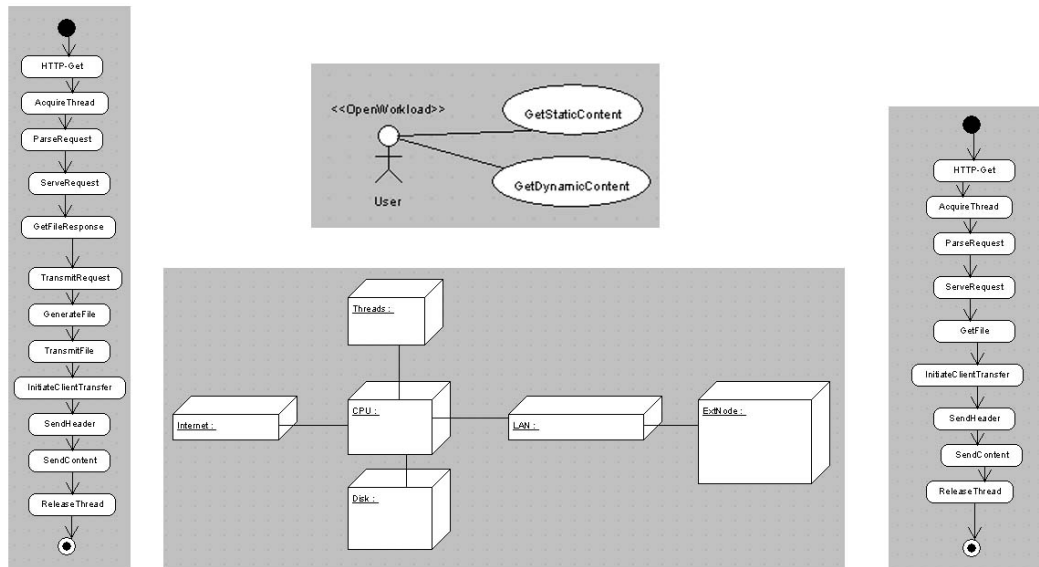


Abbildung 5.8.: Eingabeparameter umlPSI

- **umlPSI:** für dieses Verfahren wurde eine Poseidon-Projektdatei erstellt, die bereits die benötigten Anwendungsfall-, Aktivitäts- und Verteilungsdiagramme für die Architektur des Webserverns enthielt (Abbildung 5.8). Diese Vorgabe wurde gemacht, um den Aufwand für die Modellierung zu reduzieren und mehr Zeit für die Performance-Analyse zu sparen. Ähnliche Diagramme sollten auch in der Praxis bereits vorhanden sein, wenn mit der Performance-Analyse begonnen wird. Das Anwendungsfalldiagramm enthielt Anwendungsfälle für die Abfrage einer statischen und für die Abfrage einer dynamischen HTML-Datei. Die zugehörigen Aktivitätsdiagramme wurden mit Hilfe der Komponentendiagramme und der bereits fertigen Implementierung erstellt. Sie ähnelten vom Aufbau her den beim SPE benutzten Sequenzdiagrammen. Das Verteilungsdiagramm verwendete die gleichen Ressourcen wie das Warteschlangenmodell, das für das SPE-Verfahren und das Capacity Planning vorgegeben wurde. Die für das umlPSI-Werkzeug benötigten Stereotypen und Tagged-Values entsprechend des UML Performance Profiles wurden nach den Erfahrungen aus den Übungen mit Nullwerten vorgegeben.

Die komplette Aufgabenstellung (siehe auch Anhang E) schließlich enthielt:

1. den Anweisungstext (mit einer Schritt-für-Schritt Anleitung angepasst an jedes Verfahren als Hilfe für die Teilnehmer),
2. die oben genannten Softwareartefakte für das jeweilige Verfahren,
3. Unterstützungsmaterialien in Form der Folien der Übungen sowie des SPE-ED Handbuchs und der Grammatiken für die Tags des UML Performance Profiles und
4. Fragebögen zur Erfassung persönlicher Daten und weiteren Kommentaren

Die Aufgabenstellung folgte einem Extra-Subjekt-Entwurf [Pre01], d.h. jeder Teilnehmer löste jeweils nur eine Aufgabe (mit fünf ähnlichen Teilaufgaben) mit einem bestimmten Verfahren, somit waren die Ergebnisse der Gruppen disjunkt. Im Gegensatz dazu würden die Testpersonen bei einem Intra-Subjekt-Entwurf mehrere Aufgaben lösen, also die Untersuchung mit jeweils mehreren Verfahren durchführen. Dies hätte mehr Datenpunkte für die Analyse geliefert, jedoch den Aufwand für die Teilnehmer deutlich erhöht. Weitere Vorteile beim Extra-Subjekt-Entwurf sind, dass in der zur Verfügung stehenden Zeit eine größere Aufgabe bearbeitet werden kann und es bei den späteren Aufgaben keine Reihenfolgeeffekte wie Ermüdung oder Hinzulernen gibt, die das Ergebnis verfälschen könnten.

Ein Blindversuch, wie er in der Medizin häufig gemacht wird, wurde hier nicht durchgeführt. Bei einem solchen Versuch erfahren die Teilnehmer nicht, welcher Behandlung sie ausgesetzt werden bzw. welchem Ziel das Experiment dient. Auf diese Weise wird verhindert, dass sich das Verhalten der Versuchsteilnehmer im Hinblick auf die Erreichung des Ziels unnatürlich verändert. Das Ziel des Experimentes konnte den Teilnehmern nicht vorenthalten werden, da den Teilnehmern klar war, dass hier eigentlich nicht die Performance eines Webservers untersucht wurde, sondern Erkenntnisse über die einzelnen Verfahren gewonnen werden sollten.

Für die Aufgabenstellung wurde eine maximale Bearbeitungszeit von zwei Stunden festgelegt. Dies half einerseits bei der Organisation des Experiments, da die drei Gruppen das Experiment jeweils direkt im Anschluss durchführten und nur acht Rechner zur Verfügung standen. Andererseits entspricht dies auch der Praxis in der Softwareentwicklung, wo normalerweise auch unter Zeitdruck mit festen Abgabeterminen gearbeitet wird. Beim vorherigen Testen der Aufgaben erschien die Zeit als ausreichend bemessen. Ein Abnahmetest der Lösungen konnte aufgrund der begrenzten Rechnerzahl nicht während des Experiments durchgeführt werden und erfolgte nachträglich. Näheres dazu in Abschnitt 5.5.4

Der zweite Hauptkritikpunkt an den Experimenten in der Softwaretechnik neben den meist studentischen Versuchsteilnehmer ist die geringe Komplexität der Aufgabenstellung. Praktiker kritisieren häufig, dass der Umfang der Aufgaben nicht mit der realen Softwareentwicklung zu vergleichen sei und das Experiment daher nur geringe Aussagekraft besitze. Dagegen zu halten ist, dass auch in der realen Softwareentwicklung viele Aufgaben mit geringer Komplexität vorkommen, und dass Ergebnisse, die aus einer einfachen Aufgabenstellung resultieren, sich qualitativ (wenn auch nicht quantitativ) durchaus auch auf komplexere Aufgaben übertragen lassen [Pre01]. Eine einfache Aufgabe ist dann gut gewählt, wenn sie nur Aspekte enthält, die für die Beantwortung der Experimentfrage relevant sind und andere Elemente ausblendet. Beim Entwurf des Webservers und der Aufgabenstellung wurde daher darauf geachtet, dass möglichst nur diejenigen Elemente betrachtet werden, die für die Performance-Analysen wichtig sind. Aspekte wie das Monitoring von Benutzeranfragen oder die Anbindung verschiedener Protokolle wurden in der Aufgabenstellung ignoriert, da sie keinen oder nur einen geringen Anteil an der Gesamtleistung des Systems ausmachen. Die Anbindung einer Datenbank an den Webserver wurde hingegen absichtlich eingeführt, um die Performance-Analyse komplexer und weniger offensichtlich zu gestalten. Diese Two-Tier-Architektur mit Webserver und nachgeschalteter Datenbank auf einem separaten Rechner ist zwar einfach, aber in der Praxis durchaus gängig. Bei der Aufgabenstellung wurden zusätzlich Angaben über die Threadzahl und die Größe der verwendeten Dateien gemacht, da diese Aspekte eine starke Beeinflussung der Performance erwarten ließen.

Die Aufgabenstellung für ein Experiment muss so entworfen werden, dass sie keines der unter-

suchten Verfahren bevorteilt oder die Ergebnisse aufgrund der Eigenschaften eines Verfahrens verzerrt. Ist dieses Kriterium erfüllt, spricht man von *lokaler Verallgemeinerbarkeit* der Aufgabenstellung [Pre01]. In dieser Studie sollte keines der drei Verfahren durch die Aufgabenstellung benachteiligt sein, da die Eingabeparameter jeweils direkt der Methode angepasst wurden.

Die *globale Verallgemeinerbarkeit* ist dann gegeben, wenn eine repräsentative Aufgabenstellung gefunden wurde, deren relevante Eigenschaften typisch für viele andere Anwendungsfälle sind. Dazu findet sich weiteres in Kapitel 5.7.

5.5.3. Entwurfsalternativen

Mehrere Performance-optimierende Entwurfsvorschläge für die Architektur des Webservers sollten im Experiment vorgegeben werden, so dass die Versuchsteilnehmer lediglich ihre Modelle an diese Alternativen anpassen mussten, um die Performance-Analysen vorzunehmen. Die Entwurfsalternativen sollten möglichst gleichwertig erscheinen, damit nicht sofort intuitiv einer einzelnen Alternative der Vorzug gegeben wurde und die Anwendung der Verfahren besser motiviert war. Damit die Teilnehmer auch tatsächlich nur eine der Entwurfsalternativen favorisierten, wurde in der Aufgabenstellung vermerkt, dass aufgrund von Zeitmangel nur eine von ihnen implementiert werden könne. Es wurden folgende Entwurfsalternativen in die Aufgabenstellung aufgenommen:

- **1a) Statischer Cache:** ein solcher Cache speichert oft angefragte statische HTML-Seiten im Hauptspeicher des Webservers zwischen und reduziert damit die Anzahl zeitlich teurer Festplattenzugriffe. Bei dieser Entwurfsalternative ist ohne Performance-Analyse nicht klar, ob die eingesparten Festplattenzugriffe tatsächlich einen signifikanten Anteil an der Antwortzeit des Servers ausmachen und sich somit die Implementierung dieser Alternative lohnt. Für die Implementierung bot sich eine weitere Komponente mit dem `IDeliverResponse`-Interface an, die vom `RequestProcessor` angesprochen wird und als erste Komponente in die "Chain-of-Responsibility" eingehängt wird. Auf diese Weise könnte bei statischen Anfragen immer zuerst überprüft werden, ob sich die angefragte Datei bereits im Hauptspeicher-Cache befindet. Die Größe der Webseiten ist im Vergleich zum Hauptspeicher relativ gering, denn diese sind meist sehr klein, weil sie für langsame Internetverbindungen optimiert sind. Deshalb wurde für die Aufgabenstellung des Experiments zusätzlich festgelegt, dass 70% aller Anfragen für statische HTML-Seiten direkt aus dem Cache beantwortet werden können.
- **1b) Dynamischer Cache:** diese Variante des Caches ist speziell für die Speicherung dynamischer Seitenaufrufe zuständig. Solche dynamisch zusammengestellten Seiten können natürlich nur dann gecached werden, wenn sie sich nicht bei jeder Anfrage ändern. Dies ist stark applikationsabhängig, z.B. könnten sich nicht ständig ändernde Artikelbeschreibungen in einem E-Commerce System oder täglich erzeugte Wetterdaten eines Wetterdienstes in diese Kategorie fallen. Da die Wahrscheinlichkeit für einen Cache-Hit bei dieser Variante wesentlich niedriger ist als beim statischen Cache, wurde sie auf 20% festgelegt. Trotzdem bietet diese Entwurfsalternative möglicherweise großes Einsparungspotenzial, da die im Vergleich zu lokalen Festplattenzugriffen noch wesentlich teureren entfernten Datenbankabfragen reduziert werden. Die Implementierung konnte als weitere Komponente zwischen

den Komponenten "ExtAppInterface" und "ExtApp" erfolgen.

- **2) Singlethreaded Server:** der eigentlich als multi-threaded implementierte Webserver sollte auch in einer single-threaded Variante getestet werden, d.h. nicht für jede Anfrage sollte eine neuen Programmthread abgespalten werden, sondern die Anfragen sollten sequentiell hintereinander bearbeitet werden. Dies ließ Auswirkungen auf die Antwortzeiten des Servers erwarten, da die Zeit für den Start von Threads und den Kontextwechsel eingespargt werden könnte. Jedoch wäre möglicherweise eine längere Warteschlange vor dem Server zu erwarten und das Reaktionsverhalten des Server könnte sich für den Benutzer subjektiv verschlechtern. Die Implementierung konnte recht einfach erfolgen, da lediglich in der Komponente DispatcherThread das Starten von neuen Threads deaktiviert werden musste.
- **3) Komprimierung:** die Möglichkeit die Zeit für das Versenden angefragter Seiten auf langsamen Internetverbindungen zu reduzieren, indem die Seiten vor dem Versand komprimiert werden, scheint großes Optimierungspotenzial zu bieten. Dabei ist die Implementierung einer solchen Variante recht einfach vorzunehmen, da für den Kompressionsalgorithmus vorgefertigte Bibliotheken unter .NET verwendet werden können und alle modernen Internet-Browser den Empfang von komprimierten Dateien unterstützen und dies für den Benutzer völlig transparent halten. Für die Performance-Analyseverfahren ist diese Entwurfsalternative interessant, da abgewogen werden muss, ob die Zeit, die für die Komprimierung der Dateien von der CPU aufgewendet wird, die reduzierte Zeit für das kürzere Versenden übersteigt. Den Teilnehmern wurde bei dieser Alternative überlassen, eine Kompressionsrate für die Dateien abzuschätzen, denn diese beeinflusst direkt die Zeit für das Versenden der Dateien.
- **4) Clustering:** bei einem Cluster handelt es sich um einen Verbund mehrerer vernetzter Rechner, die die parallele Abarbeitung von Anfragen an ein System ermöglichen. Der Webserver ließ sich auf zwei verschiedenen Computern starten, ein vorgeschalteter Scheduler verteilte die ankommenden Anfragen gleichmäßig. Im Idealfall sollte sich durch diese Replikation der Ressourcen die Performance des Systems verdoppeln, zusätzlich erhöht sich die Ausfallsicherheit. Fraglich ist in diesem Fall, ob die Antwortzeiten für die in diesen Szenario wenigen Benutzer bei den recht kleinen abgefragten Seiten durch ein Clustering tatsächlich reduziert werden können. Eine gewisse Zeit muss zusätzlich für den Scheduler zur Verteilung der Anfragen eingeplant werden. Die Implementierung des Schedulers konnte durch eine dem Webserver vorgeschaltete Komponente realisiert werden.

5.5.4. Durchführung

Das Experiment wurde am 06.07.2004 in einem Rechnerraum durchgeführt, wo acht Windows-Rechner bereitgestellt wurden. Auf den Rechnern waren SPE-ED, Microsoft Excel und Poseidon installiert und zusätzlich die Folien der Übungen (Anhang B) hinterlegt. Zum Experiment erschienen alle Teilnehmer der Übungen, die auch alle am Ende eine Lösung der Aufgabenstellung vorlegten. Während des Experiments wurden kleinere Zwischenfragen einzelner Teilnehmer z.B. zur Aufgabenstellung beantwortet, dabei wurden jedoch die Performance-Modellierungen der

Teilnehmer nicht beeinflusst. Das Experiment dauerte für jede Gruppe jeweils zwei Stunden, wobei die meisten Teilnehmer in dieser Zeit alle Entwurfsalternativen modellieren konnten. Einige Teilnehmer (insbesondere beim Capacity Planning) schafften es nicht, innerhalb dieser Zeit die erforderlichen Berechnungen zu Ende zu führen, gaben aber trotzdem eine Empfehlung für eine Entwurfsalternative ab.

Nach Abschluss des Experiments wurden die von den Teilnehmern ausgefüllten Aufgabenzettel wieder eingesammelt und die von ihnen bearbeiteten Dateien gesichert. Die Performance-Ergebnisse für Antwortzeiten und die Auslastung von Ressourcen wurden für jedes Verfahren in Excel-Tabellen aufgenommen (siehe Anhang G). Bei der ersten Konsistenzprüfung der Daten wurde festgestellt, dass einige der Teilnehmer bei der Bearbeitung der Aufgabenstellung eine Reihe grober Fehler gemacht hatten, die die Ergebnisse verfälschten. Bei der Fehleranalyse muss zwischen drei Fehlerklassen unterschieden werden:

- **Fehler bei der Umsetzung der Aufgabenstellung:** Bei den Ergebnissen war z.T. deutlich zu sehen, dass die Teilnehmer bestimmte Vorgaben in der Aufgabenstellung nicht beachtet hatten und dadurch ihre Lösung entwertet wurde. Dies könnte u.a. auf die Stresssituation, die durch die ungewohnte Arbeitsumgebung und die begrenzte Zeit entstanden war, zurückgeführt werden.
- **Fehler bei der Anwendung der Methoden:** Solche Fehler beinhalten z.B. dass bestimmte Elemente oder Techniken eines Verfahren nicht korrekt angewandt wurden oder die eingesetzten Tools falsch verwendet wurden. Diese Fehler könnten einerseits auf Mängel in den Übungssitzungen bzw. Übungsmaterialien hindeuten, z.B. dass die entsprechenden Elemente fehlerhaft oder undeutlich dargestellt wurden. Andererseits könnten solche Fehler tatsächliche Schwächen der jeweiligen Verfahren aufdecken.
- **Fehler bei den Schätzungen:** Die Eingabeparameter des SPE- und des umlPSI-Verfahrens beruhen zum großen Teil auf Schätzungen von Performance-Daten, die von den Teilnehmern aufgrund ihrer Erfahrung als Entwickler gemacht werden sollten. Auch hier zeigte sich, dass bei den Schätzungen teilweise grobe Fehler auftraten. Zum Beispiel wurde bei einigen Teilnehmern eine Abfrage von Daten über das Internet schneller geschätzt als der Zugriff auf eine lokale Festplatte, wobei eigentlich klar sein sollte, dass dies nicht der Fall ist. Diese Klasse von Fehlern lässt sich entweder auf die mangelnde Erfahrung der Teilnehmer oder auf die Stresssituation des Experiments zurückführen. Oft scheint es so, dass die Teilnehmer ihre Lösung im Nachhinein nicht kontrolliert haben.

Da die Basis für die Evaluation der Performance-Analyseverfahren mit den im Experiment vollständig korrekt erstellten Lösungen gering war, wurde den Teilnehmern die Möglichkeit gegeben, zumindest die Fehler der ersten Klasse (Fehler bei der Umsetzung der Aufgabenstellung) im Nachhinein zu korrigieren. Dazu wurden die fehlerhaften bzw. unfertigen Lösungen an die betreffenden Teilnehmer geschickt, damit sie sie nachträglich überarbeiten konnten. Vorher wurden die Testpersonen auf grobe Fehler in ihrer Lösung bei der Umsetzung der Aufgabenstellung hingewiesen. Fehler, die bei der Anwendung der Methoden entstanden oder die auf stark fehlerhaften Schätzungen beruhen, wurden nicht angemahnt und können weiterhin in den Ergebnissen enthalten sein.

Die Möglichkeit zur Überarbeitung wurde von den meisten angeschriebenen Teilnehmern angenommen, so dass sich die Resultate der Performance-Analysen besserten und die größten Fehler aus den Lösungen entfernt werden konnten. Insgesamt konnten 22 Lösungen als gültig in die Analyse der Ergebnisse übernommen werden. Die Überarbeitung erfolgte nicht mehr in einem kontrollierten Umfeld wie im Experiment, sondern bei den Teilnehmern zu Hause. Dadurch ist nicht sichergestellt, dass die Teilnehmer sich untereinander bei der Erstellung der Ergebnisse nicht abgesprochen haben. Jedoch konnten keine offensichtlichen Übereinstimmungen in den korrigierten Lösungen gefunden werden.

Problematisch gestaltete sich die nachträgliche Simulation der Performance-Modellierungen der dritten Gruppe mit dem umlPSI-Werkzeug. Auch wenn diesmal anders als in der Übung aufgrund der Vorgaben und der Erfahrung deutlich weniger Syntaxfehler in den Tagged-Values der UML-Diagramme zu finden waren, so gelang es trotzdem nur eine einzige Lösung auf Anhieb zu simulieren. Aufgrund der unklaren Meldungen des prototypischen umlPSI-Werkzeugs und der teilweise sehr hohen Simulationsdauer über mehrere Stunden für einzelne Lösungen gestaltete sich die Fehlersuche schwierig. Teilweise wurde die Simulation nach etlichen Stunden abgebrochen, da keine Ergebnisse mehr zu erwarten waren. Grundsätzliches Problem der meisten Lösungen war, dass die geschätzten Zeiten für die einzelnen Aktivitäten der Anwendungsfälle in der Addition zum Teil deutlich über einer Sekunde lagen. Die Benutzerankunftsrate des Systems war jedoch auf genau eine Sekunde festgelegt worden, d.h. im Sekundentakt gelangten neue Benutzer in das System. Daher wurden Ressourcen mit einer Bearbeitungsdauer von mehr als einer Sekunde vollkommen überlastet, es bildeten sich immer größere Warteschlangen und die Simulation war nicht mehr zu lösen. Zur Lösung dieses Problems wurde die Benutzerankunftsrate für jede der Modellierungen individuell niedriger eingestellt. Auf diese Weise war es möglich alle der abgegebenen Lösungen zu simulieren, auch wenn sich die Ergebnisse durch diese Änderung prinzipiell nicht mehr direkt mit den Ergebnissen der anderen Verfahren vergleichen lassen, wo die Benutzerankunftsrate weiterhin konstant bei einem Benutzer pro Sekunde lag.

Die Rohdaten des Experiments finden sich in Anhang [G](#), die Ergebnisse und Auswertungen folgen im nächsten Kapitel.

5.6. Performance-Analyse der Implementierung

Die im Experiment untersuchten Entwurfsalternativen wurden anschließend für den Webserver implementiert, so dass ihre Performance tatsächlich gemessen und mit den Vorhersagen der Performance-Analyseverfahren verglichen werden konnte. Dazu folgen nun einige Erläuterung zum Messaufbau.

Als Testrechner diente ein Notebook mit einem 1,5 GHz Intel Pentium-M Prozessor und 512 MB DDR RAM, auf dem die unterschiedlichen Varianten des Webserver und eine Microsoft SQL Datenbank installiert wurden. Letztere diente zur Simulation der externen Applikation aus dem Experiment.

Für die Simulation der Clients wurden zunächst mehrere Werkzeuge zur Generierung von HTTP-Anfragen getestet. Die Ergebnisse für die Antwortzeiten während dieser Tests waren bei den verschiedenen Werkzeugen weitgehend identisch. Sie stimmten auch mit den Angaben der internen Monitor-Komponente des Webserver überein. Daher genügte es, die vollständige

Untersuchung der Entwurfsalternativen mit nur einem dieser Werkzeuge durchzuführen, welches den besten Bedienkomfort hatte.

Ausgewählt wurde schließlich der "Web Performance Trainer" der Firma Web Performance Inc. [Inc04]. Dieses Werkzeug wurde so konfiguriert, dass die Anfragen an den Webserver von den simulierten Clients mit einer Geschwindigkeit von 8 KByte/s versendet werden konnten. Dieser Wert ging aus der Aufgabenstellung des Experiments hervor, wo die Clients einfache ISDN-Verbindungen (64 KBit/s) verwenden sollten.

Die Aufrufsequenz des Werkzeugs bestand aus einer 5 KByte großen HTML-Datei (Text), einer 5 KByte großen JPG-Datei (Bild) und einer dynamisch generierten HTML-Seite, ebenfalls mit einer Größe von 5 KByte. Für die dynamische Seite wurde eine Suchanfrage an die angeschlossenen Datenbank gestellt und eine kurze Verzögerung eingebaut, damit die Abfrage exakt 0,5 Sekunden dauerte, wie es in der Aufgabenstellung des Experiments angegeben war. Die dynamische Seite wurde drei Mal hintereinander abgefragt, um das Verhältnis von 40:60 zwischen statischen und dynamischen Anfragen aus der Aufgabenstellung des Experiments nachzubilden. Die gesamte Aufrufsequenz der fünf Anfragen wurde über einem Zeitraum von 5 Minuten ständig wiederholt, um eine große Anzahl an Messwerten zu erhalten und mögliche Abweichungen aufgrund von äußeren Einflüssen durch Mittelung auszuschließen.

Mit dem Werkzeug war es nicht möglich, die Ankunftsrate von einer Anfrage pro Sekunde wie in der Aufgabenstellung exakt einzustellen. Einfluss genommen werden konnte auf die Ankunftsrate einzig durch die Angabe der Anzahl der Benutzer, die gleichzeitig auf den Webserver zugriffen. Hier wurde der Wert von 1,5 gleichzeitigen Benutzern gewählt, da die Anzahl der Anfragen dabei innerhalb von 5 Minuten am nächsten bei 300 lag (1 Anfrage pro Sekunde). Die exakten Ankunftsdaten sind im Ergebnisdiagramm enthalten und wurden über die Anzahl der in 5 Minuten von jeder Entwurfsalternative bearbeiteten Anfragen bestimmt. Auch wenn die Werte nicht genau mit den Angaben der Aufgabenstellung übereinstimmen, so zeigten Messungen mit wesentlich höheren Ankunftsdaten (10 gleichzeitige Benutzer im System), dass für die Durchlaufzeiten keine größeren Abweichungen zu erwarten waren. Daher sollten die Messungen trotzdem mit den Vorhersagen der Verfahren vergleichbar sein.

Für Alternative 1a wurde der Cache für statische Seiten so konfiguriert, dass genau 70 Prozent der Anfragen aus dem Cache beantwortet wurden, während die restlichen 30 Prozent weiterhin von der Festplatte bezogen wurden. Genauso wurde beim dynamischen Caching vorgegangen: hier wurden 20 Prozent der Anfragen für dynamische Seiten aus dem Hauptspeicher bezogen, 80 Prozent der Anfragen wurden weiterhin an die Datenbank gestellt. Damit wurden die Vorgaben der Aufgabenstellung des Experiments genau nachgebildet.

Die zusätzliche Abfrage einer JPG-Datei wurde vorgenommen, weil bei Entwurfsalternative 3 (Komprimierung) der Dateityp Einfluss auf die Durchlaufzeit hat. HTML-Dateien bestehen ausschließlich aus ASCII-Text, der gut komprimiert werden kann (im Mittel um etwa 50%, maximal sogar bis zu 90% [Kil02, p.348]). JPG-Dateien dagegen liegen bereits in einem komprimierten Format vor und können meist nicht weiter reduziert werden. Daher ist eine differenzierte Abfrage zwischen stark und schwach komprimierbaren Dateien sinnvoll, um die Ergebnisse für die Antwortzeiten zu verfeinern und die tatsächliche Auswirkung von Entwurfsalternative 3 zu untersuchen.

Für die Alternative 4 (Clustering) wurden zwei Webserver und der Scheduler gestartet. Dieser verteilte die eingehenden Anfragen gleichmäßig auf die beiden Server.

Parallel zur Messung der Durchlaufzeiten durch den "Web Performance Trainer" wurde zusätzlich die prozentuale Auslastung von CPU und Festplatte während der Simulation mit dem Windows XP Performance Monitor aufgezeichnet. Diese Werte können ebenfalls mit den Vorhersagen für die Auslastungen verglichen werden.

Bei den Messungen wurden alle nicht benötigten Prozesse auf dem Rechner beendet, um die Messungen nicht unkontrollierbar zu beeinflussen. Trotzdem ist nicht auszuschließen, dass betriebssystem-interne Aktionen die Messungen verzerrt haben, während der Tests waren jedoch keine signifikanten Auffälligkeiten zu verzeichnen. Aus den gemessenen Durchlaufzeiten über 5 Minuten wurde jeweils der Mittelwert gebildet. Die Rohdaten aller Messungen können in Anhang I eingesehen werden, Diagramme mit den Mittelwerten folgen in Abschnitt 6.1.4.

5.7. Validität

Um die Qualität der Ergebnisse der Fallstudie sicherzustellen, sollen nun einige Überlegungen angestellt werden. Dazu werden die Konstruktgültigkeit, die innere Gültigkeit, die äußere Gültigkeit und die Zuverlässigkeit der Fallstudie betrachtet [Yin94].

5.7.1. Konstruktgültigkeit

Die Konstruktgültigkeit überprüft, ob der Aufbau der Fallstudie tatsächlich dazu geeignet ist die gewünschten Fragestellungen zu beantworten, d.h. inwiefern die gemessenen Daten tatsächlich das messen, was sie messen sollen und nicht subjektiv durch den Experimentator aufgestellt wurden.

- **Performance-Daten:** Darunter werden die verschiedenen Ausgabeparameter der Performance-Analyseverfahren verstanden (Verweilzeiten, Auslastungen, Warteschlangenlängen, Durchsätze usw.) Diese Daten werden entweder automatisch berechnet (bei SPE und CP) oder durch eine Simulation (bei umlPSI) bestimmt. Daher sind diese Messungen objektiv und nicht vom Forscher abhängig. Yin [Yin94] nennt zur Sicherstellung der Konstruktgültigkeit u.a. die Verwendung von mehreren Beweisquellen. Dies wurde für die Performance-Daten insofern durchgeführt, als dass die Daten der verschiedenen Verfahren für die gleiche Architektur verglichen werden können, zusätzlich können Messungen an der prototypischen Implementierung durchgeführt werden. Somit sind mehrere Quellen für die Performance-Daten verfügbar.
- **Fehler in der Performance-Modellierung:** Offensichtliche Fehler in den Ergebnissen wurden durch die oben beschriebene nachträgliche Korrektur der Lösungen reduziert.

5.7.2. Innere Gültigkeit

Die innere Gültigkeit beschreibt, wie gut die Kontrolle der Störvariablen in einem Experiment gelungen ist [Pre01]. Störvariablen können die Ergebnisse eines Experiments verfälschen und so möglicherweise falsche Schlüsse aus einem Experiment ziehen lassen. Eine völlige Kontrolle der Störvariablen kann im Rahmen einer Fallstudie nicht erreicht werden, trotzdem sollen mögliche Bedrohungen der inneren Gültigkeit minimiert werden. Dabei lassen sich folgende Kategorien von Beeinträchtigungen unterscheiden:

- **Reifung:** Das Verhalten der Teilnehmer kann sich während des Experiments verändern. Zum einen lernt jeder Teilnehmer im Laufe des Experiments hinzu und kann das gelernte Wissen sofort anwenden, wenn sich bestimmte Teilaufgaben wiederholen. Zum anderen ist bei längeren Aufgaben ein gewisser Ermüdungseffekt zu beobachten, so dass sich ein Teilnehmer bei späteren Teilaufgaben des Experiments nicht mehr richtig konzentrieren kann. Ein Reihenfolgeeffekt kann in diesem Experiment insofern ausgemacht werden, als dass sich der Vorgang der Performance-Modellierung für die fünf Entwurfsalternativen jeweils wiederholt. Dabei bestand zwar nicht die Möglichkeit eines Lerneffekts, jedoch die Gefahr, dass Erwartungswerte für eine Alternative ohne Überprüfung bei der Modellierung der nächsten Alternative übernommen wurden. Dies trat auch tatsächlich bei einigen Teilnehmern auf, die z.B. die Beschleunigung durch die Kompression in Alternative 3 mit in die Modellierung der Alternative 4 übernahmen, obwohl von der Aufgabenstellung die unabhängige Modellierung der Alternativen gefordert war. Diese Fehler wurden jedoch bei der Überarbeitung der Lösungen angemahnt und aus den Lösungen entfernt. Ermüdungseffekte konnten bei einer Dauer von zwei Stunden noch nicht verzeichnet werden, allerdings schafften es einige Teilnehmer nicht alle Alternativen zu modellieren. Auch dieses Defizit wurde durch die Überarbeitung der Lösungen ausgeräumt.
- **Instrumentation** Außer dem Verhalten der Versuchsteilnehmer kann sich auch der Experimentaufbau bzw. das Verhalten des Experimentators bei der Datensammlung und -analyse im Verlauf der Zeit verändern. In diesem Fall spricht man von Instrumentationseffekten. Da das Experiment in dieser Studie nur einmalig durchgeführt wurde, und die Aufgabenstellung schriftlich vorgelegt wurde, ist eine Veränderung des Experimentaufbaus im Verlauf der Zeit nicht gegeben. Die Sammlung der Daten erfolgte werkzeuggestützt, außerdem mussten bei den Ergebnissen keine subjektiven Bewertungen durch den Experimentator vorgenommen werden, da die Performance-Ergebnisse durch Zahlen ausgedrückt wurden.
- **Historie:** Dauert eine Studie mehrere Wochen, so können äußere Einflüsse, wie z.B. Nachrichten in der Fachpresse Auswirkungen auf die Studie haben und die Motivation der Teilnehmer verändern. Dieser Punkt kann für diese Studie ausgeschlossen werden, da keine relevanten äußeren Einflüsse auf die Studie beobachtet werden konnte. Auch die Absprache mit früheren Versuchsteilnehmern war nicht möglich, da die Studie zum ersten Mal durchgeführt wurde.
- **Auswahl:** Dieser Punkt betrifft die Einteilung der Versuchsteilnehmer in Gruppen. Ist die Gruppeneinteilung ungünstig und gehören einer der Gruppen wesentlich leistungsfähigere Personen an, wird das Ergebnis eines Experiments verzerrt. Um für diese Studie eine balancierte Gruppeneinteilung vorzunehmen, wurden, wie weiter oben bereits erläutert, die Ergebnisse der Übungen hinzugezogen. Auf diese Weise sollten Auswahl Effekte bei der Gruppeneinteilung so gut wie möglich kontrolliert worden sein.
- **Regression:** Erbringt ein Teilnehmer eine für seine Verhältnisse besonders gute oder schlechte Leistung, führt dies zu einer unnatürlichen Abweichung vom Mittelwert und man spricht von Regressionseffekten. Für diesen Punkt kann nicht ausgeschlossen werden,

dass die Teilnehmer in einer der Übungen wesentlich von ihrer Normalleistung abgewichen sind und die Gruppeneinteilung sich dadurch verzerrt hat. Jedoch waren die Ergebnisse bei den meisten Teilnehmern über alle drei Übungen recht konstant, so dass keine wirklich offensichtlichen Regressionseffekte zu beobachten sind.

- **Sterblichkeit:** Durch das Ausscheiden von Teilnehmern während des Experiments können zum einen die anderen Teilnehmer in ihrer Motivation beeinflusst werden, zum anderen verzerrt sich dadurch die Gruppeneinteilung. Glücklicherweise war bei dem hier durchgeführten Experiment keine Mortalität zu verzeichnen, alle Teilnehmer der Übungen erschienen auch zum Experiment.
- **Anforderungscharakteristik:** Gibt es Unterschiede in den Aufgabenstellungen für die einzelnen Gruppen, z.B. dadurch dass der Experimentator eine der Methoden bevorzugt, sind die Ergebnisse des Experiments nicht mehr aussagekräftig. Bei der Entwicklung der Aufgabenstellung wurde daher darauf geachtet, die einzelnen Dokumente möglichst ähnlich zu halten und die Neutralität gegenüber jedem Verfahren zu wahren. Auch z.B. die enttäuschenden Ergebnisse mit dem umlPSI-Verfahren während der Übung führten nicht dazu, dass die Aufgabenstellung für dieses Verfahren von denen der anderen Verfahren qualitativ oder quantitativ abwich (siehe Aufgabenstellung im Anhang).
- **Verarbeitungsfehler:** Eine weitere Bedrohung für die innere Gültigkeit eines Versuchs ist die fehlerhafte Datenmessung bzw. -weiterverarbeitung. Die Ergebnisse der einzelnen Verfahren wurden werkzeuggestützt erlangt, daher sollten sich kaum Berechnungsfehler ergeben haben. Natürlich kann nicht ausgeschlossen werden, dass bei der Fülle von Zahlen die Ergebnisse fehlerhaft von den Versuchsteilnehmer aufgeschrieben wurden oder dass bei der Datenerfassung Tippfehler gemacht wurden.

5.7.3. Äußere Gültigkeit

Mit der äußeren Gültigkeit bezeichnet man den Grad, zu dem sich die Resultate der Fallstudie verallgemeinern lassen und auf andere Anwendungsfälle übertragbar sind [Pre01]. Dies ist klassischerweise der wunde Punkt von Fallstudien, die aufgrund des wenig kontrollierbaren Umfelds und der Untersuchung von Einzelfällen eine schwache Basis für Verallgemeinerungen darstellen [Yin94]. Einige Punkte sollen nun einer kritischen Bewertung unterzogen werden:

- **Repräsentative Auswahl der Teilnehmer:** Wie bereits ausgeführt waren die Teilnehmer der Studie Informatikstudenten mit abgeschlossenem Vordiplom, die also Grundlagen der Informatik bereits beherrschten. Trotzdem wäre es wünschenswert gewesen, auch Versuchsteilnehmer mit einem anderen Hintergrund, z.B. Profis aus der Industrie mit möglicherweise mehrjähriger Berufserfahrung, in der Studie zu haben. Dies war jedoch aus organisatorischen Gründen nicht möglich.
- **Größe der Stichprobe:** Die Anzahl der Teilnehmer ist mit acht Personen pro Verfahren in der Tat gering und für eine statistische Verallgemeinerung der Ergebnisse nicht ausreichend. Idealerweise müsste die Studie mit weiteren Personen wiederholt werden.

- **Verallgemeinerbarkeit des Fallbeispiels:** Der in dieser Fallstudie untersuchte Webserver kann als stellvertretend für andere Applikationen angesehen werden, bei denen die Performance-Eigenschaften schon während des Entwurfs optimiert werden müssen. Da bei der Analyse nur Performancerelevante Elemente des Servers betrachtet wurden, ist keine allzu enge Festlegung auf einen Applikationskontext gegeben. Die Verallgemeinerbarkeit der Studie auf die große Klasse der Web-Applikationen sollte auf jeden Fall möglich sein.
- **Subjektive Darstellung der Verfahren:** Da sich die Versuchsteilnehmer die Kenntnisse über die Verfahren nicht direkt aus der Literatur der Autoren aneigneten, ist natürlich der Faktor zu berücksichtigen, dass der Experimentator durch seine inhärent subjektive Vorstellung der Verfahren deren objektive Bewertung verzerrt hat. Insbesondere ist es aufgrund des Umfangs der Verfahren möglich, dass bestimmte Vereinfachungen gemacht wurden und Teile der Verfahren ausgeklammert wurden. Hier bleibt festzuhalten, dass zumindest die wichtigsten Elemente der Verfahren untersucht wurden.

5.7.4. Replizierbarkeit

Um die Replizierbarkeit der Fallstudie zu gewährleisten, es also anderen Forschern zu ermöglichen die Studie unter den gleichen Bedingungen erneut durchzuführen, wurden sämtliche Arbeitsunterlagen, Aufgabenstellungen und relevanten Dokumente in den Anhang dieser Ausarbeitung integriert. Zusätzlich sind die Rohdaten der Ergebnisse der Performance-Analysen dort zu finden, damit eine erneute, unabhängige Bewertung der Ergebnisse durchgeführt werden könnte. Somit können Zweifel an der Auswertung der Daten ausgeräumt werden, indem Dritte die Rohdaten erneut auswerten.

6. Ergebnisse

In Kapitel 3 wurden vier Forschungsfragen über Performance-Analyseverfahren gestellt und in Kapitel 5 der Aufbau der Fallstudie erläutert, mit der versucht werden soll, diese Fragen zu beantworten. Wie aus den gesammelten Daten Erklärungen und Erkenntnisse über die in Kapitel 4 ausgewählten Verfahren gewonnen werden können, soll im nun folgenden Kapitel erläutert werden.

6.1. Validität der Vorhersagen

Als erste Forschungsfrage wurde in Kapitel 3 formuliert:

1. Wie hoch ist die Genauigkeit der Vorhersagen der Verfahren?

Zur Beantwortung dieser Frage wurden vier Metriken angegeben: die Abweichung der Eingabedaten verschiedener Teilnehmer (Metrik 1), die Abweichung von Eingabe- und Ausgabedaten verschiedener Verfahren (Metrik 2), die Abweichung von Ausgabedaten der Verfahren und Messdaten einer Implementierung (Metrik 3) und der Prozentsatz richtiger Entwurfsentscheidungen (Metrik 4).

Zunächst müssen einige Bemerkungen zu Metrik 2 gemacht werden. Die Abbildung der Eingabedaten auf die Ausgaben der Verfahren wie Antwortzeiten und Ressourcenauslastung erfolgt beim SPE und beim CP durch die Analyse von Warteschlangenmodellen. Dabei werden durch Algorithmen exakte numerische Werte berechnet, eine Streuung der Werte findet nicht statt. Gibt man beiden Verfahren exakt die gleichen Eingabedaten, so ergeben sich bei beiden auch die gleichen Ausgabedaten, die Analyse der Warteschlangenmodelle ist deterministisch und wiederholbar. Deshalb kann von einer Abweichung bei der Abbildung von Eingabe- auf Ausgabedaten bei diesen Verfahren nicht gesprochen werden.

Etwas anders stellt es sich beim simulationsbasierten umlPSI-Verfahren dar. Die Ausgabedaten sind keine exakten Werte, sondern werden in Form von Konfidenzintervallen angegeben. Explizit gibt umlPSI z.B. bei den Antwortzeiten für einen Anwendungsfall die minimale und die maximale Zeit an, die bei der mehrmaligen Ausführung der Simulation dafür verbraucht wurde. Somit ist eine Streuung der Werte gegeben, die von Faktoren wie z.B. der Auslastung des Simulationsrechners und der Dauer der Simulation abhängt. Jedoch kann die relative Breite des Konfidenzintervalls eingestellt und somit die Genauigkeit beeinflusst werden. Die Simulation wird dann so lange fortgesetzt bis die Werte innerhalb des angegebenen Konfidenzintervalls liegen. Zur Simulation der Experimentergebnisse wurde eine relative Breite von 10% für das Konfidenzintervall (Standardeinstellung) eingestellt, die später ausgegebenen Werte wichen aber sogar weit weniger als 10% zwischen Minimal- und Maximalwert ab (siehe Anhang). Somit kann die Abweichung der Eingabe- von den Ausgabedaten bei diesem Verfahren als niedrig eingestuft werden.

Da die Abweichung bei der Abbildung der Eingabedaten auf die Ausgabedaten gering bzw. nicht vorhanden ist, genügt es, zur Analyse der Abweichung der Eingabedaten (Metrik 1) die Ausgabedaten zu betrachten. Als Ergebnis der Performance-Analysen im Experiment wurden bei jedem Verfahren die Durchlaufzeiten und Ressourcenauslastungen jeder Entwurfsalternative erfasst. Unter der Durchlaufzeit wird in diesem Fall die Zeit verstanden, die die Bearbeitung einer Benutzeranfrage vom Betreten des Systems bis zum Verlassen verbraucht. Mit der Ressourcenauslastung ist die prozentuale Auslastung von Hardware-Elementen wie CPU oder Festplatte, also das Verhältnis von Bearbeitungszeit zum Gesamtzeitraum der Beobachtung.

Der Vergleich einer Durchlaufzeit für jeden Teilnehmer und jede Entwurfsalternative ist dabei übersichtlicher und aussagekräftiger als die Eingabedaten für jeden einzelnen Schritt im Kontrollfluss, jeden Teilnehmer und jede Entwurfsalternative zu vergleichen. Dies soll in den nun folgenden Abschnitten 6.1.1 - 6.1.3 für jedes Verfahren geschehen. Die Messdaten der Implementierung werden in Abschnitt 6.1.4 erläutert, bevor dann in Abschnitt 6.1.5 ein Vergleich zwischen den Ausgabedaten der Verfahren und den Messungen erfolgt (Metrik 3). Die ausgewählten Entwurfsentscheidungen (Metrik 4) werden in Abschnitt 6.1.6 untersucht.

6.1.1. Vorhersagen SPE

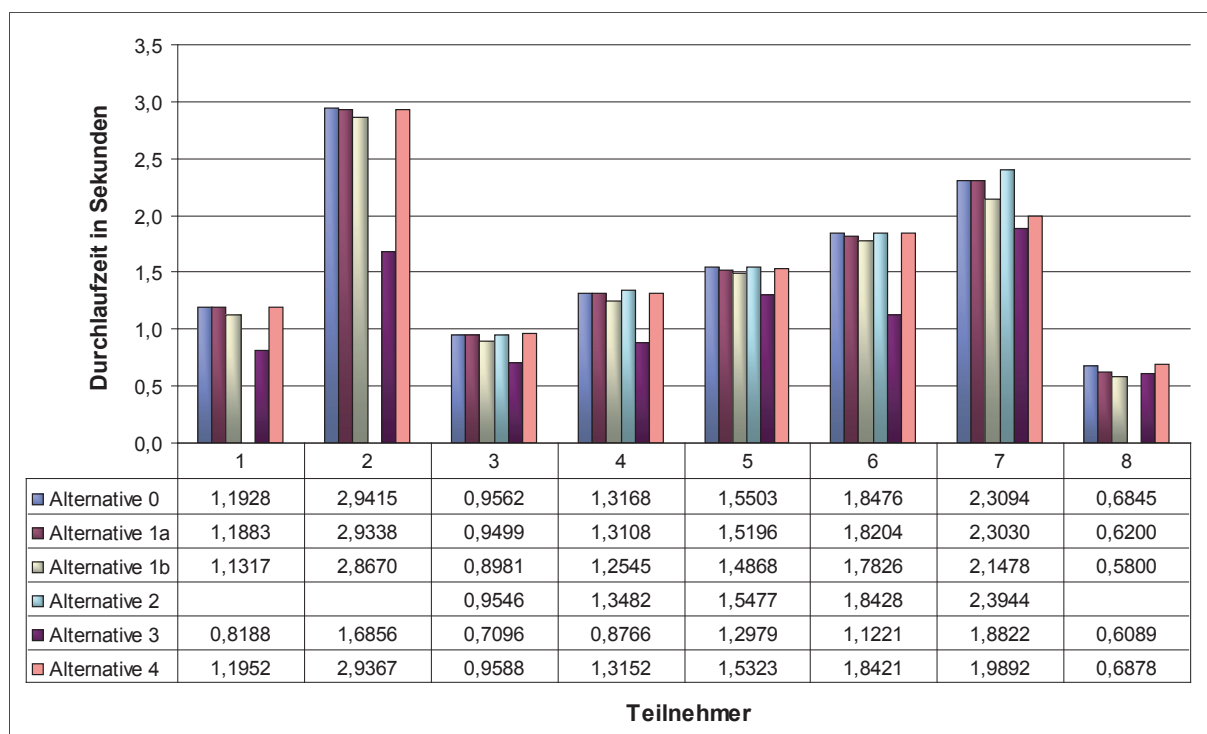


Abbildung 6.1.: Vorhergesagte Durchlaufzeiten beim SPE-Verfahren

Beim SPE-Verfahren wurde eine Durchlaufzeit sowohl für die Abfrage dynamischer als auch statischer HTML-Seiten mit einer Gewichtung von 60% für dynamische und 40% für statische Inhalte angegeben. Die Ankunftsrate der Anfragen betrug entsprechend der Aufgabenstellung eine Anfrage pro Sekunde. Mehreren Teilnehmern gelang es nicht, für Entwurfsalternative 2 (single-

threaded Server) eine Durchlaufzeit zu berechnen. Es war schwierig, die Anzahl der Threads mit dem Verfahren zu modellieren.

Der Mittelwert für die Durchlaufzeiten der nicht-optimierte Variante des Webserver liegt bei 1,5999 Sekunden, die Standardabweichung ist mit 0,7431 Sekunden ähnlich wie in den Übungen recht hoch. Dabei waren die Zeiten für einige der enthaltenen Einzelschritte in der Aufgabenstellung vorgegeben und hätten zu einer engeren Verteilung der Werte führen sollen. Trotzdem sind signifikante Muster bezüglich der Performance-Verbesserungen durch die Entwurfsalternativen zu erkennen. Bei sieben von acht Lösungen hat Alternative 3 (Komprimierung) die geringste Durchlaufzeit, die zweitschnellste Alternative ist bei den sechs Lösungen Alternative 1b (dynamischer Cache). Die anderen Entwurfsalternativen scheinen so gut wie keine Verbesserungen der Durchlaufzeit zu erzielen, bei Alternative 2 (single-threaded Server) ergibt sich im Mittel sogar eine Verschlechterung der Performance.

	CPU	Disk	LAN	Delay	Inet
Alternative 0: keine Optimierung	5%	2%	0%	53%	125%
Alternative 1a: statischer Cache	4%	2%	0%	52%	125%
Alternative 1b: dynamischer Cache	4%	3%	0%	42%	124%
Alternative 2: single-threaded Server	8%	4%	0%	102%	178%
Alternative 3: Komprimierung	5%	3%	0%	48%	78%
Alternative 4: Clustering	3%	1%	0%	48%	125%

Tabelle 6.1.: Vorhergesagte Ressourcenauslastung beim SPE-Verfahren (Mittelwert)

Zur besseren Übersichtlichkeit wurde für die Auslastung der Mittelwert über alle Teilnehmer für jede Ressource und jede Alternative in einer Tabelle angegeben. Die einzelnen Werte der Teilnehmer können im Anhang eingesehen werden. Am höchsten ist die Belastung des Internets und der externen Applikation (Delay). Die Auslastung von CPU, Festplatte und LAN ist dagegen bei allen Lösungen so gering, dass sie vernachlässigt werden kann. Weiterhin können diese Werte verwendet werden, um die Modellierungen der Teilnehmer zu validieren. So sinkt z.B. beim dynamischen Cache die Auslastung der Delay-Ressource um 10% auf 42%. Dies entspricht den Vorgaben der Aufgabenstellung, da die Daten nun durch den Cache zum Teil direkt aus dem Hauptspeicher des Webserver bezogen werden können und die externe Applikation seltener angesprochen wird. Beim single-threaded Server erhöhen sich die Werte von CPU, Delay und Internet signifikant, bei der Komprimierung wird das Internet um 48% weniger belastet, da nun weniger Daten über die Leitung geschickt werden. Die Teilnehmer schätzten die durch die Komprimierung auftretende zusätzliche Belastung der CPU nur als geringfügig ein. Beim Clustering halbieren sich die Auslastungen für CPU und Disk.

Den vorhergesagten Werten entsprechend wurden die Entwurfsentscheidungen gefällt: sechs Teilnehmer gaben Alternative 3 den Vorzug. Ein Teilnehmer entschied sich für Alternative 4 (Clustering) mit der Begründung, dass bei dieser Lösung gleichzeitig die Ausfallsicherheit der Architektur erhöht wird. Ein weiterer Teilnehmer favorisierte aufgrund seiner Ergebnisse Entwurfsalternative 1b (dynamischer Cache) als leistungstärkste Lösung.

6. Ergebnisse

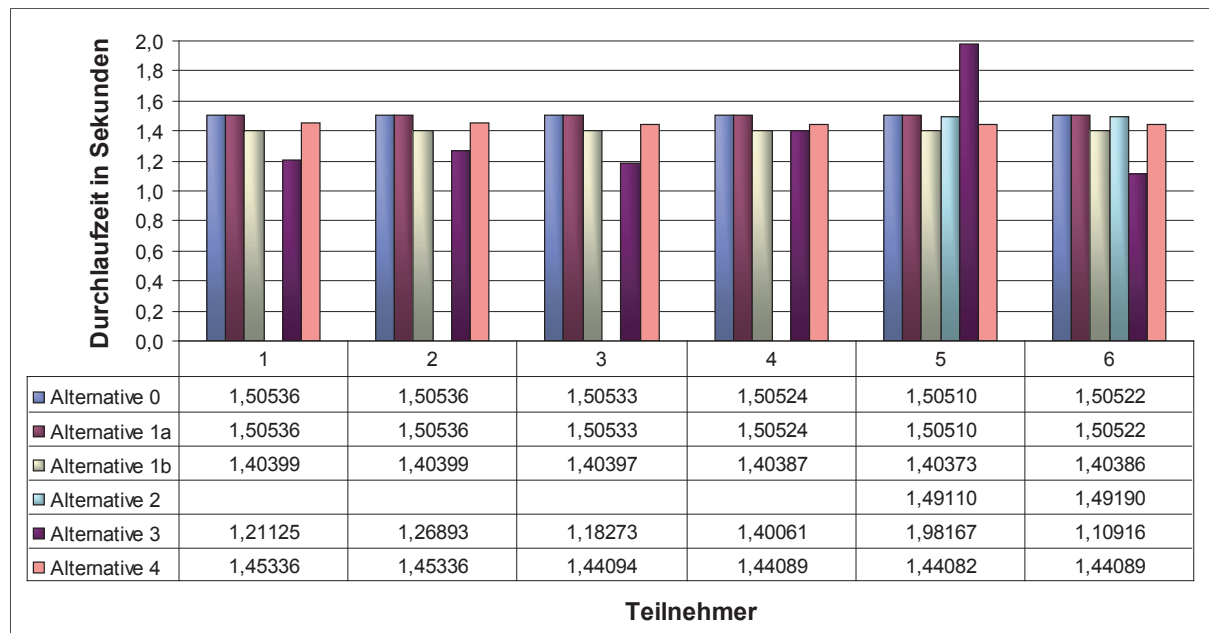


Abbildung 6.2.: Vorhergesagte Durchlaufzeiten beim CP-Verfahren (dynamisch)

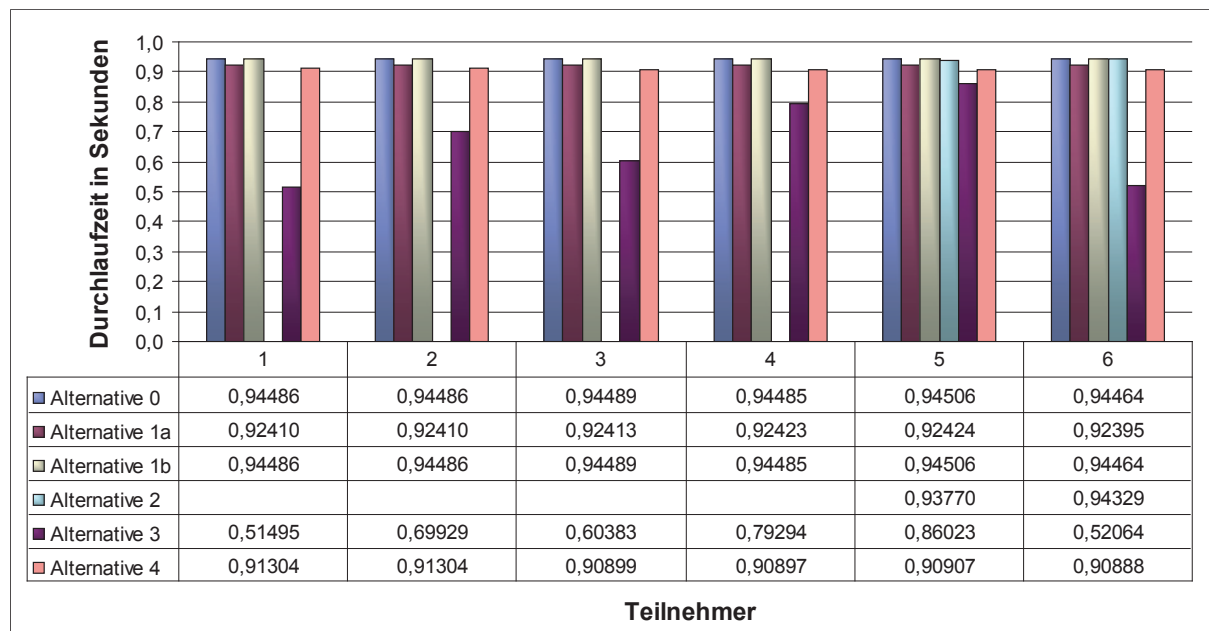


Abbildung 6.3.: Vorhergesagte Durchlaufzeiten beim CP-Verfahren (statisch)

6.1.2. Vorhersagen Capacity Planning

Für das Capacity Planning wurden anders als beim SPE getrennte Ergebnisse für statische und dynamische Seitenaufrufe berechnet. Diese waren als zwei Systemlastklassen eines Warteschlangenmodells modelliert worden. Damit konnte auch gleich die Validität der Ergebnisse für die Caches geprüft werden, denn falls z.B. ein Cache nur für dynamische Seiten modelliert wurde, sollten sich bei statischen Seitenabfragen keine Änderungen der Durchlaufzeiten ergeben. Dieses Kriterium erfüllten nach der in Kapitel 5 beschriebenen Überarbeitung alle der abgegebenen gültigen sechs Lösungen. Das Verhältnis von statischen zu dynamischen Anfragen war wie beim SPE 40:60 und die Benutzerankunftsrate lag bei einem Benutzer pro Sekunde.

Der Mittelwert der Durchlaufzeiten beträgt bei der nicht-optimierten Version des Webserver für die dynamische Anfragen 1,5053 Sekunden und für die statischen Anfragen 0,9449 Sekunden. Die Standardabweichung beträgt bei beiden Anfragentypen nur 0,0001 Sekunden. Die Werte liegen sehr eng zusammen, da sie ausschließlich aus Berechnungen hervorgehen, etwaige Abweichungen sind das Ergebnis von Rundungsfehlern. Bei der Modellierung der Entwurfsalternativen ergibt sich ebenfalls nur eine geringe Streuung der Werte, da die Vorgaben aus der Aufgabenstellung (z.B. 70% der Seiten können vom statischen Cache bearbeitet werden) direkt für die Berechnungen genutzt werden konnten. Bei der Alternative 3 (Komprimierung) musste eine Kompressionsrate für die Dateien abgeschätzt werden, deshalb ergeben sich hier sichtbare Abweichungen der Werte.

Bei fünf der sechs Lösungen war die Durchlaufzeit für Entwurfsalternative 3 am geringsten, bei allen Teilnehmern war Entwurfsalternative 1b die zweitschnellste Variante des Webserver. Nur ein Teilnehmer schätzte die CPU-Zeit für die Komprimierung höher ein als die Zeit, die durch die kürzere Belastung der Internetverbindung eingespart werden konnte. Auch bei diesem Verfahren hatten die meisten (sechs von acht) Teilnehmern für Alternative 2 (single-threaded Server) keine Idee zur Modellierung und lieferten hier keine Ergebnisse.

	CPU	Disk	LAN	Delay	Inet
Alternative 0: keine Optimierung	8,2%	1,2%	0,4%	29,5%	87,5%
Alternative 1a: statischer Cache	8,2%	0,36%	0,4%	29,5%	87,5%
Alternative 1b: dynamischer Cache	8,2%	1,2%	0,3%	23,6%	87,5%
Alternative 2: single-threaded Server	7,92%	1,2%	0,4%	29,5%	87,5%
Alternative 3: Komprimierung	15,63%	1,1%	0,4%	28,8%	48,5%
Alternative 4: Clustering	4,4%	0,6%	0,4%	29,5%	87,5%

Tabelle 6.2.: Vorhergesagte Ressourcenauslastung beim CP-Verfahren (Mittelwert)

Die Auslastungen der Ressourcen CPU, Disk und LAN waren in der Aufgabenstellung notwendigerweise vorgegeben worden, bei den anderen Ressourcen (INet, Delay) konnten diese Werte direkt aus Vorgaben berechnet werden. Daher sind die Werte bei allen Teilnehmern für Alternative 0 (keine Optimierung) gleich. Bei den Entwurfsalternativen lässt sich die korrekte Modellierung ablesen: der statische Cache reduziert die Auslastung der Festplatte in Alternative 1a, LAN und Delay werden durch den dynamischen Cache in Alternative 1b weniger belastet. Durch die Komprimierung halbiert sich die Auslastung des Internets fast, während die CPU für den Kompressionsalgorithmus etwas Rechenzeit aufbringen muss. Bei Alternative 4 reduziert

sich schließlich die Auslastung von CPU und Festplatte durch die Verdoppelung der Ressourcen bei allen Teilnehmern um die Hälfte.

6.1.3. Vorhersagen umlPSI

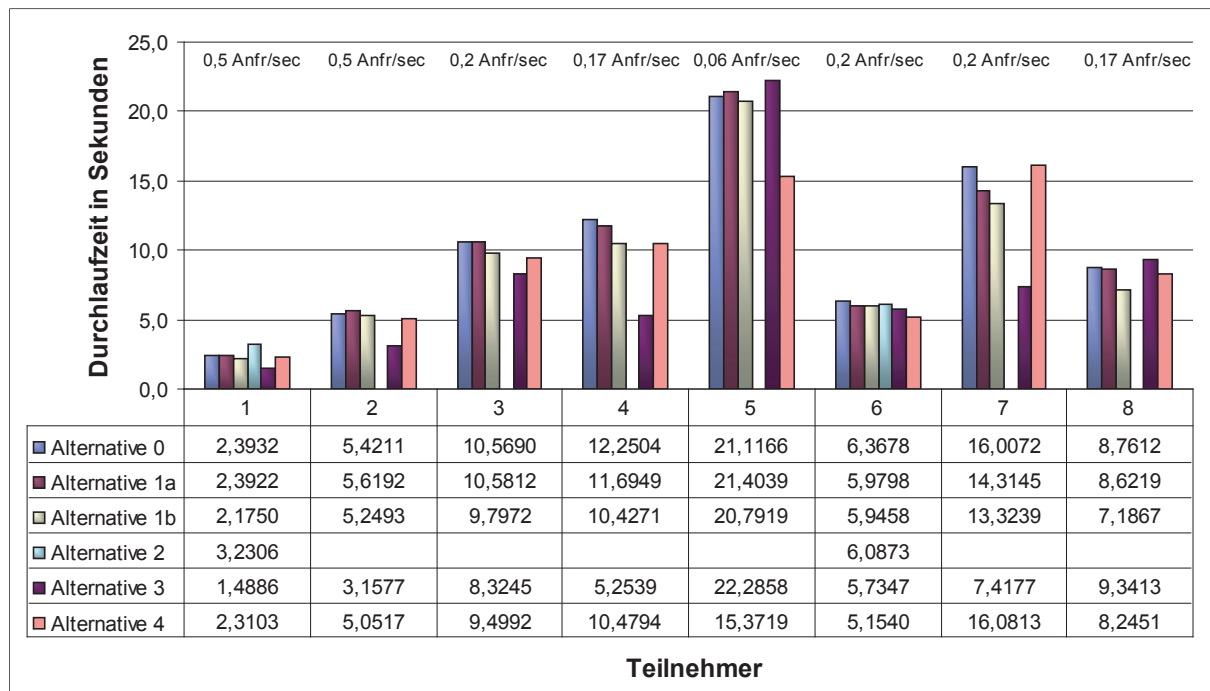


Abbildung 6.4.: Vorhergesagte Durchlaufzeiten beim umlPSI-Verfahren (dynamisch)

Das umlPSI-Verfahren liefert wie das Capacity Planning unterschiedliche Performance-Ergebnisse für statische und dynamische Seitenaufrufe im Verhältnis 40:60. Für die Simulation der Performance-Modelle musste eine andere Ankunftsrate für die eingehenden Anfragen eingestellt werden als bei den anderen Verfahren (dort war es ein Benutzer pro Sekunde). Andernfalls wäre das Simulationsprogramm aufgrund der hohen Schätzungen für die Eingabedaten überlastet gewesen und hätte immer längere Warteschlangen vor den Ressourcen produziert (siehe Abschnitt 5.5.4). Die Ankunftsrate für Anfragen wurde deshalb jeweils individuell für jeden Teilnehmer angepasst, so dass möglichst wenig Überschneidungen durch gleichzeitige Anfragen vorlagen und die Simulation nicht überlastet wurde. Daher sind die Durchlaufzeiten und Auslastungen dieses Verfahrens auch nur bedingt mit denen der anderen Verfahren vergleichbar.

Für die nicht-optimierte Version des Webserver betrug der Mittelwert der Vorhersagen bei dynamischen Anfragen 10,3608 Sekunden und bei statischen Anfragen 7,9624 Sekunden. Diese Werte sind weit höher als bei den anderen Verfahren. Auch die Standardabweichung der Werte ist mit 6,0659 Sekunden (dynamisch) bzw. 5,4411 Sekunden (statisch) wesentlich größer als beim SPE und beim CP.

Alle Teilnehmer modellierten die Alternative 2 (single-threaded Server), was lediglich der Eingabe eines Parameters in der Ressource "Threads" bedurfte. Bei der Simulation der Modelle mit umlPSI zeigte sich jedoch, dass in den meisten Fällen keine Durchlaufzeit für diese

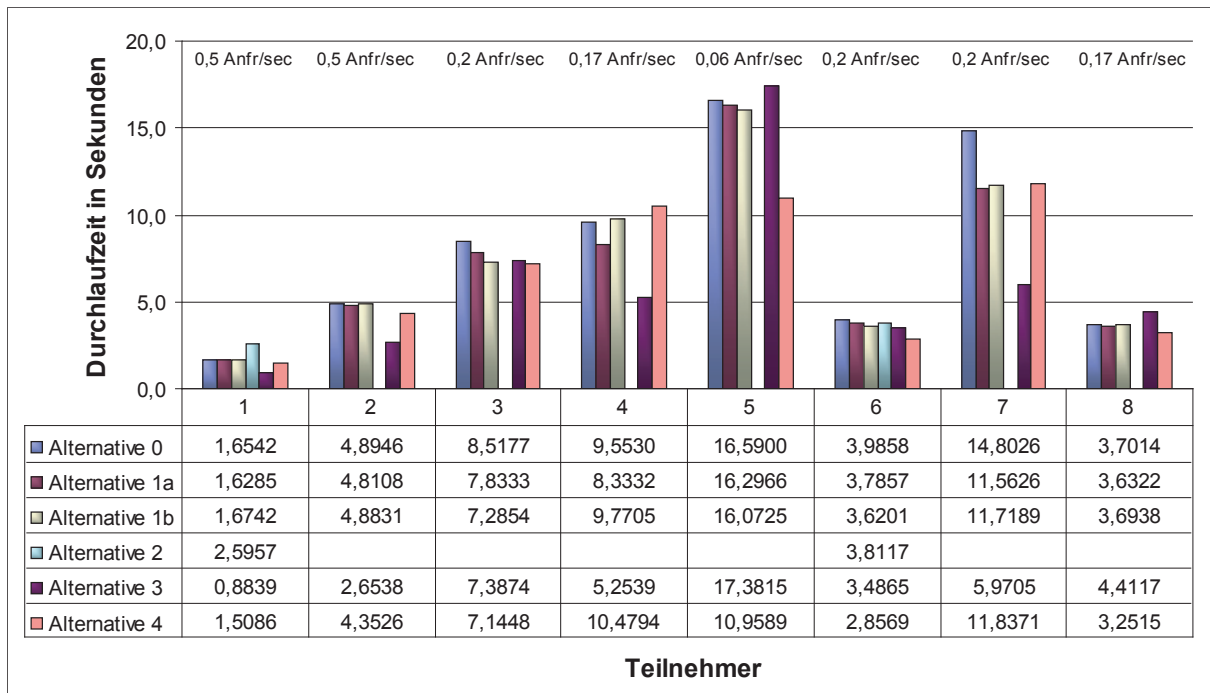


Abbildung 6.5.: Vorhergesagte Durchlaufzeiten beim umlPSI-Verfahren (statisch)

Entwurfsalternative ermittelt werden konnte. Grund dafür war die starke Überlastung der Ressource "Threads", was man in Tabelle 6.3 für die Auslastungen gut erkennen kann, wo diese Ressource bei Alternative 2 mit 100 Prozent belastet ist.

	CPU	Disk	LAN	Delay	Inet	Threads
Alternative 0: keine Optimierung	19,3%	4,0%	10,0%	9,4%	52,2%	25,8%
Alternative 1a: statischer Cache	19,6%	1,4%	10,2%	9,4%	52,0%	25,4%
Alternative 1b: dynamischer Cache	19,0%	4,0%	8,2%	7,5%	51,5%	24,4%
Alternative 2: single-threaded Server	17,8%	3,6%	9,5%	8,9%	49,7%	100,0%
Alternative 3: Komprimierung	29,9%	3,9%	10,0%	9,4%	34,1%	21,6%
Alternative 4: Clustering	12,0%	2,0%	10,0%	9,4%	52,0%	21,2%

Tabelle 6.3.: Vorhergesagte Ressourcenauslastung beim umlPSI-Verfahren (Mittelwert)

Die Auslastungen, die durch die Ausführung der Simulation ermittelt werden konnten, unterscheiden sich etwas von denen der anderen Verfahren. Die Inet-Ressource ist im Mittel bei den Modellierung mit diesem Verfahren nur zu 50% belastet, während die LAN-Verbindung mit knapp 10% im Vergleich zu den anderen Verfahren recht stark belastet erscheint. Die Auslastung der Ressourcen CPU und Disk reduziert sich bei der Alternative 4 (Clustering) erwartungsgemäß fast um die Hälfte.

Trotz der großen Abweichung der Durchlaufzeiten ist dennoch wieder ein Muster für die Durchlaufzeiten der Entwurfsalternativen erkennbar. Alternative 3 ist bei fünf von acht Lösungen die Variante mit der geringsten Durchlaufzeit, bei zwei Lösungen ist Alternative 4 am schnellsten, bei einer Lösung Alternative 1b (dynamischer Cache).

6.1.4. Messergebnisse der Implementierung

Der Testaufbau zur Performance-Messung der Implementierung, bei dem die verschiedenen Varianten des Webservers untersucht wurden, wurde in Abschnitt 5.6 näher erläutert. Es folgen nun die Ergebnisse dieser Messungen, wobei für die Durchlaufzeiten (Abbildung 6.6) und Auslastungen (Abbildung 6.7) des Webservers in jeder Variante die Mittelwerte der Testläufe vorgestellt und erklärt werden.

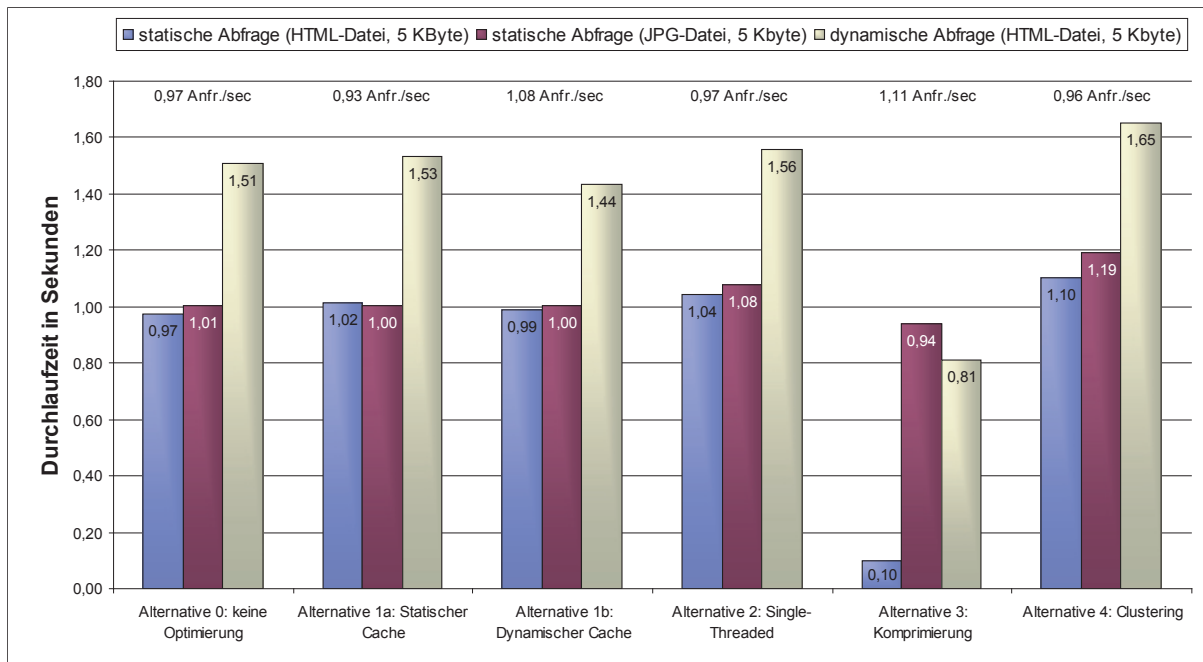


Abbildung 6.6.: Durchlaufzeiten des Webservers (Messung)

Bei der nicht-optimierten Version des Webservers sind die Durchlaufzeiten für beide Typen von statischen Seiten (HTML-Dokument und JPG-Bild von der Festplatte, jeweils 5 KByte groß) nahezu identisch, während die Anfrage für die dynamische Seite (HTML-Dokument aus der Datenbank, 5 KByte groß) im Vergleich dazu ziemlich genau 0,5 Sekunden länger dauert. Dies entspricht den Erwartungen und den Vorgaben der Aufgabenstellung des Experiments und lässt sich als Ausgangsbasis verwenden.

Zunächst fällt bei den Messungen mit dem statischen Cache auf, dass sich im Rahmen der Messgenauigkeit keinerlei Veränderung (eher eine Verschlechterung) der Durchlaufzeiten für die statischen Seiten im Vergleich zur nicht-optimierten Variante ergibt. Dies ist verwunderlich, da zumindest 70 Prozent der statischen Anfragen bei dieser Alternative aus dem Hauptspeicher beantwortet werden und keine teuren Festplattenzugriffe erfordern sollten. Zur näheren Untersuchung dieser Beobachtung wurde der Windows XP Performance Monitor verwendet, um die Anzahl der Lesezugriffe auf die Festplatte bei der nicht-optimierten Version des Servers zu messen. Dabei stellte sich heraus, dass bei den über 50 wiederholten Abfragen nur ein einziger Festplattenzugriff ganz am Anfang der Messung erfolgte, obwohl der statische Cache gar nicht aktiviert war. Die Ursache dafür liegt darin, dass das Betriebssystem Windows die abgerufenen Dateien bereits nach einmaligem Einlesen in einen Systemcache im Hauptspeicher ab-

legt, aus dem die folgenden Anfragen beantwortet werden. Die Implementierung des statischen Caches für den Webserver ist somit nutzlos, da dieses Caching bereits vom Betriebssystem durchgeführt wird. Eine Abschaltung des Systemcaches ist nicht durchführbar, da dieser tief im Betriebssystem verankert ist. Für die in der Implementierung benutzten Methoden aus der .NET-Klassenbibliothek zum Öffnen von Dateien ist dieser Cache vollkommen transparent. Die vorgeschlagene Entwurfsalternative eines statischen Caches erweist sich somit in der Praxis als nicht relevant, dies konnte jedoch durch die Performance-Analyseverfahren nicht vorhergesagt werden.

Beim Einsatz des dynamischen Caches ist erwartungsgemäß eine Reduktion der Durchlaufzeit für dynamische Seitenaufrufe zu verzeichnen (von 1,51 Sekunden auf 1,44 Sekunden). In den Rohdaten im Anhang I ist gut zu erkennen, dass der dynamische Cache nach ca. 80 Prozent der Anfragen aktiviert wird. Die Durchlaufzeit reduziert sich dann für die folgenden dynamischen Anfragen auf ca. 1 Sekunde und entspricht somit der Dauer der statischen Abfragen, die ebenfalls aus dem Hauptspeicher bezogen werden. In der Mittelung der Werte über gecachte und nicht gecachte Anfragen ergibt sich dann für diese Alternative der nur geringfügige Performance-Gewinn von 0,07 Sekunden. Dieser Wert ist eng an die Vorgabe von nur 20 Prozent gecachten Anfragen gebunden und somit stark abhängig von der jeweiligen Applikation. Erlaubt eine Web-Anwendung es, mehr dynamische Inhalte zu cachen, können sich kürzere Durchlaufzeiten ergeben, wobei die untere Grenze bei den Durchlaufzeiten für die statischen Seiten liegt. Für die statischen Seiten stellt sich bei dieser Variante keinerlei Performance-Verbesserung ein.

Der single-threaded Server (Alternative 2) zeigt im Vergleich zur nicht-optimierten multi-threaded Version des Webserver eine geringfügig höhere Durchlaufzeit. Die Anfragen stauen sich und es kommt zu höheren Wartezeiten, da immer nur eine Anfrage gleichzeitig vom Server verarbeitet werden kann. Bei der multi-threaded Version fallen die Zeiten für den Start von neuen Threads und den Kontextwechsel nicht so stark ins Gewicht, und es ergibt sich keine signifikante Verschlechterung der Performance. Der Server ist dazu in diesem Szenario noch zu wenig belastet, da sich maximal zwei Anfragen gleichzeitig im System befinden. Für den untersuchten Anwendungsfall bietet Entwurfsalternative 2 also keinen Performance-Vorteil gegenüber der nicht-optimierten Version des Webserver.

Teilweise sehr deutliche Performance-Gewinne ergeben sich hingegen bei Alternative 3 (Komprimierung). Sowohl die statische als auch die dynamisch erzeugte HTML-Seite konnten gut komprimiert werden, die Zeit für das Versenden der Dateien reduziert sich proportional zum Kompressionsfaktor. Bei der JPG-Datei ist nur eine geringfügige Komprimierung möglich, hier reduziert sich die Durchlaufzeit nur um wenige Millisekunden. Die Berechnungen, die für die Kompression angestellt werden, fallen hier kaum ins Gewicht, denn bei der Kompression kleiner Dateien arbeiten die verwendeten Algorithmen äußerst schnell.

Beim Clustering sind leichte Performance-Verluste festzustellen, die Durchlaufzeiten erhöhen sich bei allen abgefragten Seiten minimal. Dies ist durch die Verwendung des Schedulers zu erklären, der eine gewisse Zeit verbraucht, um die Anfragen auf die beiden Webserver zu verteilen. Das System ist in der vorliegenden Konfiguration noch nicht so stark ausgelastet, dass sich eine Verbesserung der Performance durch das Clustering ergeben könnte.

Die gemessenen Auslastungen (Abbildung 6.7) des Systems sind bei allen Varianten fast identisch. Die Festplattenbelastung ist kaum messbar, da, wie oben erwähnt, so gut wie keine Festplattenzugriffe erfolgen, weil die Dateien bereits im Windows-Systemcache liegen. Die CPU wird

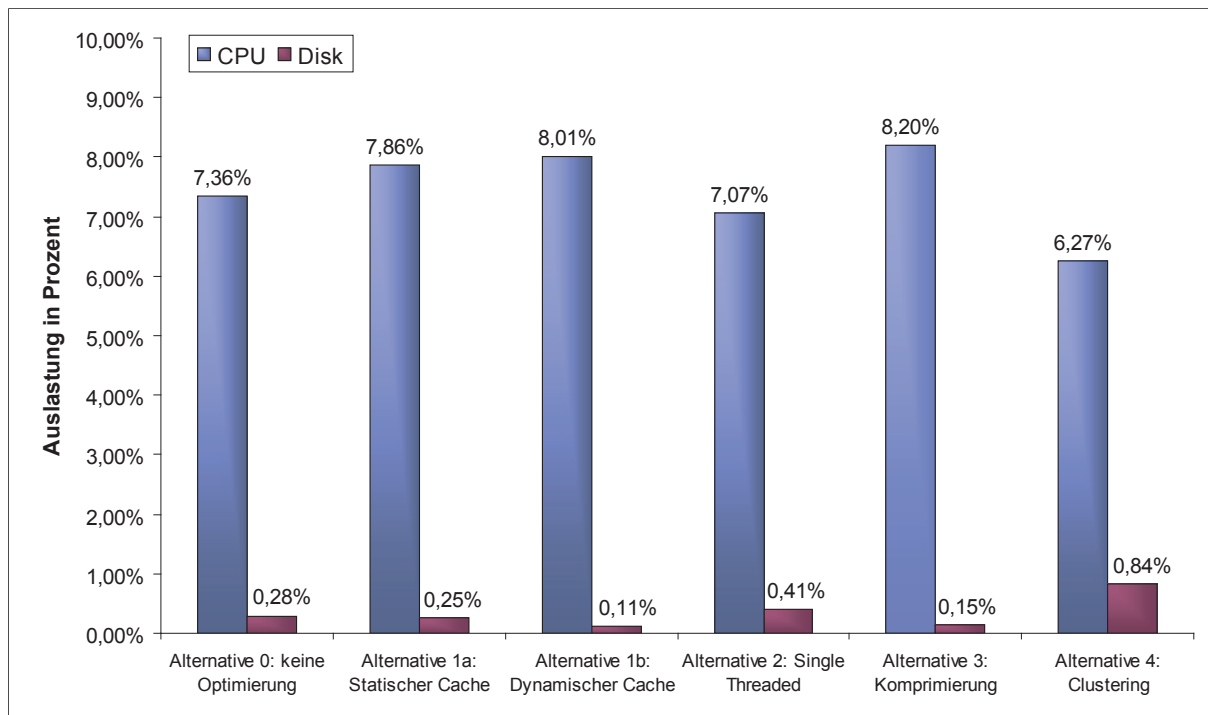


Abbildung 6.7.: Auslastungen des Webservers (Messung)

bei allen Varianten im Rahmen der Messgenauigkeit gleich wenig beansprucht. Beim single-threaded Server ist im Vergleich zur nicht-optimierten Variante eine leicht geringere Auslastung der CPU zu verzeichnen. Dies könnte aus der Tatsache resultieren, dass bei dieser Alternative immer nur eine Anfrage gleichzeitig beantwortet wird und die CPU nicht durch das Starten bzw. den Kontextwechsel von Threads beansprucht wird. Die geringere Auslastung wird jedoch mit den höheren Antwortzeiten erkauft. Beim Clustering ergibt sich ebenfalls eine Reduktion der CPU-Auslastung, was damit zu erklären sein könnte, dass die Last hier auf zwei Server verteilt wird. Allerdings sind die Abweichung der Auslastung sowohl beim single-threaded Server als auch beim Clustering so gering (etwa 1%), dass einfach Messungenauigkeiten vorliegen könnten, die bei einem solchen Versuchsaufbau nie ganz auszuschließen sind.

6.1.5. Vergleich von Vorhersagen und Messungen

In Abbildung 6.8 und 6.9 wurden die absoluten Durchlaufzeiten, die die Teilnehmer mittels der Performance-Analyseverfahren vorhersagten, und die Messungen an der Implementierung gegenübergestellt. Die Werte für die Verfahren sind jeweils die Mittelwerte über die Vorhersagen aller Teilnehmer. Beim SPE-Verfahren gibt es nur einen Wert sowohl für statische als auch dynamische Anfragen, dieser Wert konnte nicht nachträglich aufgespaltet werden und taucht daher in beiden Diagrammen auf.

Beim SPE-Verfahren und beim CP-Verfahren wurden die Durchlaufzeiten jeweils bei einer Benutzerankunftsrate von einem Benutzer pro Sekunde vorhergesagt. Beim umlPSI-Verfahren musste die Benutzerankunftsrate wegen der problematischen Simulation für jeden Teilnehmer individuell eingestellt werden. In den Messungen am Webserver konnten die Anfragen durch

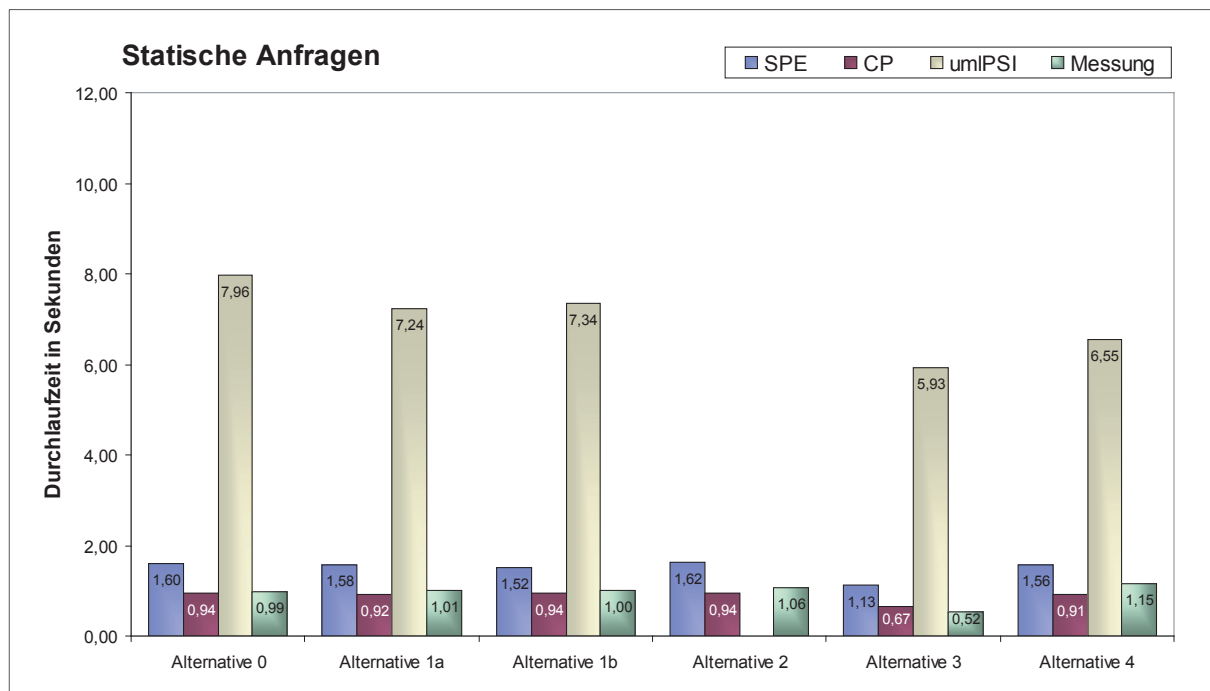


Abbildung 6.8.: Durchlaufzeiten bei den verschiedenen Verfahren (statisch)

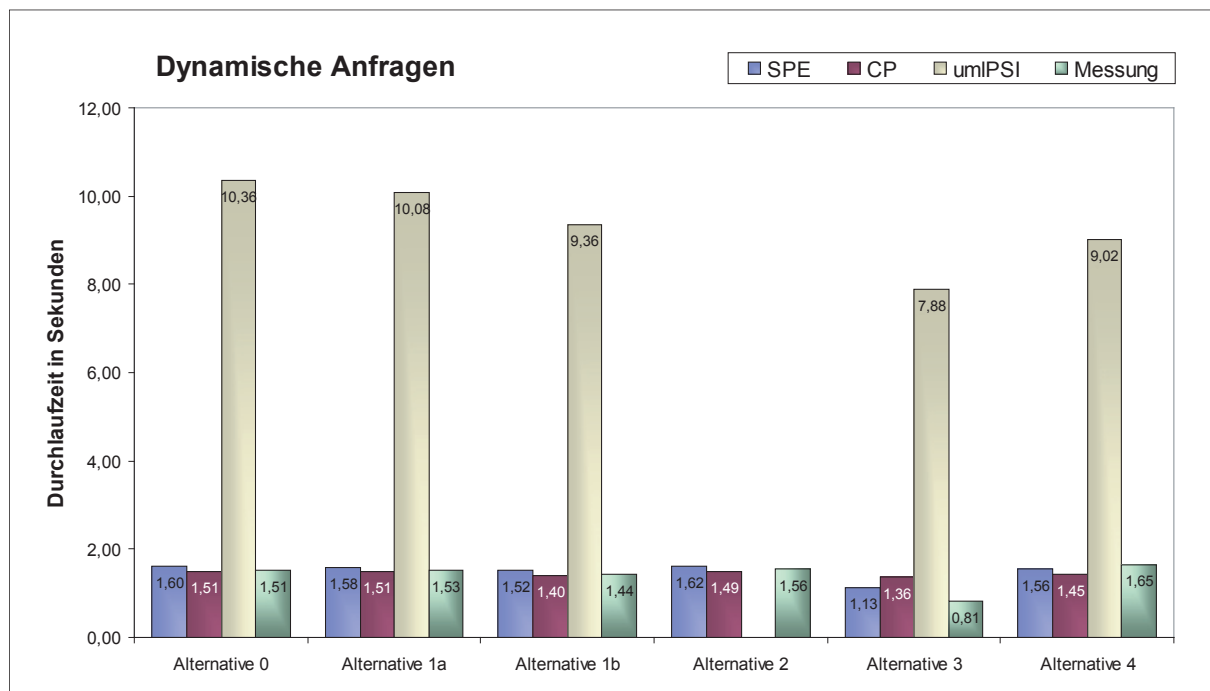


Abbildung 6.9.: Durchlaufzeiten bei den verschiedenen Verfahren (dynamisch)

das Testwerkzeug nicht exakt im Sekundentakt erzeugt werden, jedoch wurde versucht, die Abweichung möglichst gering zu halten. Messungen mit höheren Ankunftsrate zeigten in den Messergebnissen keine großen Abweichungen, so dass die erzielten Ergebnisse ausreichend für den Vergleich mit den Vorhersagen sind.

Sehr deutlich stechen in diesen Diagrammen die hohen Werte für das umlPSI-Verfahren hervor, während die Werte vom SPE- und CP-Verfahren nahe an den Messungen der Implementierung liegen. Die prozentualen Abweichungen der absoluten Durchlaufzeiten über alle Varianten des Webservers können in Tabelle 6.4 eingesehen werden.

Abweichung	SPE	CP	umlPSI	Messung
statische Seiten:	53,59%	9,46%	530,03%	0%
dynamische Seiten:	8,77%	11,93%	494,54%	0%

Tabelle 6.4.: Mittelwerte der Abweichung über alle Varianten

Die Autoren der Verfahren betonen oft, dass sich die Verfahren nur bedingt zur Vorhersage von absoluten Performance-Werten eignen. Dies scheint sich in den zum Teil großen Abweichungen der Verfahren zu bestätigen. Jedoch sollen Entwurfsalternativen bewertet werden können, deshalb erscheint eine Analyse der prozentualen Abweichungen der Werte für die einzelnen Entwurfsalternativen sinnvoll. Bei jedem Teilnehmer wird die Alternative 0 als Basis mit 100% Durchlaufzeit angesetzt und für die anderen Varianten des Webservers die gemittelte prozentuale Abweichung bestimmt. Die Ergebnisse finden sich in den Diagrammen 6.10 und 6.11. Hier liegen die Vorhersagen der Performance-Analyseverfahren und die Messungen wesentlich enger zusammen. Insbesondere bei Alternative 3 zeigt sich eine deutliche Reduktion der Durchlaufzeit, während die Abweichung bei den anderen Alternativen gering ist.

Die Abweichung bei den prozentualen Werten liegt nun sehr an den Messungen (Tabelle 6.5). Auch wenn die absoluten Zahlen z.T. stark von den Messungen abweichen, so ist bei den relativen Vergleichen der Entwurfsalternativen ein signifikantes Muster zu erkennen.

Abweichung	SPE	CP	umlPSI	Messung
statische Seiten:	7,24%	3,00%	7,63%	0%
dynamische Seiten:	5,97%	11,07%	12,11%	0%

Tabelle 6.5.: Mittelwerte der prozentualen Abweichung über alle Varianten

6.1.6. Entwurfsentscheidungen

Bei der Entscheidung für eine Entwurfsalternative sollten die Teilnehmer laut Aufgabenstellung alle verfügbaren Informationen berücksichtigen, z.B. auch Kosten, erwartete Komplexität der Implementierung und weitere denkbare Vor- und Nachteile. Die meisten Teilnehmer legten ungeachtet dessen ihrer Entscheidung die Durchlaufzeiten ihrer Modellierung zugrunde und empfahlen die Alternative mit der niedrigsten Durchlaufzeit. Dies war bei einem Großteil der Teilnehmer (72%) Alternative 3 (Komprimierung). Einer der Teilnehmer schrieb in seiner Begründung, dass er zunächst nicht damit gerechnet hatte, dass gerade Alternative 3 die schnellste Durchlaufzeit erwarten ließ. In seinem Fall hat die Anwendung des Verfahrens offensichtlich die

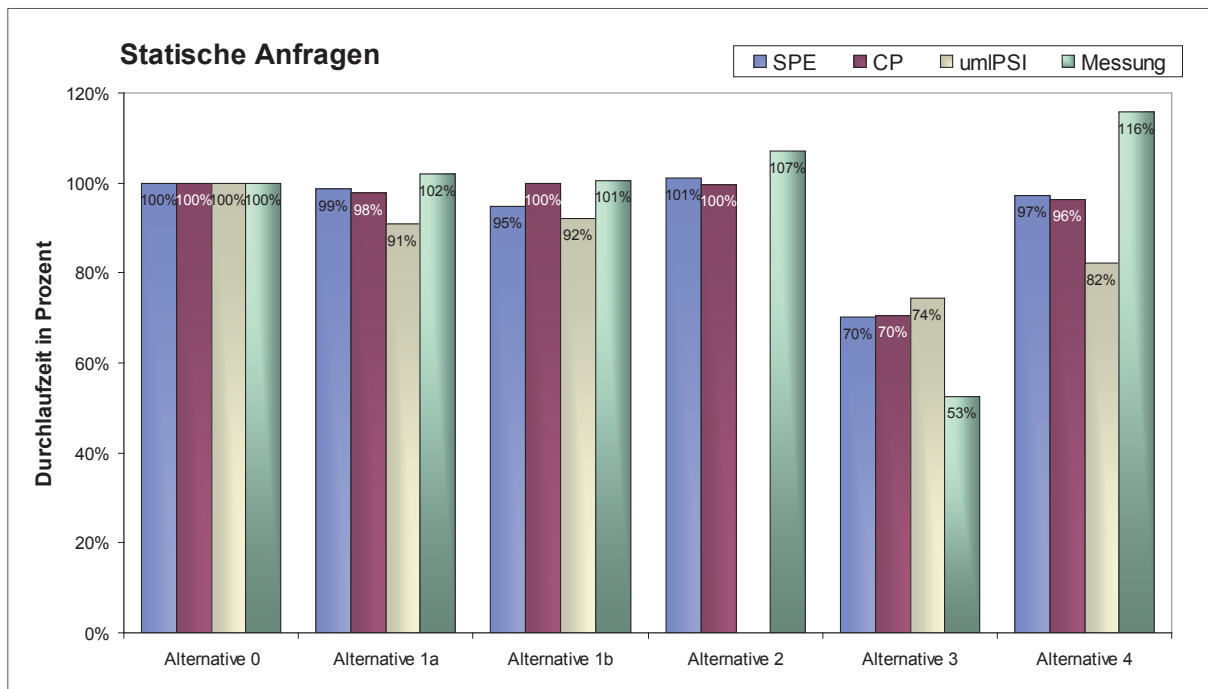


Abbildung 6.10.: Prozentuale Durchlaufzeiten (statisch)

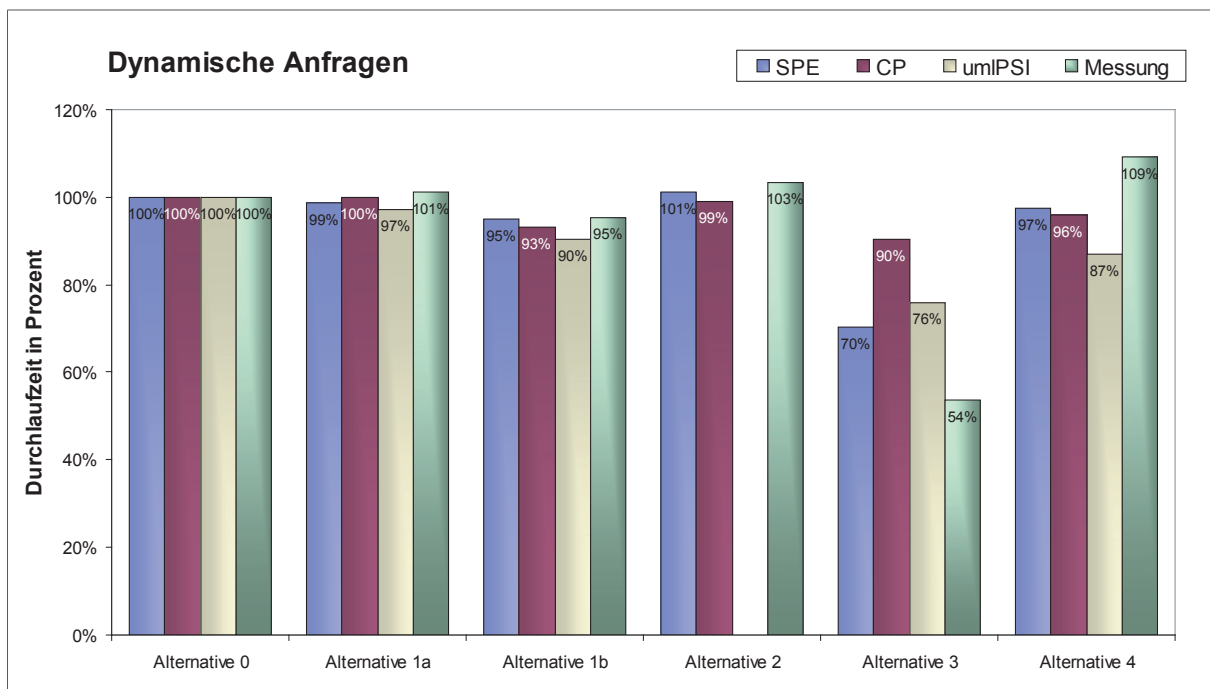


Abbildung 6.11.: Prozentuale Durchlaufzeiten (dynamisch)

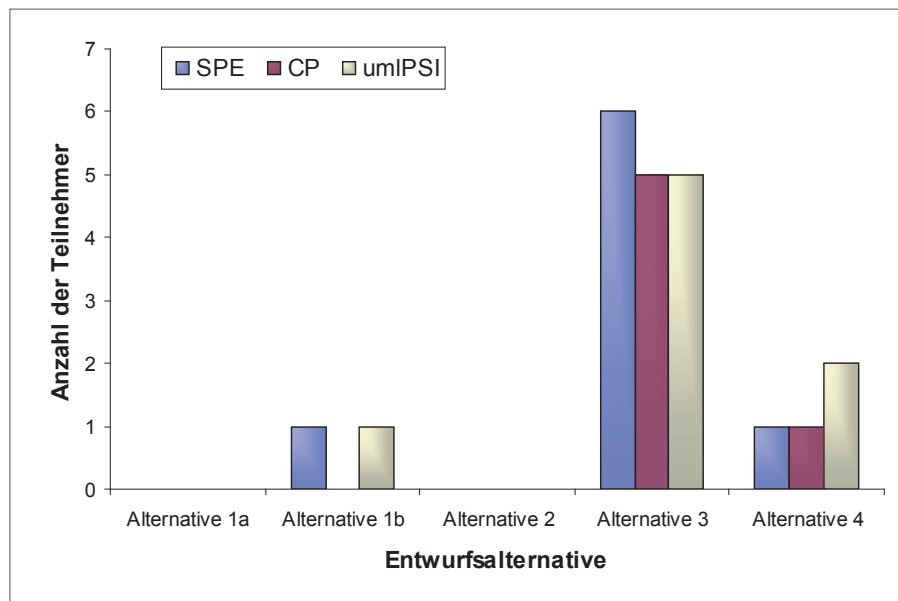


Abbildung 6.12.: Empfehlungen der Teilnehmer für eine Entwurfsalternative

Entwurfsentscheidung gegenüber der intuitiven Analyse beeinflusst.

Vier Teilnehmer schlugen Alternative 4 (Clustering) für die Implementierung vor, was ebenfalls mit den in der Modellierung berechneten geringsten Durchlaufzeiten begründet wurde. Ein Teilnehmer empfahl das Clustering, obwohl dies bei seiner Schätzung mit knappem Rückstand die zweitschnellste Variante war und begründet dies mit der höheren Ausfallsicherheit. Keiner der Teilnehmer argumentierte jedoch, dass die bei dieser Alternative reduzierten Werte für die Auslastung der Ressourcen des Webserver eine bessere Skalierbarkeit der Architektur über eine ansteigende Benutzerzahl erwarten lassen. Dies wäre jedoch auch ein durchaus relevantes Kriterium für eine leistungsstarke Architektur gewesen.

Da Alternative 2 (single-threaded Server) von den meisten Teilnehmern nicht modelliert werden konnte, gab es für diese Variante keine Empfehlungen. Ebenso wenig gab es Empfehlungen für Alternative 1a, da die Teilnehmer hier nur einen äußerst geringen Performance-Gewinn vorhersagten. Einige Teilnehmer schlugen vor, mehrere der Alternativen zu kombinieren und z.B. Alternative 1a und 3 oder 1a und 4 gleichzeitig zu implementieren, denn die Alternativen schlossen sich nicht gegenseitig aus. Auf dem Aufgabenblatt war jedoch deutlich vermerkt worden, dass nur genau eine Alternative ausgewählt werden sollte. Bei diesen Teilnehmern wurde daher von den beiden Alternativen jeweils die in das obige Diagramm aufgenommen, die die geringste Durchlaufzeit hatte. Komplementäre Alternativen waren aufgrund der Einfachheit der Webserver-Architektur beim Erstellen der Aufgabestellung nur schwierig zu entwickeln. Trotzdem sollte es realen Bedingungen entsprechen, dass aufgrund von Zeitdruck nur eine einzige Alternative implementiert werden kann.

Anhand der Durchlaufzeiten kann zusätzlich für jeden Teilnehmer eine Reihenfolge für die Entwurfsalternativen mit der besten Performance angegeben werden (Abbildung 6.13). Um ein Maß für die Abweichung der Reihenfolge der Vorhersagen mit der Messung zu konstruieren, wird bei jedem Teilnehmer angegeben, wie weit die Vorhersage seiner Platzierungen von der

Ranking Entwurfsalternativen:	Messung	SPE								CP						umlPSI							
		Teilnehmer 1	Teilnehmer 2	Teilnehmer 3	Teilnehmer 4	Teilnehmer 5	Teilnehmer 6	Teilnehmer 7	Teilnehmer 8	Teilnehmer 1	Teilnehmer 2	Teilnehmer 3	Teilnehmer 4	Teilnehmer 5	Teilnehmer 6	Teilnehmer 1	Teilnehmer 2	Teilnehmer 3	Teilnehmer 4	Teilnehmer 5	Teilnehmer 6	Teilnehmer 7	Teilnehmer 8
1. (schnellste Durchlaufzeit)	3	3	3	3	3	3	3	3	1b	3	3	3	3	1b	3	3	3	3	3	4	4	3	4
2.	1b	1b	1b	1b	1b	1b	1b	4	3	1b	1b	1b	1b	4	1b	1b	4	4	1b	1b	3	1b	1b
3.	1a	1a	1a	1a	1a	1a	1a	1b	1a	4	4	4	4	2	4	4	1b	1b	4	1a	1b	1a	1a
4.	2	4	4	2	4	4	4	1a	4	1a	1a	1a	1a	1a	1a	1a	1a	1a	3	1a	4	3	
5. (langsamste Durchlaufzeit)	4			4	2	2	2	2						3	2	2				2			
		Abweichung: (1 Punkt für jeden Platz)																					
		0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	4	4	0	4
		0	0	0	0	0	0	3	1	0	0	0	0	3	0	0	3	3	0	0	1	0	0
		0	0	0	0	0	0	1	0	2	2	2	2	1	2	2	1	1	2	0	1	0	0
		1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3	1	1	3
				0	1	1	1	1						4	1	1					1		
Summe:		1	1	0	2	2	2	6	3	3	3	3	3	10	4	4	5	5	3	7	8	1	7
Mittelwert: (über alle Teilnehmer)		2								4,3						5							

Abbildung 6.13.: Bewertung der Entwurfsalternativen nach Durchlaufzeit

Messung abweicht. Es wird z.B. eine Abweichung von 3 angegeben, wenn die Platzierung einer Entwurfsalternative um 3 Plätze von der der Messung abweicht. Es zeigt sich, dass die Reihenfolge der Bewertungen beim SPE-Verfahren im Mittel über alle Teilnehmer die geringste Abweichung von der Messung aufweist (2 Fehlplatzierungen), während die Werte beim CP-Verfahren (4,3) und umlPSI-Verfahren (5) deutlich höher liegen. Dabei spielt natürlich eine Rolle, dass die Abweichung in der Durchlaufzeit bei den hinteren Platzierung recht gering war und sich die Reihenfolge hier leicht ändern konnte. Bei den meisten Modellierungen liegt Entwurfsalternative 3 jedoch genau wie bei der Messung mit einem deutlichen Abstand in der Durchlaufzeit auf Platz 1 und wird gefolgt von Alternative 1b.

6.1.7. Interpretation und Erklärungsversuche

In Tabelle 6.6 finden sich nochmals zusammengefasst die Daten, die für die Metriken 1-4 aus den vorherigen Abschnitten bezogen werden konnten. Die in dieser Fallstudie aufgezeichnete Datenmenge reicht zwar nicht aus, um statistische Testverfahren wie z.B. einen t-test [Pre01] durchzuführen und Hypothesen zu belegen. Dennoch soll versucht werden, die Ergebnisse zu erklären, auch wenn sie sich möglicherweise stark auf den untersuchten Anwendungsfall beziehen und nur begrenzt auf andere Architekturen übertragbar sind.

Die vorhergesagten Durchlaufzeiten liegen bei den Verfahren, die nicht allein mit Messungen arbeiten (SPE, umlPSI), in den absoluten Zahlen zum Teil weit auseinander. Die Abweichungen kommen nicht durch die Analyse- oder Simulationstechniken der Verfahren zustande, sondern dadurch wie die Teilnehmer aus den gleichen Basisinformationen für den Webserver ihre Eingabedaten für die Verfahren bestimmten.

	Abweichung Eingabedaten verschiedener Teilnehmer (Metrik 1)	Abweichung Eingabedaten und Ausgabedaten (Metrik 2)	Abweichung Ausgabedaten und Messdaten (Metrik 3)	Prozentsatz richtiger Entwurfsentscheidungen (Metrik 4)
SPE	MW: 1,5999 s STABW: 0,7431 s	-	absolut: 53,59%, 8,77% relativ: 7,24%, 5,94% (niedrig/mittel)	75% Ranking-MW: 2
CP	dynamisch: MW: 1,5053 s STABW: 0,0001 s statisch: MW: 0,9449 s STABW: 0,0001 s	-	absolut: 9,46%, 11,93% relativ: 3,00%, 11,07% (niedrig)	83,3% Ranking-MW: 4,3
umlPSI	dynamisch: MW: 10,3608 s STABW: 6,0659 s statisch: MW: 7,9624 s STABW: 5,4411 s	gering	absolut: 530,03%, 494,54% relativ: 7,63% 12,11% (hoch)	62,5% Ranking-MW: 5

Tabelle 6.6.: Validität der Vorhersagen

Das SPE-Verfahren teilt die Performance-Indizes in "Software Resource Requirements" und "Computer Resource Requirements" ein. Letztere können meistens schon (wie auch in der Aufgabenstellung des Experiment) ohne eine Implementierung angegeben werden, da sie Hardware-Charakteristiken abbilden und die Software darauf keinen Einfluss hat. Schätzungen und Erfahrungswerte müssen dann nur bei den "Software Resource Requirements" einfließen, wodurch stark abweichende Endergebnisse vermieden werden. Die Größenverhältnisse bei den Berechnungszeiten einzelner Hardware-Ressourcen wie CPU, Festplatte oder Netzwerk können durch die Trennung besser eingehalten werden. Es kann z.B. angegeben werden, dass in einem Schritt auf die Festplatte zugegriffen und eine Netzwerknachricht versendet wird. Die Dauer dieser Aktionen wird dann aus den "Computer Resource Requirements" bezogen und ist immer die gleiche.

Beim umlPSI-Verfahren gibt es die Trennung von Software- und Hardware-Performancedaten nicht. Für jeden Schritt im Kontrollfluss muss die absolute Berechnungszeit angegeben werden. Die Dauer der gleichen Aktion kann an verschiedenen Stellen auch unterschiedlich sein, denn der Entwickler muss die Zeit anders als beim SPE-Verfahren jedes Mal neu angeben. Das SPE-Verfahren ist somit besser auf die Aufnahme von noch unsicheren Performance-Daten vorbereitet.

Die Teilnehmer des Experiments taten sich oft schwer damit, die Gültigkeit ihrer Ergebnisse zu prüfen. Nur wenige Teilnehmer betrachteten ihre Vorhersagen losgelöst vom Verfahren und von den Werkzeugen, was sich gut an den Inkonsistenzen in den ursprünglich unkorrigierten Lösungen beobachten ließ.

Beim SPE-Verfahren bemängelten einige Teilnehmer den Bedienkomfort des SPE-ED Werkzeuges und merkten an, dass die Ergebnisse des Werkzeuges mit einer Menge von Zahlen für sie keine Aussagekraft besäßen. Positiv für das Werkzeug kann jedoch vermerkt werden, dass die berechneten Zahlen direkt an die einzelnen Elemente in den Ausführungsgraphen geschrieben werden und die Schritte, die die meiste Zeit verbrauchen, im Ergebnis farbig markiert werden. Somit wird die Identifikation von Flaschenhälsen und Performance-Mängeln in der ursprünglichen Architektur visuell unterstützt und sollte recht intuitiv sein.

Für die teilweise sehr hohen Gesamtdurchlaufzeiten beim umlPSI-Verfahren von über 5 Sekunden gibt es außer der fehlenden Trennung von Hardware- und Software-Daten weitere mögliche Ursachen:

- **Erschwerte Überprüfung:** Die Teilnehmer konnten während des Experiments nur die Zeiten für einzelne Teilschritte in den Aktivitätsdiagrammen abschätzen, ihre komplette Modellierung jedoch nicht simulieren. Daher waren sie sich über die hohen Gesamtdurchlaufzeiten nicht im Klaren. Ohne die Gesamtzeit als Vergleichsbasis erschwerte sich die realistische Zeitabschätzung für die Teilschritte.
- **Fehlerhafte Angaben:** Bei einigen Teilnehmern konnten auch offensichtlich fehlerhafte Erwartungswerte beobachtet werden, wenn z.B. für das Lesen einer 5 KByte großen Datei von einer Festplatte mehrere Sekunden angesetzt wurden. Dabei sollte bekannt sein, dass moderne Festplatten Zugriffszeiten im Millisekundenbereich und hohe Übertragungsraten besitzen. Offensichtlich fehlte diesen Teilnehmern ein Gefühl für den Zeitverbrauch einzelner Ressourcen. Das Versenden einer 5-KByte-Datei über eine ISDN-Verbindung mit der Übertragungsrate von 8 KByte/s wurde bei einigen Teilnehmern des umlPSI-Verfahrens

mit mehreren Sekunden angegeben (anstatt 5/8 Sekunden). Hier konnten die Teilnehmer die Vorgaben der Aufgabenstellung nicht umsetzen.

- **Grobgranulare Angaben:** Bei der Angabe der Eingabedaten für das umlPSI-Verfahren in die Tagged-Values der UML-Diagramme war außerdem zu verzeichnen, dass die meisten Teilnehmer die Einzelschritte mit einer maximalen Genauigkeit von Zehntelsekunden angaben. Dabei sollte die Dauer einiger der enthaltenen Schritte, beispielsweise zum Parsen einer Anfrage und zu deren Weiterleitung, eher im Mikrosekundenbereich liegen. Aufgrund der zu grobgranularen Angaben steigerten sich in der Summe die Gesamtdurchlaufzeiten auf unrealistische Werte.

Beim Capacity Planning erfolgte die Schätzung der Nutzungsdauer von Hardware-Ressourcen bei einzelnen Anfragen und die der Systemlast bereits vor dem Experiment, da die Versuchsteilnehmer hier diese Parameter benötigten, um Berechnungen anzustellen. In diese Berechnungen flossen dann keine unsicheren Werte mehr ein, daher sind die Durchlaufzeiten bei diesem Verfahren bis auf Rundungsfehler identisch. Sofern die Angaben der Aufgabenstellung realistisch sind, sind die Teilnehmer also in der Lage Werte für die verschiedenen Alternativen zu berechnen. In die Modellierung der Entwurfsalternativen mussten dann auch Erwartungswerte der Teilnehmer einfließen, da z.B. keine Kompressionsrate bei Alternative 3 angegeben war. Allerdings war es bei den meisten Entwurfsalternativen möglich, die Vorgaben aus der Aufgabenstellung direkt in das Warteschlangenmodell zu übernehmen (z.B. 70% der Daten von der Festplatte werden gecacht). Daher gibt es auch bei der Modellierung der Entwurfsalternativen keine großen Abweichungen zwischen den Teilnehmern.

Da beim Capacity Planning nur eine Modellierung der Hardware-Ressourcen (als Warteschlangenmodell) vorgenommen wird, ist eine feingranulare Zuordnung z.B. der Dauer einzelner Schritte im Kontrollfluss der Software auf einer einzigen Ressource in diesem Verfahren nicht möglich. Es kann nur die Gesamtbelastung der einzelnen Hardware-Ressourcen angegeben werden, ein Modell für die Software fehlt.

Bei allen Verfahren äußerten einzelne Teilnehmer, dass sie sich beim Schätzen der Werte unwohl fühlten und kein richtiges Gefühl dafür entwickeln konnten. Insbesondere die Vorhersage von CPU-Zeit für die Ausführung bestimmter Aktionen fanden die Versuchsteilnehmer meist sehr schwierig. Ihnen hätte jedoch klar sein müssen, dass die genaue Vorhersage der CPU-Zeit bei der Architektur des Webserver gar nicht notwendig ist, sofern die Größenordnung in Relation zu Festplattenzugriffen oder Netzwerkverbindungen eingehalten wird. Dann fallen die Berechnungen der CPU bei der Angabe der Gesamtdurchlaufzeit kaum ins Gewicht, denn vom Webserver werden keine komplexen Algorithmen ausgeführt.

Smith und Williams [Smi02] erwähnen die Problematik des schwierigen Abschätzens ebenfalls in ihrem Buch und stellen fest, dass sich Anfänger meistens schwer damit tun. Sie empfehlen die Schätzungen mit Messungen an ähnlichen Systemen oder halbfertigen Prototypen zu belegen. Zusätzlich sollen, falls möglich, Performance-Experten bei der Bestimmung der Werte hinzugezogen werden. Eine Technik zur Abschätzung von Performance-Werten ist die Festlegung von best- und worst-case Schranken, also die Angabe von minimaler und maximaler erwarteter Bearbeitungsdauer. Auf diese Weise kann die Unsicherheit abgebaut werden. Wichtig ist in jedem Fall das Fokussieren auf die dominanten Performance-Werte und das Ausblenden von unwichtigen Details (beim Webserver z.B. die CPU-Zeit). Eine Faustregel besagt, dass 80% der

Berechnungszeit von Software in 20% des Programmcodes verbraucht werden. Diese 20% sollten im Zentrum der Performance-Modellierung stehen. Die Performance-Modelle können schrittweise erweitert und verfeinert werden, wenn der Entwurf der Software detaillierter wird oder bereits einzelne Komponenten implementiert wurden.

Auch wenn die Absolutwerte von den Messungen abweichen und nicht durchweg realistisch sind, so ist doch festzustellen, dass sich bei der Bewertung der Entwurfsalternativen signifikante Muster ergeben (siehe oben), so dass z.B. Alternative 3 bei den meisten Modellierungen die geringste Durchlaufzeit besitzt. Stellt man also die Unterschiede, die sich bei den Performance-Modellen durch die Entwurfsalternativen, den Unterschieden in der Implementierung gegenüber, so kommen die Vorhersagen der Verfahren den späteren Messungen an der Implementierung schon sehr nahe. Der Unterschied zwischen Vorhersage und Messung liegt dann nur noch bei unter 15 Prozent. Diese Genauigkeit ist für die Bewertung von Entwurfsalternativen ausreichend, was auch damit zu belegen ist, dass im Schnitt über alle Verfahren 72% der Teilnehmer die laut Messung leistungsstärkste Alternative 3 favorisiert haben.

Der Grad des Realismus der vorhergesagten Absolutwerte hängt eng mit dem Aufwand zusammen, der bei der frühen Performance-Analyse betrieben werden kann. Je genauer die Schätzungen sind und je mehr Messungen bereits vorgenommen werden können, desto besser stimmen Vorhersage und tatsächlich gemessener Wert anschließend überein. Der Aufwand für die Schätzungen und die Erstellung der Performance-Modelle muss so bemessen werden, dass er für die Bewertung von Entwurfsalternativen ausreicht.

6.2. Aufwand der Performance-Analyse

Mit der zweiten Forschungsfrage aus Kapitel 3 sollte der von Praktikern häufig kritisierte Zeitaufwand für frühe Performance-Analyse mit den Verfahren untersucht werden.

2. Wie hoch ist der Aufwand für die Performance-Analyse?

Der Aufwand für die Performance-Analyse muss in dieser Fallstudie zunächst in die Dauer für die Bearbeitung der Übung und die Dauer des eigentlichen Experiments unterteilt werden. Die Erfassung genauer Daten für den Zeitaufwand gestaltete sich aus organisatorischen Gründen schwierig. Die Übungssitzungen waren jedes Mal mit einer festen Zeit von zwei Stunden angesetzt und die Bearbeitung der Übungszettel erfolgte unter unkontrollierten Bedingungen bei den Testpersonen zu Hause. Daher muss sich bei der Analyse der Dauer auf deren Angaben verlassen werden. Für das Experiment war ebenfalls aus organisatorischen Gründen eine feste Bearbeitungsdauer von zwei Stunden vorgesehen, da die benutzten Rechner jeweils von den aufeinander folgenden Gruppen benötigt wurden.

Die nach Abschluss des Experiments von allen Teilnehmern erhobenen Angaben über ihre Bearbeitungszeit der Übungen können in Abbildung 6.14 eingesehen werden. Dabei sind die Angaben mit Vorsicht zu genießen, basieren sie doch auf den vagen Schätzungen der Teilnehmer. Dass die Bearbeitung des umlPSI-Übungszettels die meiste Zeit eingenommen hat, ist damit zu erklären, dass die Teilnehmer viel Zeit für die nicht erfolgreiche Installation des umlPSI-Werkzeugs aufgebracht haben. Ansonsten sind in den Bearbeitungen der Übungszettel keine signifikanten Unterschiede zu verzeichnen.

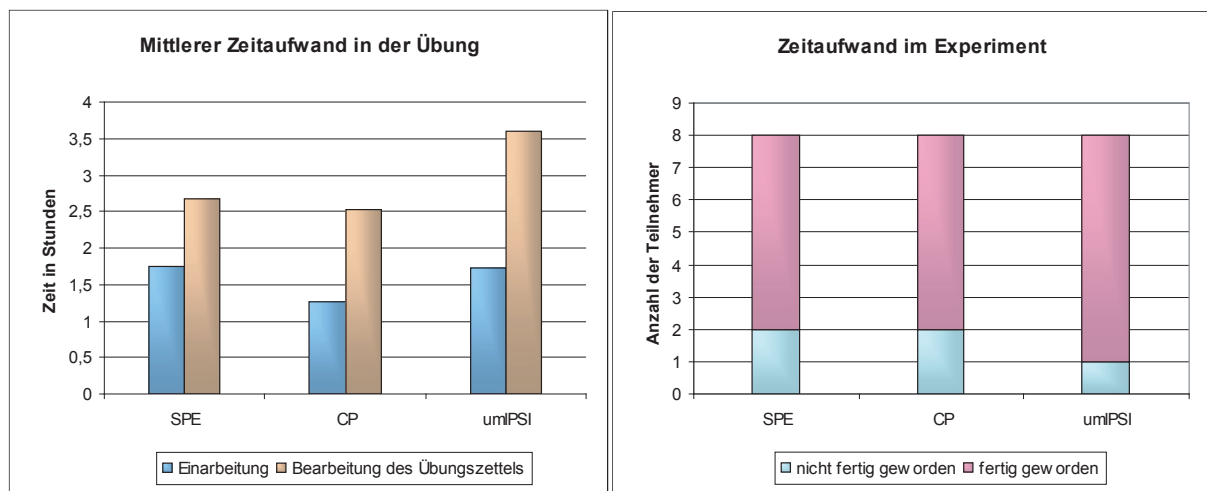


Abbildung 6.14.: Zeitaufwand in Übung und Experiment

Während des Experiments war ein so genannter "ceiling effect" zu verzeichnen, d.h. die Bearbeitungsdauer der Teilnehmer füllte die vorhandene Zeit. Keiner der Teilnehmer gab seine Lösung vor Ablauf der zwei Stunden ab, jedoch gab es in den Gruppen auch höchstens zwei Personen, die die Bewertung aller Entwurfsalternativen nicht in der vorgegebenen Zeit abschließen konnten. Damit zeigte, sich dass die Untersuchung der relativ einfachen Architektur des Webservers innerhalb von zwei Stunden gut durchführbar war.

Ob diese Ergebnisse über den Zeitaufwand der Performance-Analysen eine Aussagekraft über die Grenzen des Experiments hinaus besitzen, bleibt fraglich. Festzuhalten ist, dass eine zumindest grobe Vermittlung der wichtigsten Elemente der Verfahren durch die zweistündigen Übungssitzungen mit der Bearbeitung von Übungsblättern möglich war. Um die Verfahren in ihrer gesamten Breite vorzustellen und alle Möglichkeiten auszuschöpfen, ist sicherlich ein höherer Zeitaufwand erforderlich. Dazu sei auf [Dug04] verwiesen, wo ein SPE-Seminar an einem amerikanischen College über ein ganzes Semester vorgestellt wird. In einem solchen Seminar ist zusätzlich zur Vermittlung des SPE-Methodik auch genügend Zeit für Fallstudien und praktische Übungen vorgesehen, die zum Erlernen des Verfahrens unerlässlich sind. Auch über das Capacity Planning nach Menasce gibt es eigene Lehrveranstaltungen, die mit Zeitaufwand von einem vollen Semester angegeben werden. Das umlPSI-Verfahren ist mehr auf das eingesetzte Werkzeug fixiert und erfordert zur Anwendung nur Kenntnisse über das UML Performance Profile. Hinweise zur besseren Abschätzung von Performance-Werten oder allgemeine Performance-Prinzipien werden bei diesem Verfahren jedoch nicht angegeben.

Der Aufwand für die Erstellung der Performance-Modelle ist bei allen Verfahren flexibel anpassbar. Je mehr Daten zu frühen Entwicklungszeitpunkten feststehen, desto komplexere und genauere Modelle können erstellt werden. Das SPE-Verfahren ist recht gut auf die Verfeinerung der Modelle abgestimmt, da es die Entwicklung einer Software während des gesamten Lebenszyklus begleiten soll. Bei der Performance-Modellierung im Capacity Planning gibt es eine Reihe von speziellen Modellierungstechniken für z.B. gleichzeitige Ressourcennutzung oder lastabhängige Ressourcen in Warteschlangenmodellen. Oft erfordern diese Spezialfälle jedoch genaue Kenntnisse über die mathematischen Zusammenhänge. Hier muss vor dem Hintergrund der Zielsetzung der Performance-Analyse im Einzelfall abgewogen werden, ob sich eine komplexe Modellierung wirklich lohnt.

Genaue Untersuchungen zum Aufwand und den Kosten der Performance-Modellierung mit den Verfahren konnten im Rahmen der hier durchgeführten, begrenzten Fallstudie jedoch nicht angestellt werden. Dazu müssten größere Projekte in der Industrie mit Feldstudien über einen längeren Zeitraum untersucht werden. Auch das Abwägen zwischen Kosten der Modellierung und späteren Einsparungen durch die Reduktion nachträglicher Änderungen des Programmcodes war mit dem untersuchten Webserver nicht möglich.

6.3. Methodisches gegen intuitives Vorgehen

Eine weitere Forschungsfrage aus Kapitel 3 war zur Untersuchung der Effektivität und des Nutzens der Verfahren formuliert worden:

3. Welche Vorteile bringen Performance-Analyseverfahren gegenüber einem intuitiven Vorgehen?

Die Aufgabenstellung des Experiments wurde dazu zusätzlich an 17 weitere Informatik-Studenten ausgegeben. Diese hatten nicht am Experiment teilgenommen und besaßen keine Kenntnisse über die verwendeten Performance-Analyseverfahren. Auf diese Weise sollte untersucht werden, ob sich bei einer intuitiven Bewertung der Entwurfsalternativen Unterschiede gegenüber dem methodischen Vorgehen abzeichnen. Gleichzeitig konnte durch diese Kontrollgruppe getestet werden, ob die Aufgabenstellung einen offensichtlichen Vorteil einer Entwurfsalternative implizierte.

Die sieben abgegebenen Lösungen stammten von sechs Studenten der Informatik im Hauptstudium und einem Doktoranden, somit unterscheidet sich diese Gruppe bezüglich der Qualifikation nur unwesentlich von den Teilnehmern des Experiments. Die Teilnehmer gaben an, im Schnitt 40 Minuten für die Abgabe der Entwurfsempfehlung aufgewendet zu haben. Dabei merkten einige Teilnehmer an, dass die UML-Sequenzdiagramme in der Aufgabenstellung für sie bei der Performance-Analyse keinen Nutzen hatten. Die Diagramme waren in die Aufgabenstellung integriert worden, damit die Eingaben bei allen teilnehmenden Gruppen gleich waren und es zu keiner Verzerrung der Ergebnisse durch unterschiedliche Basisinformationen kommen konnte.

Vier der abgegebenen Lösungen favorisierten Entwurfsalternative 1b (dynamischer Cache), drei Teilnehmer entschieden sich für Alternative 3 (Komprimierung). Drei Teilnehmer gaben zusätzlich an, dass durch eine Kombination von Alternative 1b und 3 die beste Performance des Webservers zu erwarten wäre.

Natürlich ist eine Stichprobe von nur sieben Teilnehmern kaum aussagekräftig, trotzdem ist es auffällig, dass sich bei der intuitiven Performance-Analyse die Mehrheit der Teilnehmer (57%) für eine Entwurfsalternative entschieden, die im Experiment mit den Performance-Analyseverfahren nur von 9% der Teilnehmer gewählt wurde. Die Teilnehmer, die sich für Alternative 1b entschieden, identifizierten die Anbindung des externen Servers mit einer Verzögerung von 0,5 Sekunden als den Flaschenhals der Architektur. Dass das Versenden der 5-KByte-Dateien an die Clients über eine ISDN-Leitung mindestens 5/8 Sekunden in Anspruch nimmt (dazu addiert sich noch die Übertragung von Header-Daten), taucht in fast keiner Begründung auf. Die starke Applikationsabhängigkeit von Alternative 1b wurde ebenfalls kaum herausgestellt.

Damit ist zumindest gezeigt, dass die Aufgabenstellung des Experiments bei einem intuitiven Vorgehen nicht automatisch zur Entscheidung für Alternative 3 führte und die Anwendung eines Verfahrens auf die Architektur sinnvoll ist. Weitere Schlüsse lassen sich aufgrund der geringen Teilnehmerzahl der Kontrollgruppe nicht ziehen. Festzuhalten bleibt, dass die Verfahren gegenüber einem intuitiven Vorgehen bei der frühen Performance-Analyse den Vorteil zur feingranularen Aufschlüsselung von Performance-Werten bieten. Einzelne Schritte im Kontrollfluss können mit einer vorhergesagten Bearbeitungsdauer versehen werden. Gerade bei komplexeren Architekturen sollte es mit den Modellen der Verfahren besser möglich sein, den Überblick zu behalten und leistungshemmende Flaschenhälse zu identifizieren.

6.4. Bewertung einzelner Eigenschaften

Eine detaillierte Bewertung einzelner Eigenschaften der Performance-Analyseverfahren war die Zielsetzung der Frage 4 aus Kapitel 3:

4. Wie sind die Eigenschaften der Performance-Analyseverfahren zu bewerten?

Dazu sollen nun die dort aufgestellten Kriterien an den Verfahren untersucht werden.

6.4.1. Notation

Mächtigkeit:

Das SPE-Verfahren verwendet ein Softwaremodell, mit dem die Eigenschaften des Kontrollflusses der Software modelliert werden, und ein Systemmodell auf Basis von Warteschlangenmodellen, mit dem Vorhersagen bei mehreren gleichzeitigen Benutzern oder Prozessen auf einer Ressource getroffen werden können. Das Software-Modell besteht aus Ausführungsgraphen, mit denen Schleifen und alternative, optionale oder parallele Schritte im Kontrollfluss ausgedrückt werden können. Den Teilnehmern fiel es im Experiment schwer, mit dieser Notation mehrere Programmthreads zu modellieren. Auch in [Smi02] und im Handbuch zum SPE-ED Werkzeug finden sich keine expliziten Angaben für eine solche Modellierung. Die Warteschlangenmodelle des Systemmodells werden vom SPE-ED-Werkzeug vollständig gekapselt und sind für den Benutzer nur indirekt modifizierbar. Es können offene und geschlossene Benutzerklassen modelliert werden, diese können auch gemeinsam in einem Modell vorkommen und werden dann jeweils als einzelnes Szenario dargestellt. Mehrere Szenarios auf einem Rechnersystem können durch eine Simulation ausgewertet werden. Im SPE-ED-Werkzeug ist bereits ein "Advanced System Model" vorgesehen, jedoch noch nicht implementiert. Mit diesem soll es möglich sein, verschiedene Warteschlangendisziplinen oder passive Hardware-Ressourcen zu modellieren. Smith/Williams weisen darauf hin, dass bei der frühen Performance-Analyse aufgrund der noch unsicheren Informationen meist keine detaillierten Modellierungen gemacht werden können bzw. müssen. Der Fokus beim SPE-Verfahren liegt klar auf der möglichst frühen Performance-Analyse, deshalb ist es das Verfahren, das mit den wenigsten Eingaben bereits eine Analyse liefern kann.

Beim Capacity Planning gibt es kein Softwaremodell, sondern nur ein Systemmodell bestehend aus Warteschlangenmodellen, das der Benutzer anders als beim SPE-Verfahren auch direkt abändert. Eine explizite Modellierung des Kontrollflusses innerhalb einer Ressource ist hier nicht vorgesehen, es kann für jede Ressource nur die Gesamtnutzungszeit während eines Szenarios angegeben werden. Das Verfahren ist Hardware-nah und klammert Aspekte der Software fast völlig aus. Für die Warteschlangennetzwerke werden in den zugehörigen Büchern eine Reihe von speziellen Modellierungen z.B. für offene, geschlossene und gemischte Benutzerklassen, blockierende oder lastabhängige Ressourcen sowie verschiedene Warteschlangendisziplinen (First Come, First Serve, Priority Queueing, Round Robin, Processor Sharing) angegeben. Für einige dieser Modelle gibt es jedoch keine Werkzeugunterstützung, so dass zunächst die mathematischen Hintergründe erlernt und die Modelle dann manuell ausgewertet werden müssen. Im Einzelfall muss abgewogen werden, ob sich der erhöhte Aufwand für die genaueren Performance-Modellierungen tatsächlich bezahlt macht oder ob schon einfachere Modelle aussagekräftige Ergebnisse liefern. Für die Analyse mit diesem Verfahren muss eine Implementierung oder zumindest ein Prototyp

verfügbar sein, damit die benötigten Messungen angestellt werden können. Deshalb eignet sich dieses Verfahren weniger für die frühe Performance-Analyse während der Entwurfsphasen einer Software, sondern besser zur Kapazitätsanalyse bereits bestehender Systeme.

Das umlPSI-Verfahren benutzt zur Darstellung des Kontrollflusses Aktivitätsdiagramme aus der UML. Sie unterstützen ähnliche Konstrukte wie die Ausführungsgraphen beim SPE, d.h. es können Schleifen sowie parallele Schritte im Kontrollfluss abgebildet werden. Für einzelne Aktivitäten können unterschiedliche Wahrscheinlichkeitsverteilungen der Performance-Werte angegeben werden. Damit ist man nicht, wie bei den anderen Verfahren, ausschließlich an Exponentialverteilungen gebunden, denn die Simulation kann mit beliebigen Verteilungen arbeiten. Die Struktur der Hardware bzw. die Zuordnung von Software zu Hardware wird mit Verteilungsdiagrammen realisiert. Dort können aktive oder passive Ressourcen verwendet und ein Faktor für die Geschwindigkeit einer Ressource angegeben werden. Die Zeit für die Bearbeitung auf einer Ressource wird in den Aktivitätsdiagrammen festgelegt, wo einzelne Aktivitäten jeweils bestimmten Ressourcen aus dem Verteilungsdiagramm zugeordnet sind. Wünschenswert wäre eine konkretere Darstellung von Hardware-Ressourcen z.B. mit Vorlagen für bestimmte Rechnerkomponenten wie Prozessoren, Festplatten oder Netzwerkverbindungen. Jedoch ist das Verfahren hier durch die Möglichkeiten der UML eingeschränkt. Weiterhin könnten auch andere Diagrammtypen der UML berücksichtigt werden, so dass z.B. auch Komponenten- oder Zustandsdiagramme simuliert werden könnten.

Strukturierung:

Mit dem SPE-ED Werkzeug können einzelne Schritte in Ausführungsgraphen durch Subgraphen verfeinert werden. Insgesamt ist eine Hierarchisierung der Subgraphen über 27 Ebenen möglich. Damit ist die Komplexität der Performance-Modellierung gut dem aktuellen Stand der Entwicklung des Systems anpassbar, denn das Modell kann nach und nach verfeinert werden.

Beim Capacity Planning sind keine mehrfachen Ebenen in den Warteschlangenmodellen vorgesehen, allerdings sind diese Modelle von der Anzahl der Elemente her auch meist nicht so komplex wie z.B. die Ausführungsgraphen beim SPE, da ausschließlich Hardware-Ressourcen modelliert werden.

Das umlPSI-Verfahren lässt die Modellierung beliebig vieler Anwendungsfälle zu, die dann jeweils mit Aktivitätsdiagrammen näher beschrieben werden. Dabei ist es möglich, mehrere Anwendungsfälle in einen einzigen Anwendungsfall zu kapseln und somit beliebig viele Aktivitätsdiagramme nacheinander oder nebenläufig zu modellieren.

Aufnahme von Performance-Indizes:

Mit Performance-Indizes ist hier gemeint, auf welche Weise Informationen wie Bearbeitungsdauer und Ressourcenverbrauch in die Performance-Modelle gelangen.

Beim SPE-Verfahren werden die Performance-Indizes zerlegt in "Software Resource Requirements" und "Computer Resource Requirements" (oder auch "Processing Overhead"). Deren Typ kann beliebig definiert werden (z.B. als Anzahl von Netzwerknachrichten oder Bearbeitungsdauer bei einem Festplattenzugriff). Die Software-Ressourcen verbrauchen jeweils eine bestimmte Zeit auf einer oder mehreren Hardware-Ressourcen. Die Zuordnung wird über eine Tabelle vorgenommen, in der zusätzlich festgelegt wird, welche Zeit eine Hardware-Ressource für eine

Elementaroperation benötigt. Die Aufteilung in diese beiden Gruppen erscheint sinnvoll, insbesondere dann, wenn die Daten für Hardware-Ressourcen von einem Performance-Experten oder System-Administrator geliefert werden können. In diesem Fall ist es auch weniger erfahrenen Entwicklern möglich, Performance-Vorhersagen durchzuführen.

Da es beim Capacity Planning keine graphische Darstellung der Warteschlangenmodelle gibt, können auch keine Performance-Indizes direkt an die Modelle angefügt werden. In die Excel-Tabellen für Analyse von Warteschlangenmodellen werden die Benutzerankunftsrate, die Anzahl und Art der Ressourcen und die jeweilige Bearbeitungsdauer auf einer Ressource eingetragen.

Am weitesten fortgeschritten und an einen Standard gebunden sind die Performance-Indizes beim umlPSI-Verfahren, wo das UML Performance Profile [OMG03a] verwendet wird. Durch die Befolgung dieses Standards (mit kleineren Abweichungen) könnten die mit Performance-Werten annotierten UML-Diagramme theoretisch auch durch andere Werkzeuge als umlPSI analysiert werden. Problematisch ist die zunächst etwas umständliche Syntax der Tagged Values im UML Performance Profile, die bei den Übungen zu vielen Fehlern führte. Mit dieser ist es möglich unterschiedliche Wahrscheinlichkeitsverteilungen für die Performance-Werte anzugeben und zu vermerken, ob es sich bei den Daten um gemessene oder geschätzte Daten handelt. Die Eingabe der Tagged Values sollte nach Möglichkeit von Software-Werkzeugen gekapselt werden und direkt beim Eintippen auf Syntax-Konformität geprüft werden. Dies war in dem benutzten Modellierungswerkzeug Poseidon jedoch nicht der Fall. Trotzdem scheint das UML Performance Profile die ideale Basis für Analysewerkzeuge zu sein, die UML-Diagramme auswerten.

6.4.2. Werkzeugunterstützung

Benutzungsoberfläche, Ergonomie:

Das SPE-ED Werkzeug wurde bereits 1996 programmiert und seitdem nur unwesentlich verändert. Daher ist die Benutzungsoberfläche etwas veraltet und entspricht nicht mehr den heutigen Standards. Die Eingabe der Ausführungsgraphen ist etwas umständlich und unterscheidet sich von gängigen Graphikeditoren. Zum Löschen einzelner Elemente muss z.B. ein anderer Bearbeitungsmodus eingestellt werden. Auch gibt es kein Drag&Drop bei der Erstellung der Modelle oder Karteikartenreiter zur besseren Übersicht bei vielen Subgraphen. Eine Anbindung an gängige Modellierungswerkzeuge ist nicht vorgesehen, jedoch soll das Programm in späteren Versionen den Export der Modelle in ein standardisiertes Dateiformat für Performance-Modelle erlauben (Performance Model Interchange Format, PMIF [SL04]). Die Autoren betonen, dass es sich bei SPE-ED um kein Zeichenwerkzeug handelt, sondern um eine Schnittstelle zur Auswertung von Performance-Modellen.

Für das Capacity Planning liegt eine Reihe von Spreadsheets vor, die mit Tabellenkalkulationen wie Microsoft Excel ausgewertet werden können. Darin enthalten sind z.B. Vorlagen zur Lösung von Warteschlangenmodellen oder zur Cluster-Analyse. Der Bedienkomfort hängt daher von der eingesetzten Tabellenkalkulation ab.

umlPSI besitzt keine graphische Oberfläche und wird über die Kommandozeile gestartet. Die Ergebnisse der Simulation werden direkt in die XML-Dateien zurückgeschrieben, die zuvor als Eingabe verwendet worden waren. Diese können dann vom Benutzer wieder in sein Modellierungswerkzeug eingelesen werden, wo eine sofortige Zuordnung von Performance-Problemen im

UML-Modell möglich ist. Beim Modellierungswerkzeug ist man in der aktuellen Version von umlPSI an Poseidon der Firma Gentleware gebunden, da nur die von diesem Programm erzeugten XMI-Dateien in einem bestimmten Format geparkt werden können. Poseidon bietet die üblichen Möglichkeiten zur Modellierung von UML-Diagrammen. Stereotypen oder Tagged Values können über Karteikartenreiter für jedes UML-Element eingegeben werden.

Robustheit gegenüber Fehleingaben:

Das SPE-ED Werkzeug markiert die Schritte im Kontrollfluss mit einer hohen Bearbeitungsdauer oder stark ausgelastete Ressourcen farbig, somit sind Flaschenhälse in einer Architektur genauso wie fehlerhafte Modellierungen leichter zu erkennen. Teilweise gab es Probleme mit dem Werkzeug wenn Benutzer Kommata statt Punkte (amerikanischen Notation) bei Kommazahlen verwendeten oder die Werte in den Computer Resource Requirements falsch angaben. Eine Validitätsüberprüfung für Benutzereingaben bietet das Werkzeug nicht, auch unrealistische Werte werden ohne Rückfragen übernommen.

Beim Capacity Planning ist nach initialer Erstellung des Performance-Modells eine schnelle Anpassung und erneute Lösung bei evtl. fehlerhaften Werten möglich, eine Überprüfung der eingegebenen Werte findet jedoch ebenfalls nicht statt.

Neben den oben erwähnten Mängeln bei der Eingabe der Performance-Indizes liefert das umlPSI-Werkzeug oft unverständliche Fehlermeldungen, falls ein Modell nicht simulierbar ist. Teilweise wurden aber auch Syntaxfehler in den Tagged Values bestimmter Aktivitäten gemeldet, so dass die Fehler genau zuzuordnen waren.

Stabilität:

SPE-ED läuft in der vorliegenden Version recht stabil, sehr selten gab es Darstellungsfehler oder Programmabstürze. Das Speichern der Performance-Modelle ist manchmal nicht ganz nachvollziehbar, teilweise wurde nicht der angezeigte Inhalt abgespeichert, und die aktuell bearbeiteten Szenarien gingen beim Beenden des Programms verloren.

Die Spreadsheets beim Capacity Planning zeigten sich als stabil in der Ausführung, waren jedoch nicht unter Open Office zu verwenden, da dieses die enthaltenen Visual Basic Makros nicht verarbeiten konnte.

umlPSI war im Test nicht besonders stabil gegenüber aufwändigen Simulationen. Teilweise dauerte die Ausführung mehrere Stunden oder wurde nach langer Zeit ergebnislos und mit unverständlichen Fehlermeldungen abgebrochen. Bei diesem Werkzeug handelt es sich nicht um ein marktreifes Produkt, sondern um eine Demonstration für die Methode.

Präsentation der Ergebnisse

Die einzelnen Schritte in den Ausführungsgraphen in SPE-ED werden direkt mit den berechneten Durchlaufzeiten und Auslastungen der Analyse beschriftet. Damit kann sehr feingranular aufgeschlüsselt, welche Abschnitte im Kontrollfluss besonders performance-relevant sind. Schritte mit einer hohen Durchlaufzeit werden zusätzlich mit unterschiedlichen Farben markiert, wodurch Flaschenhälse in der Architektur gut identifiziert werden können. Eine Rückabbildung der Ergebnisse in das ursprüngliche Software-Modell (z.B. in UML) ist bei diesem Verfahren jedoch

nicht vorgesehen und muss manuell vorgenommen werden.

Das Capacity Planning liefert die Länge von Warteschlangen, Auslastungen der Ressourcen sowie Durchlaufzeiten in Tabellenform zurück. Dies kann bei komplexeren Architekturen eine große Menge von Zahlen sein. Es werden jeweils Gesamtzeiten und Gesamtauslastung der Ressourcen über einen Anwendungsfall angegeben. Da kein Software-Modell verwendet wird, ist auch eine Rückabbildung der Performance-Ergebnisse nicht möglich.

umlPSI stellt die simulierten Performance-Ergebnisse ebenfalls als Tabelle dar. Enthalten sind Auslastungen und Durchsatz von Ressourcen sowie Durchlaufzeiten für Anwendungsfälle und einzelne Aktionen aus den Aktivitätsdiagrammen. Dies ist das einzige hier untersuchte Verfahren, das die Ergebnisse auch in das ursprüngliche Software-Modell zurückmeldet. Die Zeiten bzw. Auslastungen werden in bestimmte Tagged Values in die XMI-Dateien der ursprünglichen UML-Diagramme geschrieben, die dann beim erneuten Einlesen in das UML-Modellierungswerkzeug (hier: Poseidon) untersucht werden können. Somit können Entwickler aufgrund dieser Daten sofort Änderungen an den UML-Diagrammen vornehmen und die Simulation wiederholen. Wünschenswert wäre hier auch eine farbige Markierung von Flaschenhälsen in der Architektur, dies hängt jedoch von dem eingesetzten Modellierungswerkzeug ab.

Praxistauglichkeit

Das SPE-ED-Werkzeug ist ein ausgereiftes Produkt, dass zur Performance-Analyse in der Praxis eingesetzt werden kann. Problematisch ist der Bruch zwischen Software-Modell und Performance-Modell, da dieses Werkzeug eigenständig und unabhängig von der Software-Modellierung einsetzt wird. Eine Integration dieses Werkzeugs mit anderen CASE-Werkzeuge und Entwicklungsumgebungen ist nicht vorgesehen. Die Benutzungsoberfläche von SPE-ED ist leicht veraltet und es liegt nur eine Windows-Version vor, andere Plattformen werden nicht unterstützt. Frei verfügbar ist SPE-ED nur in einer eingeschränkten Demonstrationsversion, die Vollversion ist kostenpflichtig, wobei Preisangaben nicht verfügbar waren.

Die Spreadsheets für das Capacity Planning sind im Rahmen der Bücher von Menasce und Almeida verfügbar. Sie können aus dem Internet frei heruntergeladen und mit Passwörtern aus den Büchern freigeschaltet werden. Zur Auswertung von Warteschlangenmodellen sind diese Dateien gut brauchbar, jedoch fehlt eine graphische Aufbereitung der Ergebnisse. Viele der Experimentteilnehmer bevorzugten die Bearbeitung der Spreadsheets vor den anderen Werkzeugen.

Das Werkzeug umlPSI ist ebenfalls frei im Internet verfügbar. Praxishemmend ist die Tatsache, dass dieses Programm nur auf Linux-Rechnern installiert werden kann. Vom neuesten GNU C-Compiler kann es nicht kompiliert werden und es werden weitere Bibliotheken (z.B. XML-Bibliothek für GNOME) vorausgesetzt. Die Stabilität und Ausführungsgeschwindigkeit des Werkzeugs ist nicht befriedigend. Der Ansatz, die Performance-Analysen eng mit einem CASE-Werkzeug zur Modellierung vom UML-Diagrammen zu verbinden, erscheint jedoch sehr sinnvoll auch wenn er hier nicht konsequent umgesetzt wurde. Der Entwickler muss das Werkzeug noch extern von der Modellierungsumgebung starten und die Ergebnisdateien nach der Simulation wieder manuell öffnen.

6.4.3. Eignung für verschiedene Domänen

In [Smi02] finden sich mehrere Anwendungsbeispiele des SPE-Verfahrens für unterschiedlichen Typen von Softwaresystemen. So liegt der Schwerpunkt zumeist auf verteilten Systemen und insbesondere Web-Applikationen. Zusätzlich wird demonstriert, wie die Konzepte auf eingebettete Realzeitsysteme übertragen werden können. Dabei ist man prinzipiell nicht an ein Programmierparadigma gebunden, obwohl meist Sequenzdiagramme der UML als Eingabe dienen, und somit objektbasierte Systeme untersucht wurden.

Das Capacity Planning ist ebenfalls vorwiegend für verteilte Systeme wie Client/Server-Systeme durchgeführt worden. In [MA02] werden speziell Webserver bzw. Web-Applikationen betrachtet, während in [MA00] E-Business Anwendungen im Vordergrund stehen.

Das umlPSI-Verfahren ist durch die Modellierung in UML recht stark an die objektorientierte Entwicklung gebunden. Prinzipiell eignet es sich jedoch auch für komponentenbasierte Systeme.

6.4.4. Auswertungsmethode

Das SPE-Verfahren bietet insgesamt drei unterschiedliche Auswertungsmethoden.

- Mit der "No Contention Solution" werden die Softwareausführungsmodelle ausgewertet und die abgelaufene Zeit für einen einzigen Benutzer in einem Szenarios geliefert. Damit kann ein anfänglicher Vergleich der unteren Schranke der Antwortzeit mit den zuvor festgelegten Performance-Zielen angestellt werden. Zur Berechnung werden die Anfragen an die Software-Ressourcen mit den Werten in den "Computer Resource Requirements" multipliziert, wobei Übergangswahrscheinlichkeiten zwischen Zuständen und Anzahl von Schleifenwiederholungen einfließen.
- Die "Contention Solution" wertet das Systemmodell (also ein Warteschlangenmodell) für mehrere Benutzer in einem Szenario aus. Dazu muss die Benutzerankunftsrate oder die Anzahl der Benutzer im System angegeben werden. Bei dieser Analyse werden die Wartezeiten, die sich durch die Konkurrenz mehrerer Benutzer um eine Ressource ergeben, einbezogen. Die "No Contention Solution" dient bei dieser Auswertung als Eingabe, um die Berechnungsanforderungen an die einzelnen Hardware-Ressourcen anzugeben. Für die "Contention Solution" kommt der rekursive Mean-Value-Anaylsis-Algorithmus zum Einsatz, dessen Berechnung in Sekundenschnelle erfolgt.
- Zusätzlich können mit der "System Model Solution" die Antwortzeiten für mehrere Benutzer und mehrere gleichzeitige Szenarien auf einer Architektur berechnet werden. Dazu führt das SPE-ED-Werkzeug eine CSIM-Simulation [Sof04] des Systemmodells mit mehreren Szenarien durch. Auch hier dienen die Ergebnisse der "No Contention Solution" als Eingabe. Die Berechnungsanforderungen an alle Hardware-Ressourcen müssen exponential verteilt sein, und für jeden Server wird ein Prioritäten-Scheduling verwendet, wozu für jedes Szenario eine bestimmte Priorität spezifiziert werden muss. Für die Simulation kann entweder ein Konfidenzintervall oder die maximale Simulationsdauer angegeben werden. Als Ergebnis liefert die Simulation die minimale, mittlere und maximale Gesamtantwortzeit für jedes Szenario deren Verteilung auf die Hardware-Ressource. Zusätzlich wird die

Zahl der fertig gestellten Anfragen und die Varianz der Antwortzeiten angegeben. Die Durchführung der Simulation dauert in den meisten Fällen nur eine kurze Zeit.

Für die frühe Performance-Analyse ist somit eine hinreichende Auswertung der Modelle möglich. Weitere Modellierungen, wie z.B. unterschiedliche Ressourcentypen oder verschiedene Warteschlangendisziplinen können hingegen mit dem SPE-ED-Werkzeug nicht ausgewertet werden.

Das Capacity Planning verwendet ausschließlich Warteschlangenmodelle zur Performance-Modellierung, daher kommen hier die gängigen Analysemethoden wie die Mean-Value-Analysis zum Einsatz. Dazu nehmen die bei diesem Verfahren verwendeten Spreadsheets Eingabeparameter für Warteschlangennetzwerke wie Bearbeitungsdauer von Ressourcen und Benutzerankunftsrate auf. Die Antwortzeiten, Auslastungen und Warteschlangenlängen werden dann per Knopfdruck augenblicklich berechnet. Für Warteschlangenmodelle existieren eine Reihe von Formeln mit denen z.B. bei mehreren Benutzerklassen, lastabhängig-performanten Servern oder blockierenden Ressourcen Näherungslösungen erstellt werden können. Das theoretische Fundament der Auswertung von Warteschlangenmodellen ist weit ausgearbeitet und kann eine große Menge von Situationen aus der Praxis nachbilden.

Das umlPSI-Verfahren basiert auf einer Simulation zur Auswertung des Performance-Modells. Vorteil von Simulationen ist, dass weniger einschränkende Voraussetzungen für die auswertbaren Modelle gemacht werden müssen. So sind bei Warteschlangenmodellen nur wenige Arten von Wahrscheinlichkeitsverteilungen wie z.B. die Exponentialverteilung für die Benutzerankunftsrate und die Berechnungszeiten auf Server erlaubt. Auch bezüglich von Warteschlangendisziplinen sind nur wenige Formen wie z.B. "First Come, First Serve" auswertbar. Bei Simulationen existieren solche Einschränkungen nicht. Die Modelle können sehr flexibel sein und einen beliebigen Detaillierungsgrad besitzen. Nachteil sind die häufig schwierig zu interpretierenden Ergebnisse von Simulationen, die aus Zufallsvariablen bestehen. Simulationen können außerdem sehr zeitaufwändig sein und hohe Rechenanforderungen zur Ausführung haben. Tatsächlich dauerte die Simulation bei den umlPSI-Testläufen teilweise mehrere Stunden. Vorteilhaft bei diesem Verfahren ist jedoch, dass die Simulation automatisch aus annotierten UML-Diagrammen erstellt und ausgeführt werden kann.

6.4.5. Prozessintegration

Zum SPE gehört ein schrittweises Vorgehensmodell, das genau den Prozess der Performance-Analyse vorgibt. Ein spezielles Kapitel in [Smi02] widmet sich der Integration des Verfahrens in bekannte Vorgehensmodelle im Software Engineering. Dabei werden das Wasserfall-Modell, das Spiral-Modell und der Unified Process betrachtet.

Beim Capacity Planning ist der Prozess der Performance-Analyse nicht so straff organisiert wie beim SPE. Die einzelnen Techniken und Methoden des Verfahrens müssen jeweils an Einzelfälle angepasst werden. Für jedes untersuchte Modell müssen individuell die Eingabeparameter für die Warteschlangenmodelle bestimmt werden. In [MAD04] werden eine Reihe von Einzelfallbetrachtungen wie Datenbank-Server, Web-Applikationen oder E-Business Services vorgestellt. Die Software-Entwicklung und deren Vorgehensmodelle werden bei diesem Verfahren allerdings nicht berücksichtigt.

Die Performance-Analyse mit dem umlPSI-Werkzeug kann ohne großen Aufwand in die objektorientierte Entwicklung eingebettet werden. Liegen Entwürfe bereits in UML vor, können

diese mit Performance-Indizes ausgestattet und anschließend simuliert werden.

6.5. Daten der Verfahren

	SPE	CP	umlPSI
Autor	Smith/Williams	Menasce/Almeida	Mazolla/Balsamo
Jahr	2002	2004	2004
Literatur	[Smi90, Smi02]	[MA02, MAD04]	[BGM03, Mar04]
Ziel	Bewertung von Entwurfsalternativen	Kapazitätsplanung	Analyse von UML-Diagrammen
Untersuchungsobjekt	frühe Designdokumente	(halb-)fertige Systeme	UML-Diagramme
Ausgangsbasis	UML: Anwendungsfall-, Sequenz-, Verteilungsdiagramme	Workload-Traces, Benutzerankunfts-raten, Service Demands	UML: Anwendungsfall-, Aktivitäts-, Verteilungsdiagramme
Softwaremodell	Ausführungsgraphen	-	Aktivitätsdiagramme
Systemmodell	Warteschlangenmodelle	Warteschlangenmodelle	Anwendungsfall-, Verteilungsdiagramme
Auswertungsmethode	Analyse/Simulation	Analyse	Simulation
<i>Ergebnisse</i>			
Antwortzeit	j	j	j
Durchsatz	j	j	j
Auslastung	j	j	j
Warteschlangenlänge	j	j	n
Feedback der Ergebnisse	nein	nicht möglich	ja
Prozessmodell	detailliert	lose	einfach
<i>Anwendungsbeispiele</i>			
OO-Systeme	j	n	j
Web-Applikationen	j	j	j
Verteilte Systeme	j	j	j
Embedded Systems	j	n	n
Werkzeug	SPE-ED	Excel-Tabellen	umlPSI

Tabelle 6.7.: Daten der Verfahren

6.6. Bewertung der Verfahren

	SPE	CP	umlPSI
Abweichung Eingabedaten verschiedener Teilnehmer (M1)	niedrig/mittel	niedrig	hoch
Abweichung Eingabedaten Ausgabedaten (M2)	-	-	niedrig
Abweichung Vorhersage Messung (M3)	niedrig/mittel	niedrig	hoch
Richtige Entwurfsentscheidungen (M4)	75%	83,3%	62,5%
Lernaufwand Übung (M5)	2 Std.	2 Std.	2 Std.
Modellierungsaufwand Experiment (M6)	2 Std.	2 Std.	2 Std.
Eigenschaften (M8)	(Skala: ++,+,0,-,-)		
<i>Notation</i>			
Mächtigkeit	+	o	+
Strukturierung	+	o	+
Performance-Indizes	+	n.v.	+
<i>Werkzeug</i>			
Ergonomie	o	o	-
Robustheit	o	o	-
Stabilität	+	++	-
Präsentation	+	-	+
Praxistauglichkeit	+	+	--
Flexibilität	+	o	+
Auswertung	+	+	+
Prozessintegration	++	o	+

Tabelle 6.8.: Bewertung der Verfahren

7. Schlussfolgerungen und Ausblick

Das folgende Kapitel zieht Schlussfolgerungen dieser Arbeit, resümiert nochmals die Ergebnisse, stellt Anknüpfungspunkte für Folgearbeiten vor und schließt mit einem Ausblick in die Zukunft der Performance-Analyse für Software-Architekturen.

7.1. Zusammenfassung

Die im Rahmen dieser Diplomarbeit durchgeführte Fallstudie hat gezeigt, auf welche Weise die Performance-Analyse einer Software-Architekturen schon zu einem frühen Entwicklungszeitpunkt durchgeführt werden kann. Eine Gruppe von 24 Informatik-Studenten stellte mit drei Verfahren Vorhersagen über die Performance eines Webservers an. Jeder Teilnehmer schlug aus fünf Entwurfsalternativen eine Alternative für die Implementierung vor. Der Großteil (72%) favorisierte dabei Alternative 3, die Komprimierung von HTTP-Anfragen. Diese Entwurfsalternative wies später bei Messungen an der Implementierung tatsächlich die geringste Durchlaufzeit aller Alternativen auf.

Dabei wichen die vorhergesagten absoluten Durchlaufzeiten bei den Verfahren, die mit Schätzungen arbeiten (SPE, umlPSI), bei einigen Teilnehmern um mehrere Sekunden von den Messdaten ab. Mit der Abschätzung von Performance-Werten für bestimmte Aktionen und Ressourcen taten sich viele der Teilnehmer schwer. Auch die nachträgliche Überprüfung ihrer Ergebnisse auf Gültigkeit fand nur selten statt. Trotzdem gelang den Teilnehmern eine realistische Bewertung der Entwurfsalternativen, deren relative Unterschiede herausgestellt werden konnten. Der durch die verschiedenen Entwürfe prozentual erzielbare Performance-Gewinn konnte mit einer Abweichung von weniger als 15% in Relation zur Messung an der Implementierung vorhergesagt werden.

Die drei untersuchten Verfahren konnten den Studenten in jeweils zweistündigen Übungssitzungen und mit der Bearbeitung eines Übungszettels vermittelt werden. Der Aufwand für die Analyse des Webservers durch die Teilnehmer betrug ebenfalls zwei Stunden. Damit darf die These viele Praktiker, dass frühe Performance-Analysen nur unter großem Aufwand durchzuführen sind, zumindest angezweifelt werden.

Die Aufgabenstellung über die Untersuchung des Webservers wurde zusätzlich einer Kontrollgruppe von sieben Studenten vorgelegt, die die Verfahren nicht kannten. Dabei kam es zu einer leicht anderen Bewertung der Entwurfsalternativen als im Experiment. Die Mehrheit favorisierte Alternative 1b, die bei den Vorhersageverfahren und der Messung nur die zweitschnellste Alternative war. Dieses Ergebnis zeigt, dass die Aufgabenstellung mit den fünf Entwurfsalternativen für den Webserver bei einem intuitiven Vorgehen nicht automatisch zur Favorisierung der Alternative 3 führt. Somit haben die Performance-Analyseverfahren einen Effekt auf die Entwurfsentscheidung der Teilnehmer gehabt.

Eine Reihe von Eigenschaften der Verfahren, wie die Mächtigkeit der Notationen, die Ergonomie der Werkzeuge oder die Auswertungsmethoden, wurden näher erläutert. Es zeigte sich, dass die Verfahren formal recht weit ausgearbeitet sind, es aber noch an der Praxistauglichkeit und der Verfügbarkeit von effizienten Werkzeugen mangelt. Insbesondere die Integration der Analysen in gängige Entwicklungswerkzeuge ist noch nicht oder bisher nur unzureichend erfolgt. Die Entwickler müssen noch viel manuellen Aufwand in die Analysen stecken, um tatsächlich zu genauen Ergebnissen und realistischen Vorhersagen zu gelangen. Mit der Verfügbarkeit der UML und des UML Performance-Profiles gibt es jedoch Ansatzpunkte, die eine bessere Integration von Performance-Analysen in die objekt-orientierte Software-Entwicklung erwarten lassen.

Ein schwieriges Problem stellt weiterhin die Motivation von Entwicklern für frühe Performance-Analysen dar. Die "fix-it-later"-Einstellung war auch bei den Teilnehmern dieser Fallstudie spürbar, und viele von ihnen fühlten sich bei der Schätzung von Performance-Werten unwohl. Im Verlauf der Fallstudie konnten jedoch einige Erkenntnisse über die Verfahren gefunden werden.

Das SPE-Verfahren von Smith und Williams ist gut ausgelegt für frühe Performance-Analysen von Software-Architekturen. Es ist das Verfahren, das mit den wenigsten Eingaben bereits erste Performance-Vorhersagen treffen kann. Die Methode ist nach über 20-jähriger Forschung weit ausgearbeitet. Es existiert ein Werkzeug (SPE-ED), mit dem die in [Smi02] erläuterten Konzepte praktisch umgesetzt werden können. Dabei werden die zugrunde liegenden Warteschlangenmodelle gut vor dem Entwickler verkapselt, so dass sich die Komplexität der Modellierung reduziert. Im Buch von Smith und Williams finden sich viele Hinweise und Techniken zur frühen Abschätzung von Performance-Daten, außerdem ist eine Liste von allgemeinen Performance-Prinzipien, -Patterns und -Antipatterns enthalten, mit denen bewährte Entwurfsmuster dokumentiert werden. Auch über die Integration von SPE in Vorgehensmodelle aus der Software-Technik wurden Überlegungen angestellt.

Trotzdem ist bei diesem Verfahren noch immer ein Bruch zwischen Software-Modellierung und Performance-Modellierung bemerkbar. Die benutzten Ausführungsgraphen müssen manuell aus Software-Spezifikationen (z.B. im UML) erstellt werden, eine automatische Transformation bietet das SPE-ED-Werkzeug nicht. Auch eine Rückabbildung der Performance-Ergebnisse in die Software-Spezifikationen ist hier nicht vorgesehen. SPE-ED ist ein eigenständiges Werkzeug und nicht in andere CASE-Werkzeuge oder Entwicklungsumgebungen integrierbar. Es gibt auch keine Möglichkeit Laufzeitumgebungen wie in der Programmiersprache Java oder den Einfluss von Application Servern adäquat zu modellieren. Messungen an fertigen Teilen der Implementierung, die die Vorhersagen des Verfahrens verfeinern können, müssen nachträglich manuell in die Performance-Modelle integriert werden. Als Verbesserung für dieses Werkzeug wäre beispielsweise ein XMI-Import für UML-Diagramme sinnvoll, die automatisch in die erforderlichen Ausführungsgraphen transformiert werden könnten. Auch die bereits angedachten Erweiterungen wie das "Advanced System Model" mit neuen Modellierungsmöglichkeiten und der Export der Performance-Modelle in ein standardisierte XML-basiertes Austauschformat namens PMIF (Performance Model Interchange Format) [SL04] sind sehr sinnvoll. Zusätzlich sollte die graphische Benutzungsoberfläche überarbeitet werden.

Das CP-Verfahren ist eng verbunden mit der klassischen Warteschlangentheorie und eignet sich weniger gut für die frühe Performance-Analyse, sondern eher für die Untersuchung bereits bestehender Systeme. Nach über 30 Jahren Forschung in der Warteschlangenmodellierung existieren eine Reihe von Algorithmen und Formeln, um eine Vielzahl von realen Computersy-

stemen auszuwerten. Dabei können teilweise nur Näherungslösungen erstellt werden. Setzt man das Verfahren zur frühen Performance-Analyse ein, muss abgewogen werden, ob es sich lohnt die teilweise recht komplexen Modelle zu erstellen und sich Kenntnisse über die mathematischen Hintergründe für die Auswertung anzueignen. Die Eingabeparameter sind in den meisten Beispielen zu diesem Verfahren Messdaten und basieren nicht auf Schätzungen. Die Analyse findet bei diesem Verfahren fast ausschließlich auf der Hardware-Ebene statt. Eine Integration von Software-Modellen ist nicht vorgesehen, die UML beispielsweise wird in der zugehörigen Literatur [MA02, MAD04] überhaupt nicht erwähnt. Auch die Kopplung der verfügbaren Werkzeuge mit gängigen CASE-Werkzeuge ist daher bei diesem Verfahren nicht angedacht.

Einen interessanten Ansatz stellt das umlPSI-Verfahren dar: es benutzt UML-Diagramme aus denen automatisch eine Simulation des Performance-Verhaltens einer Architektur generiert wird. Dazu werden die Diagramme vorher mit dem standardisierten UML Performance Profile um Performance-Daten ergänzt. Es ist das erste Verfahren, dass auf diesem Standard aufbaut und eine Simulation verwendet. Nach der Simulation können die Ergebnisse in die Diagramme zurücktransferiert werden, somit ist es das einzige hier untersuchte Verfahren, das einen Feedback-Mechanismus implementiert. Eine solche Ergebnisaufbereitung sollte nicht unterschätzt werden, denn einige Teilnehmer der Fallstudie äußerten, dass die große Menge an Zahlen, die vom SPE- und vom CP-Verfahren produziert wird, für sie wenig Aussagekraft besäße. Das umlPSI-Verfahren ist der realen Software-Entwicklung von den untersuchten drei Verfahren an nächsten und daher wohl am bestens dazu geeignet Entwickler von der Machbarkeit früher Performance-Analysen zu überzeugen. Diese müssen lediglich das UML Performance Profile lernen, Kenntnisse über die Auswertungsverfahren sind hier nicht notwendig.

Auch wenn das Konzept dieses Verfahrens sehr sinnvoll erscheint, so mangelt es noch an der Umsetzung. Das zur Demonstration der Methode rudimentär implementierte Werkzeug umlPSI ist lediglich ein Prototyp und arbeitet noch nicht befriedigend. Nicht nur die Installation gestaltete sich äußerst schwierig, auch die Ausführungsdauer war oft sehr hoch. Zudem zeigte sich das Werkzeug nicht besonders robust gegenüber Fehleingaben der Studenten. Momentan muss das Werkzeug noch außerhalb der Modellierungsumgebung (in diesem Fall Poseidon) gestartet werden und importiert XMI-Dateien. Die Umsetzung als Plugin für weit verbreitete Modellierungswerkzeuge wie Together oder Rose erscheint hier sinnvoller als die jetzige Variante. Auch ist eine Reihe von Erweiterungen denkbar. Weitere Diagrammtypen (z.B. Sequenz-, Zustands- oder Komponentendiagramme) könnten unterstützt werden. Die Ergebnisse der Simulation könnten noch ausführlicher gefasst werden, momentan wird nur die minimale und maximale Dauer der Simulation angegeben. Denkbar wäre auch die Unterstützung weiterer UML Profiles z.B. für Zuverlässigkeit oder Sicherheit und die Integration des Werkzeugs in ein Framework für die Analyse anderer Quality-of-Service Eigenschaften. Interessant ist die Idee, das Werkzeug in einer integrierten Entwicklungsumgebung, die neben UML-Diagrammen auch den zugehörigen Programmcode bearbeiten kann, mit einem Code-Profiler [rgS04] zu verbinden. Mit dem Profiler könnten die Ausführungszeiten für einzelne, bereits fertig gestellte Codefragmente in Testläufen gemessen werden und in Tagged-Values der UML-Diagramme zurückgeliefert werden. Auf diese Weise könnten die Schätzungen in den Diagrammen mit den genaueren Messungen aktualisiert werden. Die Modelle könnten dann in späteren Entwicklungsphasen automatisch verbesserte Vorhersagen der Performance des Gesamtsystems liefern.

7.2. Anknüpfungspunkte für Folgearbeiten

Ein umfassender Vergleich aller Techniken und Verfahren zur frühen Performance-Analyse von Software-Architekturen war im Zeitrahmen einer Diplomarbeit nicht machbar. Insbesondere Verfahren, die auf Petri-Netzen beruhen oder Prozessalgebren als Performance-Modell verwenden, konnten hier nicht untersucht werden. Dabei wäre interessant gewesen, wie stark sich die bekannten Probleme dieser Notationen wie z.B. die Explosion des Zustandsraums schon bei einfachen Architekturen bemerkbar machen. Auch könnte man vergleichen, ob sich allein durch die benutzte Notation Abweichungen in den Vorhersagen ergeben.

Zur Untersuchung des Zeitaufwands und der tatsächlichen Kosten für frühe Performance-Analysen müssen Feldstudien an realen Projekten in der Wirtschaft durchgeführt werden. Dabei wäre es interessant zu sehen, ob sich die Verfahren für wesentlich größere Systeme als den hier untersuchten Webserver eignen. Möglich wäre es auch, solche Studien mit den Entwurfsdokumenten von studentischen Projektgruppen durchzuführen.

Um die Ergebnisse der Fallstudie zu verbessern, müssten wesentlich mehr Teilnehmer für ein Experiment hinzugezogen werden, als das hier möglich war. Auch andere Gruppen außer Studenten, z.B. Praktiker aus der Software-Branche oder Performance-Experten müssten an einer solchen Studie teilnehmen.

7.3. Zukunft des Performance Engineering

Das Ziel vieler aktueller Bemühungen der Forschung zum Performance Engineering ist eine enge Integration von Software-Entwicklung und Performance-Analysen. Mit der Verwendung von Standard-Notationen wie der UML, und dem Versuch daraus Performance-Modelle automatisch zu generieren, versucht man eine Brücke zwischen den beiden Disziplinen zu schlagen. Damit sollen Performance-Analysen auch für nicht speziell geschulte Entwickler ohne großen manuellen Aufwand möglich werden. Bevor allerdings wirklich brauchbare Software-Werkzeuge vorliegen, die die Performance-Analyse so weit wie möglich verkapseln und die Komplexität der dahinter stehenden Modelle und Algorithmen verbergen, muss noch einiges Arbeit geleistet werden.

Mit der Reifung des Performance Engineering werden neue Performance-Patterns und -Antipatterns dokumentiert werden. Denkbar wäre es Methoden zu entwickeln, die bekannte Antipatterns in bestehenden Architekturen automatisch erkennen und passende Performance-Patterns oder andere Verbesserungen vorschlagen.

Angedacht ist weiterhin eine bessere Verbindung der Methoden des Software Performance Engineering mit dem Capacity Planning [Smi01]. Beispielsweise könnten die komplexen Möglichkeiten zur Modellierung von Warteschlangenmodellen, wie sie beim CP vorkommen, auch im SPE zur Erstellung genauerer Vorhersagen genutzt werden. Gleichsam könnte das Capacity Planning schon in frühen Entwicklungsphasen von Computersystemen durchgeführt werden und um Konstrukte zur Modellierung von Software ergänzt werden.

Komponentenbasierte Software-Systeme [SGM02] bieten neue Möglichkeiten zur Performance-Analyse. Die Spezifikation von Komponenten kann von ihren Entwicklern mit Performance-Eigenschaften ergänzt werden. Abhängig von Hardware-Umgebung und Nutzungsprofil können dann Entwickler, die die Komponente einsetzen, Rückschlüsse auf die zu erwartende Performance ziehen. Wenn alle Teilkomponenten eines Gesamtsystems mit Performance-Charakteristiken

attribuiert sind, kann auch die Performance des Gesamtsystems vorhergesagt werden.

In Bezug auf frühe Performance-Analysen muss sich sicher noch die Mentalität vieler Entwickler wandeln und eine bessere Vermittlung durch Lehrinrichtungen wie Universitäten erfolgen. Die frühe Bewertung von Leistungskriterien sollte in Zukunft nicht mehr als lästige Zusatzaktivität gesehen werden, sondern als integraler und notwendiger Bestandteil der Software-Entwicklung. Auch andere nicht-funktionale Eigenschaften von Softwaresystemen wie Zuverlässigkeit, Verfügbarkeit, Sicherheit, Fehlertoleranz oder Ergonomie müssen in der Software-Entwicklung proaktiv berücksichtigt werden.

Literaturverzeichnis

- [AABI00] F. Andolfi, F. Aquilani, S. Balsamo, and P. Inverardi. Deriving performance models of software architectures from message sequence charts. In *ACM Proceedings of the Second International Workshop on Software and Performance*, pages 47–57, 2000.
- [ABI01] F. Aquilani, S. Balsamo, and P. Inverardi. Performance analysis at the software architecture design level. *Performance Evaluation*, 45(2):205–221, 2001.
- [AS99] L. B. Arief and N. A. Speirs. Automatic generation of distributed system simulations from uml. In *Proceedings of ESM '99, 13th European Simulation Multiconference*, 1999.
- [BBS02] S. Balsamo, M. Bernardo, and M. Simeoni. Combining stochastic process algebras and queueing networks for software architecture analysis. In *ACM Proceedings of the International Workshop on Software and Performance*, 2002.
- [BCD00] M. Bernardo, P. Ciancarini, and L. Donatiello. Aempa: A process algebraic description language for the performance analysis of software architectures. In *ACM Proceedings of the International Workshop on Software and Performance*, pages 1–11, 2000.
- [BCMP75] F. Baskett, K.M. Chandy, R.R. Muntz, and F.G. Palacios. Open, closed and mixed networks of queues with different classes of customers. *J.ACM* 22, pages 248–260, 1975.
- [BCR94] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The goal question metric approach. *Encyclopedia of Software Engineering - 2 Volume Set*, pages 528–532, 1994.
- [BDM02] S. Bernardi, S. Donatelli, and J. Merseguer. From uml sequence diagrams and state-charts to analysable petri net models. In *Proceedings of WOSP2002*, 2002.
- [BG98] M. Bernardo and R. Gorrieri. A tutorial on empa: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, 202:1–54, 1998.
- [BGM03] S. Balsamo, M. Grosso, and M. Marzolla. Towards simulation-based performance modeling of uml specifications. Technical report, Dipartimento di Informatica, Università Ca' Foscari di Venezia, 2003.

- [BINN99] J. Banks, J.S. Carson II, B.L. Nelson, and D.M. Nicol. *Discrete-Event System Simulation*. Prentice Hall, 1999.
- [BM03] S. Balsamo and M. Marzolla. A simulation-based approach to software performance modeling. Technical report, Università Ca' Foscari di Venezia, 2003.
- [BMDI04] S. Balsamo, M. Marzolla, A. DiMarco, and P. Inverardi. Experimenting different software architectures performance techniques: A case study. In *Proceedings of the fourth international workshop on Software and performance*, pages 115–119. ACM Press, 2004.
- [BMIS04] Simonetta Balsamo, Antiniscia Di Marco, Paola Inverardi, and Marta Simeoni. Model-based performance prediction in software development: A survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, May 2004.
- [Buz71] J.P. Buzen. *Queueing Network Models of Multiprogramming*. PhD thesis, Harvard University, 1971.
- [CDI01] V. Cortellessa, A. D'Ambrogio, and G. Iazeolla. Automatic derivation of software performance models from case documents. *Performance Evaluation*, 45:81–105, 2001.
- [CLGL04] S. Chan, Y. Liu, I. Gorton, and A. Liu. Performance prediction of component-based applications. *Journal of System Software Automated Component-Based Software Engineering Special Issue*, 2004.
- [CM02] V. Cortellessa and R. Mirandola. Prima-uml: a performance validation incremental methodology on early uml diagrams. In *Proceedings of WOSP*, 2002.
- [Cor04] Standard Performance Evaluation Corporation. Spec benchmark. <http://www.specbench.org/>, 2004.
- [CS03] Volker Claus and Andreas Schwill. *Duden Informatik*. Bibliographisches Institut, Mannheim, 2003.
- [DS00] Evgeni Dimitrov and Andreas Schmietendorf. Uml-basiertes performance engineering. In *Tagungsband zum 1. Workshop Performance Engineering in der Softwareentwicklung*, 2000.
- [Dug04] Robert F. Dugan. Performance lies my professor told me: The case for teaching software performance engineering to undergraduates. In *Proceedings of the Fourth International Workshop on Software and Performance*, 2004.
- [Dum02] Reiner Dumke, editor. *Performance Engineering: State of the Art and Current Trends*. Number 2047 in Lecture Notes in Computer ScienceS. Springer Verlag, 2002.
- [Eng04] Tigris.org Open Source Software Engineering. Argouml. <http://argouml.tigris.org/>, 2004.

- [Feh99] Martin Fehrenbach. Vergleich verschiedener spezifikationstechniken und -werkzeuge am beispiel. Master's thesis, Universität Kaiserslautern, 1999.
- [Fow03] Martin Fowler. *UML Distilled*. Addison-Wesley, 2003.
- [Gen04] Gentleware. Poseidon for uml. <http://www.gentleware.com/>, 2004.
- [GH94] S. Gilmore and J. Hillston. The pepa workbench: A tool to support a process algebra-based approach to performance modelling. In *Proceedings of the Seventh International Conference for Modelling Techniques and Tools for Performance Evaluation*, pages 353–368, 1994.
- [GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Systems*. Addison-Wesley, 1995.
- [GL03] Ian Gorton and A. Liu. Performance evaluation of alternative component architectures for enterprise javabeen applications. *IEEE Internet Computing*, 7(3):18–23, 2003.
- [Gla98] Robert L. Glass. *Software Runaways. Lessons learned from Massive Software Project Failures*. Prentice Hall, 1998.
- [GM00] H. Gomaa and D. A. Menasce. Design and performance modeling of component interconnection patterns for distributed systems. In *ACM Proceedings of the International Workshop on Software and Performance*, 2000.
- [GM01] H. Gomaa and D. A. Menasce. Performance engineering of component-based distributed software systems. In *Performance Engineering*, volume 2047, pages 40–55. Springer Verlag, 2001.
- [GM02] V. Grassi and R. Mirandola. Primamob-uml: A methodology for performance analysis of mobile software architectures. In *ACM Proceedings of the International Workshop on Software and Performance*, pages 262–274, 2002.
- [GP02] G. Gu and D.C. Petriu. Xslt transformations from uml models to lqn performance models. In *ACM Proceedings of the International Workshop on Software and Performance*, 2002.
- [HH95] P.G. Harrison and J. Hillston. Exploiting quasi-reversible structures in markovian process algebra models. *Computer Journal*, 38(7):510–520, 1995.
- [HHK02] H. Hermanns, U. Herzog, and J.P. Katoen. Process algebra for performance evaluation. *Theoretical Computer Science*, 274(1-2):43–87, 2002.
- [Hil93] J. Hillston. Pepa - performance enhanced process algebra. Technical report, Dept. of Computer Science University of Edinburgh, 1993.
- [Hoe00] F. Hoebe. Using uml models for performance calculations. In *Proceedings of WOSP*, 2000.

- [HRP03] A. Hennig, D. Revill, and M. Ponitsch. From uml to performance measures - simulative performance predictions of it-systems using the jboss application server with omnet++. In *Proc. of ESM '03, the 17th European Simulation Multiconference*, 2003.
- [HUM00] U. Herzog, U.Klehmet, and V. Mertsiotakis. Compositional performance modelling with the tiptool. *Performance Evaluation*, 39(1-4):5–35, 2000.
- [Inc04] Web Performance Inc. Web performance trainer. <http://www.webperformanceinc.com>, 2004.
- [Kan92] K. Kant. *Introduction to Computer System Performance Evaluation*. McGraw-Hill, 1992.
- [Käh01] P. Kähkipuro. Uml-based performance modeling framework for component-based distributed systems. *Lecture Notes in Computer Science 2047*, 2001.
- [Kil02] Patrick Killelea. *Web Performance Tuning*. O'Reilly, 2nd edition, 2002.
- [Knu73] D.E. Knuth. *The Art of Computer Programming. Vol. 3: Sorting and Searching*. Addison-Wesley, 1973.
- [KP00] P.J.B. King and R.J. Pooley. Derivation of petri net performance models from uml specifications of communications software. *Lecture Notes in Computer Sciece 2047*, pages 262–276, 2000.
- [Kra03] Benedikt Kratz. Empirical research on the relationship between functionality and interfaces of software components. Master's thesis, Universiteit van Tilburg, 2003.
- [Lib03] Jesse Liberty. *Programming #*. O'Reilly, 3rd edition, June 2003.
- [Lit61] J.C. Little. A proof of the queueing formula $l = \lambda w$. *Operations Research*, 9:383–387, 1961.
- [LR80] Steve Lavenburg and Martin Reiser. Mean value analysis of closed multichain queueing networks. *Journal of the ACM*, 27(2), April 1980.
- [LTK⁺02] C. Lindemann, A. Thümmler, A. Klemm, M. Lohmann, and O.P. Waldhorst. Performance analysis of time-enhanced uml diagrams based on stochastic processes. In *ACM Proceedings of the International Workshop on Software an Performance*, pages 25–34, 2002.
- [LZGS84] E. D. Lazowska, J. Zahorjan, G. Scott Graham, and K.C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, 1984.
- [MA98] Daniel A. Menasce and Virgilio A. F. Almeida. *Capacity Planning for Web Performance: metric, models, and methods*. Prentice Hall, 1998.

- [MA00] Daniel A. Menasce and Virgilio A. F. Almeida. *Scaling for E-Business: technologies, models, performance, and capacity planning*. Prentice Hall, 2000.
- [MA02] Daniel A. Menasce and Virgilio A.F. Almeida. *Capacity Planning for Web Services*. Prentice-Hall, 2002.
- [MAD94] Daniel A. Menasce, Virgilio A. F. Almeida, and Lawrence W. Dowdy. *Capacity Planning and Performance Modeling: from mainframes to client-server systems*. Prentice Hall, 1994.
- [MAD04] Daniel A. Menasce, Virgilio A.F. Almeida, and Lawrence W. Dowdy. *Performance by Design*. Prentice Hall, 2004.
- [Mar90] M. Ajmone Marsan. Stochastic petri nets: An elementary introduction. In *Advances in Petri Nets*, volume 424 of *Lecture Notes in Computer Science*, pages 1–29. Springer Verlag, 1990.
- [Mar04] Moreno Marzolla. *Simulation-Based Performance Modeling of UML Software Architectures*. PhD thesis, Universit‘a Ca Foscari di Venezia, 2004.
- [MBG84] M. Ajmone Marsan, G. Balbo, and G.Conte. A class of generalized stochastic petri nets for the performance evaluation of mulitprocessor systems. *ACM Transactions on Computer Systems*, 1984.
- [Men02] Daniel A. Menasce. Software, performance, or engineering? In *Proc. ACM 2002 Workshop on Software and Performance*, 2002.
- [MG98] D. A. Menasce and H. Gomaa. On a language based method for software performance evaluation of client/server systems. In *ACM Proceedings of the International Workshop on Software and Performance*, 1998.
- [MG00] D. A. Menasce and H. Gomaa. A method for design and performance modelling of client/server systems. *IEEE Transaction on Software Engineering*, 26(11):1066–1085, November 2000.
- [MLH⁺00] M. De Miguel, T. Lamobolais, M. Hannouz, S. Betge-Brezetz, and S. Piekarec. Uml extensions for the specifications and evaluation of latency constraints in architectural models. In *Proceedings of WOSP*, 2000.
- [Mol82] M.K. Molloy. Performance analysis using stochastic petri nets. *IEEE Transactions on Computers*, 1982.
- [Moo65] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.
- [MR04] Kjeld H. Mortensen and Heiko Rölke. Petri net tool database. <http://www.daimi.au.dk/PetriNets>, 2004.
- [New03] Heise Newsticker. Virtuelle arbeitsagentur erntet heftige kritik. <http://www.heise.de/newsticker/meldung/42566>, Dezember 2003.

- [OMG02] Object Management Group OMG. Xml metadata interchange (xmi) specification. <http://www.omg.org/technology/documents/formal/xmi.htm>, Jan. 2002.
- [OMG03a] Object Management Group OMG. Uml profile for schedulability, performance and time. <http://www.omg.org/cgi-bin/doc?formal/2003-09-01>, 2003.
- [OMG03b] Object Management Group OMG. Unified modelling language (uml), version 1.5. <http://www.omg.org/cgi-bin/doc?formal/03-03-01>, 2003.
- [Pet62] C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für Instrumentelle Mathematik Bonn, 1962.
- [Poo99] R. Pooley. Using uml to derive stochastic process algebra models. In *Proceedings of the 25th UK Performance Engineering Workshop*, pages 23–34, 1999.
- [Pre00] Lutz Prechelt. An empirical comparison of seven programming languages. *IEEE Computer*, 33(10):23–29, October 2000.
- [Pre01] Lutz Prechelt. *Kontrollierte Experimente in der Softwaretechnik*. Springer Verlag, 2001.
- [PS02] D.C. Petriu and H. Shen. Applying the uml performance profile: graph grammar-based derivation of lqn models from uml specifications. In *Proc. Seventh International Conference for Modelling Techniques and Tools for Performance Evaluation*, pages 159–177, 2002.
- [PW99] D.C. Petriu and X. Wang. From uml descriptions of high-level software architectures to lqn performance models. In *Proc. Applications of Graph Transformations with Industrial Relevance Workshop (AGTIVE’99)*, pages 47–62, 1999.
- [PW02a] D.C. Petriu and C.M. Woodside. Software performance models from system scenarios in use case maps. In *Proceedings of the 12th International Conference for Modelling Tools and Techniques for Computer and Comm. System Performance Evaluation*, pages 141–158, 2002.
- [PW02b] Lutz Priese and Harro Wimmel. *Theoretische Informatik Petri-Netze*. Springer Verlag, 2002.
- [RB96] R.S. Casselman R.J.A. Buhr. *Use CASE Maps for Object-Oriented Systems*. Prentice Hall, 1996.
- [Rec99] W3C Recommendation. Xsl transformations (xslt). <http://www.w3.org/TR/xslt>, November 1999.
- [rgS04] red-gate Software. Ants-profiler. http://www.red-gate.com/code_profiling.htm, 2004.
- [RKFPS04] Ralf H. Reussner, Juliana Küster-Filipe, Iman H. Poernomo, and Sandeep Shukla. Report on the workshop on formal foundations of embedded software and component-based software architectures (fesca). *FESCA 2004 Workshop Report*, 2004.

- [RS95] J.A. Rolia and K.C. Sevcik. The method of layers. *IEEE Transactions on Software Engineering*, 21(8):682–688, 1995.
- [SG96] Mary Shaw and David Garland. *Software Architecture. Perspectives on an Emerging Discipline*. Prentice Hall, 1996.
- [SGM02] Clemens Szyperski, Dominik Gruntz, and Stephan Murer. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 2nd edition, 2002.
- [SL04] Connie U. Smith and Catalina M. Llado. Performance model interchange format (pmif 2.0): Xml definition and implementation. Technical report, Performance Engineering Services, New Mexico; Universitat Illes Balears, Departament de Matemàtiques I Informàtica, Palma de Mallorca, Spain, February 2004.
- [Smi80] C.U. Smith. *The Prediction and Evaluation of the Performance of Software from Extended Design Specifications*. PhD thesis, University of Texas, 1980.
- [Smi81] C.U. Smith. Increasing information systems productivity by software performance engineering. In Donald R. Deese, Robert J. Bishop, Jeffrey M. Mohr, and H. Pat Artis, editors, *Int. CMG Conference*, pages 5–14. Computer Measurement Group, 1981.
- [Smi90] C.U. Smith. *Performance Engineering of Software Systems*. Addison-Wesley, 1990.
- [Smi01] Connie U. Smith. Origins of software performance engineering: Highlights and outstanding problems. In *Performance Engineering: State of the Art and Current Trends*. Springer Verlag 2, 2001.
- [Smi02] Connie U. Smith. *Performance Solutions: A Practical Guide To Creating Responsive, Scalable Software*. Addison-Wesley, 2002.
- [Sof04] Mesquite Software. Csim simulation software. <http://www.mesquite.com/>, 2004.
- [Spe04] Special Interest Group on Software Engineering. *Proceedings of the Fourth International Workshop on Software and Performance*, volume 29 of *Software Engineering Notes*, Redwood Shores, California, USA, 2004.
- [SS01] Andreas Schmietendorf and Andre Scholz. Aspects of performance engineering - an overview. In *Performance Engineering: State of the Art and Current Trends*. Springer Verlag, 2001.
- [Sta99] ITU Telecommunication Standardization. Message sequence charts, itu-t recommendation, 1999.
- [SW93] Connie U. Smith and Lloyd G. Williams. Software performance engineering: A case study including performance comparison with design alternatives. *IEEE Transactions On Software Engineering*, 19(7):720–741, July 1993.

- [SW97] Connie U. Smith and L.G. Williams. Performance engineering evaluation of object-oriented systems with speed. In *Lecture Notes Of Computer Science*, volume 1245, pages 135–153. Springer Verlag, 1997.
- [Tei04] Yvette Teiken. Webserver performance vorhersage. Individuelles projekt, Carl-von-Ossietzky Universität Oldenburg, August 2004.
- [vKKR⁺98] Antje von Knethen, Erik Kamsties, Ralf Reussner, Christian Bunse, and Bin Shen. A comparative case study with industrial requirements engineering methods. In *Proceedings of the 11th International Conference on Software Engineering an its Applications*, 1998.
- [WHSB01] C.M. Woodside, C. Hrischuk, B. Selic, and S. Brayarov. Automated performance modeling of software generated by a design environment. *Performance Evaluation*, 45:107–123, 2001.
- [WLA⁺01] G. Waters, P. Linington, D. Akehurst, P. Utton, and G. Marting. Permabase: predicting the performance of distributed systems at the design stage. *IEEE Transactions on Software*, 2001.
- [WNPM95] C.M. Woodside, J.E. Neilson, D.C. Petriu, and S. Majumdar. The stochastic rendezvous network model for performance of synchronous client-server-like distributed software. *IEEE Transactions on Computers*, 44(1):20–34, January 1995.
- [Woo88] C.M. Woodside. Throughput calculation for basic stochastic rendezvous networks. *Performance Evaluation*, 9:143–160, 1988.
- [Yin94] Robert K. Yin. *Case Study Research: Design and Methods*. Sage Publications, 1994.

Abbildungsverzeichnis

1.1. Aufbau der Ausarbeitung	5
2.1. Performance Engineering im Software-Lebenszyklus	8
2.2. Teilbereiche des Performance Engineering [SS01]	9
2.3. Markow-Kette eines einfachen Rechners [MAD04, p.268]	11
2.4. Einfaches Warteschlangenmodell [MAD04, p.292]	12
2.5. Offenes und geschlossenes Warteschlangennetzwerk	13
2.6. Geschichtetes Warteschlangenmodell [PW99]	14
2.7. Petri-Netz [CS03]	16
2.8. Generischer Prozess der Performance-Analyse	18
3.1. Goal, Question, Metric Methode [BCR94]	23
4.1. SPE-Prozess [Smi02]	37
4.2. Simulation von UML-Modellen [Mar04]	40
4.3. Vorgehensmodell beim Capacity Planning [MAD04]	41
5.1. Aufbau der Fallstudie	44
5.2. Informationen über die Versuchsteilnehmer	46
5.3. SPE-Ergebnisse	49
5.4. CP-Ergebnisse	50
5.5. Komponentendiagramm des Webserver	53
5.6. Eingabeparameter SPE	54
5.7. Eingabeparameter Capacity Planning	55
5.8. Eingabeparameter umlPSI	56
6.1. Vorhergesagte Durchlaufzeiten beim SPE-Verfahren	68
6.2. Vohergesagte Durchlaufzeiten beim CP-Verfahren (dynamisch)	70
6.3. Vohergesagte Durchlaufzeiten beim CP-Verfahren (statisch)	70
6.4. Vohergesagte Durchlaufzeiten beim umlPSI-Verfahren (dynamisch)	72
6.5. Vohergesagte Durchlaufzeiten beim umlPSI-Verfahren (statisch)	73
6.6. Durchlaufzeiten des Webserver (Messung)	74
6.7. Auslastungen des Webserver (Messung)	76
6.8. Durchlaufzeiten bei den verschiedenen Verfahren (statisch)	77
6.9. Durchlaufzeiten bei den verschiedenen Verfahren (dynamisch)	77
6.10. Prozentuale Durchlaufzeiten (statisch)	79

6.11. Prozentuale Durchlaufzeiten (dynamisch)	79
6.12. Empfehlungen der Teilnehmer für eine Entwurfsalternative	80
6.13. Bewertung der Entwurfsalternativen nach Durchlaufzeit	81
6.14. Zeitaufwand in Übung und Experiment	86
H.1. Boxplot der Durchlaufzeiten beim SPE-Verfahren	lxxxi
H.2. Vorhergesagte Auslastungen beim SPE-Verfahren	lxxxii
H.3. Boxplot der Durchlaufzeiten beim CP-Verfahren (statische Anfragen)	lxxxii
H.4. Boxplot der Durchlaufzeiten beim CP-Verfahren (dynamische Anfragen)	lxxxiii
H.5. Vorhergesagte Auslastungen beim CP-Verfahren	lxxxiii
H.6. Boxplot der Durchlaufzeiten beim umlPSI-Verfahren (statische Anfragen)	lxxxiv
H.7. Boxplot der Durchlaufzeiten beim umlPSI-Verfahren (dynamische Anfragen)	lxxxv
H.8. Vorhergesagte Auslastungen beim umlPSI-Verfahren	lxxxv

Tabellenverzeichnis

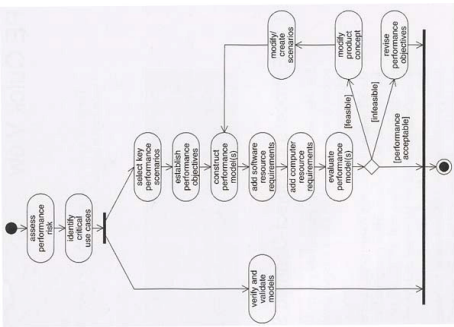
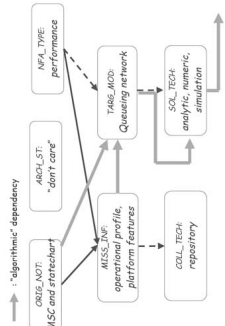
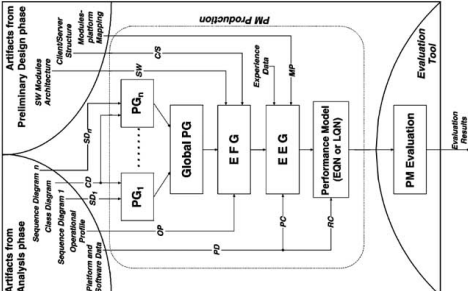
3.1. Validität der Vorhersagen	25
3.2. Überblick GQM	27
5.1. Punkte der Teilnehmer des Experiments in der Übung	52
6.1. Vorhergesagte Ressourcenauslastung beim SPE-Verfahren (Mittelwert)	69
6.2. Vohergesagte Ressourcenauslastung beim CP-Verfahren (Mittelwert)	71
6.3. Vohergesagte Ressourcenauslastung beim umlPSI-Verfahren (Mittelwert)	73
6.4. Mittelwerte der Abweichung über alle Varianten	78
6.5. Mittelwerte der prozentualen Abweichung über alle Varianten	78
6.6. Validität der Vorhersagen	82
6.7. Daten der Verfahren	96
6.8. Bewertung der Verfahren	97

A. Vergleichstabelle

Performance-Analyseverfahren

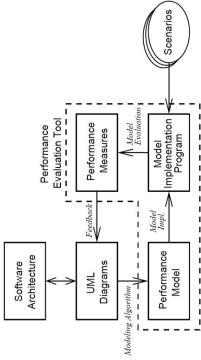
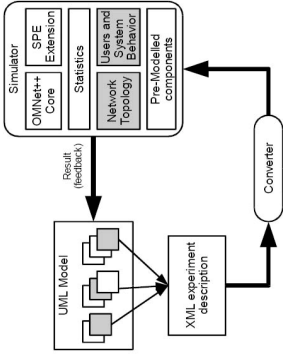
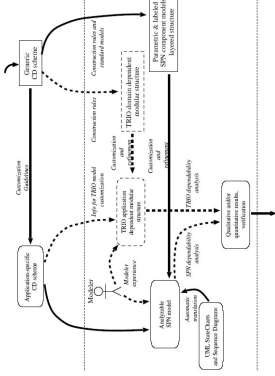
Die folgende Tabelle enthält zusammengefasst die Daten der in Kapitel 4 vorgestellten Performance-Analyseverfahren für Software-Architekturen. Eine Abbildung des jeweiligen Vorgehensmodells wurde integriert, sofern diese vorlag. Folgende Abkürzungen werden verwendet:

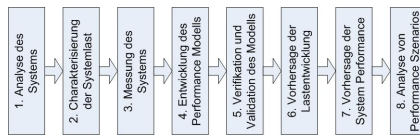
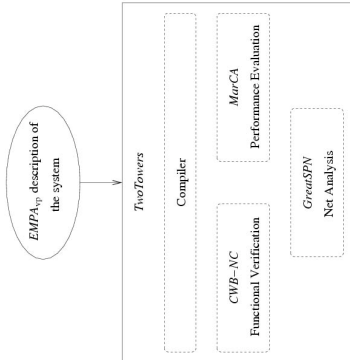
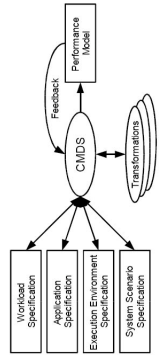
- AQN: Augmented Queueing Network
- EQN: Extended Queueing Network
- LTS: Layered Transaction System
- LQN: Layered Queueing Network
- MSC: Message Sequence Chart
- QN: Queueing Network
- UML: Unified Modelling Language
- XSLT: Extensible Stylesheet Language Transformations

Name	V1 (SPE)	V2 (PRIMA-UML)	V3	V4
Allgemeines				
Author	Smith / Williams	Cortellessa / Miranda / Grassi	Cortellessa / D'Ambrogio / Iazeolla	Gomaa / Menasce
Jahr	2002 (bzw. 1990)	2002	2001	2000, 2001
Literatur	[Smi90, Smi02]	[CM02, GM02]	[CDI01]	[GM00, GM01]
Architektureinschränkung	-	-	-	Komponentenbasierte Systeme
Applikationsdomäne	allgemein (Fallstudien für Web-Applikationen und eingebettete Realzeitsysteme)	allgemein (spezielle Erweiterung für mobile Software-Architekturen vorhanden)	allgemein	allgemein
Modellierung				
Software-Modell	MSC, Execution Graphs	Use-Case-, Sequenz-, Deploymentdiagramme, Execution Graphs	Klassen-, Sequenzdiagramme, Execution Graphs	Klassen-, Kollaborationsdiagramme
Performance-Modell	QN	EQN	LQN	EQN
Auswertungsmethode	Analyse/Simulation	Analyse	Analyse	Analyse
Feedback der Ergebnisse in das Software-Modell	nein	nein	nein	nein
Werkzeug	SPE-ED: Erstellung von Software-Execution-Models (Execution Graphs) und System-Execution-Models (QN), automatisiert die Berechnungen	Halbautomatische Ableitung der Performance-Modelle aus den Software-Modellen	Manuelle Transformation der Software-Modelle in LQNs, Nutzung von Standard CASE-Tools	-
Prozessmodell				
Sonstiges	- Performance Patterns/Antipatterns - ausführliche Abhandlung von Vorgehensmodellen (Integration in Unified Process)	proprietäre Annotation der UML-Diagramme		erster Ansatz für komponentenbasierte Systeme

Name	V5	V6 (CLISSPE)	V7	V8
Allgemeines				
Autor	Petriu / Gu / Wang	Menasce / Gomaa	Andolfi / Aquilani / Inverardi	Woodside / Hirschuk / Selic / Brayarov
Jahr	1999, 2002	1998, 2000	2000	2001
Literatur	[PW99, PS02, GP02]	[MG98], [MG00]	[AAB100]	[WHSB01]
Architektureinschränkung	Architekturmuster (Client/Server, Pipe-and-Filter, Broker usw.)	Client/Server	-	-
Applikationsdomäne	allgemein	allgemein	allgemein	allgemein
Modellierung				
Software-Modell	UML-Kollaborationen, Deploymentdiagramme	Use-Case-, Klassen-, Kollaborationsdiagramme	MSC, LTS	MSC
Performance-Modell	LQN	QN	QN	LQN
Auswertungsmethode	Analyse	Analyse	Analyse	Analyse
Feedback der Ergebnisse in das Software-Modell	vorgesehen	nein	?	nein
Werkzeug	XSL T-Transformationen	CLISSPE System	?	ObjectTime Developer, PAMB (Performance Analysis Model Builder)
Prozessmodell				
	<pre> graph LR UML[UML Tool] --> Transform[UML to LQN Transformation] Transform --> Model[Performance Model] Model --> Analyze[Performance Analysis] Analyze --> LQN[LQN Tool] </pre>			
Sonstiges	• automatische Transformation des Software-Modells in ein Performance-Modell mit Graphen-Transformation	• Performance-Annotationen in XML		

Name	V9	V10	V11	V12
Allgemeines				
Autor	Kähkipuro	Lindemann et. al.	Miguel et. al.	Arief / Spears
Jahr	2001	2002	2000	1999
Literatur	[Käh01]	[LTK+02]	[MLH+00]	[AS99]
Architektureinschränkung	-	-	-	-
Applikationsdomäne	Komponentenbasierte Systeme	allgemein	Real-Time-Systems	allgemein
Modellierung				
Software-Modell	UML, Zwischendarstellung Performance Modelling Language (PML)	Zustands-, Aktivitätsdiagramme	UML	Klassen-, Sequenzdiagramme
Performance-Modell	AQN	Generalisierte Semi-Markow Prozesse	OPNet Simulationsmodell	Simulation Modelling Language (SimML)
Auswertungsmethode	Analyse	Analyse	Simulation	Simulation
Feedback der Ergebnisse in das Software-Modell	ja	vorgesehen	ja	nein
Werkzeug	OAT (Object-oriented performance modelling and Analysis Tool)	DSPNexpress2000	AMG (Analysis Model Generator) SMG (Simulation Model Generator)	Parser für die SimML Notation (Perl- Parser erstellt C++-SIM Simulationen, Java-Parser erstellt JavaSIM Simulationen)
Prozessmodell				
Sonstiges				

Name	V13 (umIPSI)	V14	V15	V16
Allgemeines				
Autor	Marzolla / Balsamo	Hennig / Revill / Ponitsch	King / Pooley	Bernardi
Jahr	2003	2003	2000	2002
Literatur	[BM03] [Mar04]	[HRP03]	[KP00]	[BDM02]
Architektureinschränkung	-	-	-	-
Anwendungsdomäne	allgemein	allgemein	allgemein	allgemein
Modellierung				
Software-Modell	Use-Case-, Aktivitäts-, Deploymentdiagramme, UML Performance Profile	UML (Sequenz-, Kollaborations-, Deploymentdiagramme)	UML (Kollaborationsdiagramme mit eingebetteten Zustandsdiagramme)	UML (Zustands-, Sequenzdiagramme)
Performance-Modell	lipcppsim-Simulation	Simulation	Generalized Stochastic Petri-Nets	Generalized Stochastic Petri-Nets
Auswertungsmethode	Simulation	Simulation	Analyse	Analyse
Feedback der Ergebnisse in das Software-Modell	ja	ja	nein	nein
Werkzeug	umIPSI (UML Performance Simulator) lipcppsim	OMNET++ mit SPE-Erweiterungen	SPNP	nein
Prozessmodell				
			1. UML-Diagramme 2. manuelle Transformation in GSPN 3. numerische Auswertung der Petri-Netze mit SPNP	
Sonstiges	• PhD-Thesis veröffentlicht	• Tests mit JBoss		• PhD-Thesis veröffentlicht

Name	V17	V18 (PERMABASE)	V19	V20 (Capacity Planning)
Algemeines				
Autor	Bernardo	Waters et. al.	Hoeben	Menasce / Almeida / Dowdy
Jahr	2000	2001	2000	1994-2004
Literatur	[BCD00a]	[WLA+01]	[Hoe00]	[MA02],[MAD04]
Architekturreinschränkung	-	-	-	Client-Server Systeme
Applikationsdomäne	allgemein	ATM-based Applications and Services	allgemein	allgemein
Modellierung				
Software-Modell	ADL (AEMPA)	UML	UML (Klassen-, Komponenten-, Sequenz-, Kollaborations-, Deploymentdiagramme)	-
Performance-Modell	Extended Markovian Process Algebra (EMPA)	Zwischendarstellung CMDS (Composite Model Data Structure)	QN	QN
Auswertungsmethode	Analyse	Analyse	Analyse	Analyse
Feedback der Ergebnisse in das Software-Modell	nein	Feedback in CMDS	nein	nicht möglich
Werkzeug	TwoTowers 2.0	?	?	Excel-Spreadsheets zur Auswertung von QN.
Prozessmodell				
				
Sonstiges				
	PhD-Thesis veröffentlicht			keine Modellierung der Software-Architektur, nur System-Modelle

B. Folien der Übungen

Zum Training der Versuchsteilnehmer wurde für jedes untersuchte Verfahren ein Foliensatz zusammengestellt. Die enthaltenen Abbildungen und Beispiele entstammen jeweils aus der zugehörigen Literatur [Smi02], [MAD04], [Mar04]. Mit den Folien kann nachvollzogen werden, auf welcher Wissensbasis die Versuchsteilnehmer ihre Performance-Analysen durchführten.

Software Performance Engineering (SPE)

Smith/Williams

Inhalt

- Allgemeines
- Vorgehensmodell
- SPE und UML
 - erweiterte Sequenzdiagramme
- Software Execution Models
 - Execution Graphs
- System Execution Models
 - Queueing Network
- Performance-orientiertes Design
- Beispiel

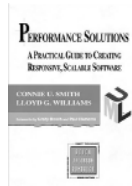
31.08.2004

Software Performance Engineering

2

Allgemeines

- 1981: Prägung des Begriffs „Software Performance Engineering“ durch Connie U. Smith
- 1990: Buch „Performance Engineering of Software Systems“
 - erste vollständige Methode zur Performance-Analyse von Software-Systemen während des Entwurfs
- 2002: Buch „Performance Solutions“
 - Integration der UML
 - Anpassungen für verteilte bzw. eingebettete Systeme
 - Tool SPE-ED
 - Performance-Patterns/Antipatterns



31.08.2004

Software Performance Engineering

3

SPE-Vorgehensmodell

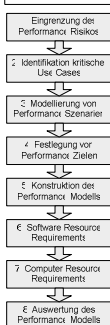
31.08.2004

Software Performance Engineering

4

Vorgehensmodell

- Beispiel: Entwurf eines Geldautomaten



31.08.2004

Software Performance Engineering

5

Vorgehensmodell

- Festlegung des Umfangs des SPE-Projekts je nach Signifikanz der Performance
- Erstellung eines detaillierteren Modells bei Performance-kritischeren Anwendungen
- Beispiel Geldautomat: geringes Performance-Risiko, einfaches Modell ausreichend

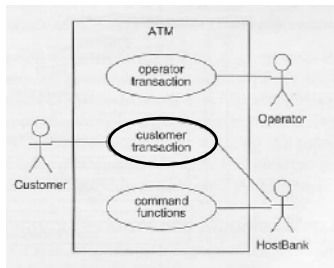
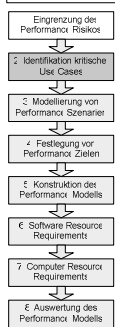


31.08.2004

Software Performance Engineering

6

Vorgehensmodell

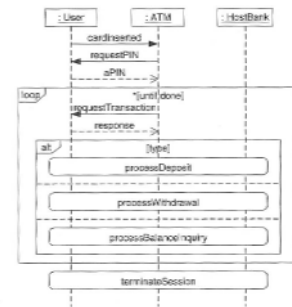


31.08.2004

Software Performance Engineering

7

Vorgehensmodell

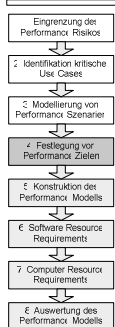


31.08.2004

Software Performance Engineering

8

Vorgehensmodell



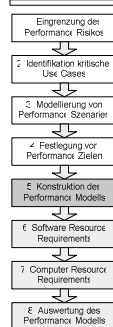
- Konkrete Vorgaben für z.B. Antwortzeiten, Durchsatz, Ressourcennutzung usw.
- Beispiel Geldautomat: „Die Bedienung darf nicht länger als 30 Sekunden dauern.“

31.08.2004

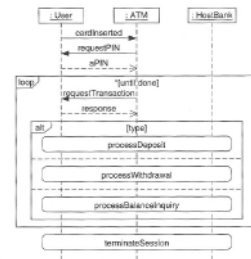
Software Performance Engineering

9

Vorgehensmodell



- Transformation der Sequenzdiagramme in Ausführungsgraphen

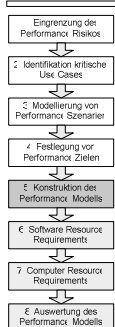


31.08.2004

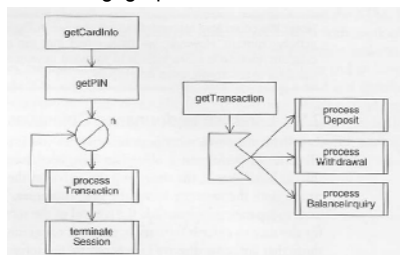
Software Performance Engineering

10

Vorgehensmodell



- Transformation der Sequenzdiagramme in Ausführungsgraphen

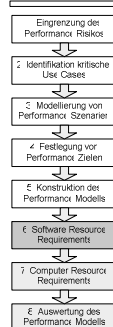


31.08.2004

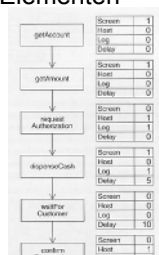
Software Performance Engineering

11

Vorgehensmodell



- Bestimmung der Anforderungen von Software-Elementen



31.08.2004

Software Performance Engineering

12

Vorgehensmodell



- Bestimmung der Anforderungen von Hardware-Elementen

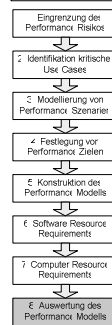
Devices	CPU	Disk	Display	Delay	Net
Quantity	1	1	1	1	1
Service Units	Sec.	Phys. I/O	Screens	Units	Msgs.
Screen	0.001		1		
Host	0.005			3	
Log	0.001	1			
Delay				1	
Service Time	1	0.02	1	1	0.05

31.08.2004

Software Performance Engineering

13

Vorgehensmodell



- Berechnung der Performance-Werte
- Beispiel Geldautomat: „Die Bedienung dauert 29 Sekunden und ist somit knapp innerhalb der Vorgaben“
- bei Nichteinhaltung der Vorgaben:
 - Modifikation der Software
 - Modifikation der Szenarien
 - Überarbeitung der Performance-Ziele

31.08.2004

Software Performance Engineering

14

SPE und UML

31.08.2004

Software Performance Engineering

15

SPE und UML

- UML: Standard-Notation für die Modellierung objektorientierter Systeme
- Basis für die Erstellung des Performance-Modells
- Erweiterungen der UML für Performance-spezifische Informationen (angelehnt an Message Sequence Charts)
 - hierarchische Strukturierung von Sequenzdiagrammen
 - graphische Repräsentation von Schleifen und Auswahlknoten in Sequenzdiagrammen
 - Nebenläufigkeit in Sequenzdiagrammen

31.08.2004

Software Performance Engineering

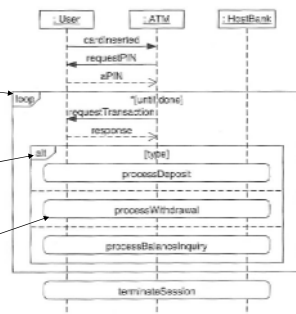
16

Beispiel: Erweiterte Sequenzdiagramme

Schleife:
gebunden an Schleifenklausel

Auswahl:
gebunden an Bedingungsklausel

Referenz:
verzweigt in ein weiteres
Sequenzdiagramm



31.08.2004

Software Performance Engineering

17

Sequenzdiagramme: Kontrollfluss

- • Prozeduraufrufe: Fortsetzung der Ausführung erst nach Beendigung aller verschachtelten Aufrufe
- • Nichtprozeduraler Aufruf: nächster Schritt in der Sequenz
- • Asynchrone Kommunikationen zwischen zwei Objekten
- > • return-Anweisung eines Prozeduraufrufs

31.08.2004

Software Performance Engineering

18

Erweiterte Features für Sequenzdiagramme:

-  • **Referenz**
 - Details in einem weiteren Sequenzdiagramm
-  • **Iteration**
 - n-malige Wiederholung des eingeschlossenen Abschnitts
-  • **Alternative**
 - Ausführung genau eines Abschnittes gemäß der Bedingungsklausel
-  • **Optionale Abschnitte**
 - Ausführung des eingeschlossenen Abschnitts optional gemäß der Bedingungsklausel
-  • **Parallele Abschnitte**
 - gleichzeitige Ausführung der eingeschlossenen Abschnitte

31.08.2004

Software Performance Engineering

19

Software Execution Models

31.08.2004

Software Performance Engineering

20

Vom Sequenzdiagramm zum Ausführungsgraph

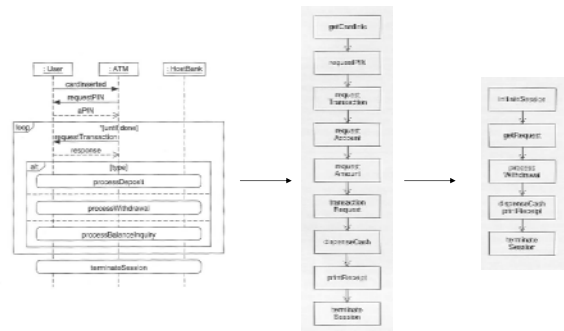
- **Execution Graph**
 - Modell zur vereinfachten Aufnahme von Performance-Informationen
 - Ähnlichkeit mit UML-Aktivitätsdiagrammen
- Umwandlung jedes Aufrufes im Sequenzdiagramm in einen Knoten
- Zusammenfassung verwandter bzw. Performance-unkritischer Aufrufe in einen einzigen Knoten
- Ergänzung von Ausführungshäufigkeiten / -wahrscheinlichkeiten

31.08.2004

Software Performance Engineering

21

Beispiel: Sequenzdiagramm -> Ausführungsgraph

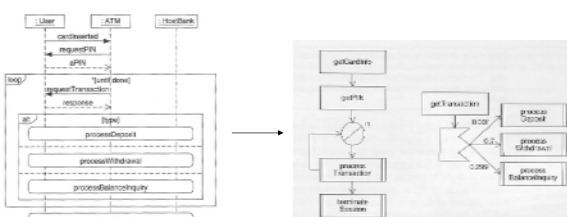


31.08.2004

Software Performance Engineering

22

Beispiel: Sequenzdiagramm -> Ausführungsgraph



31.08.2004

Software Performance Engineering

23

Ausführungsgraphen

-  • **Basisknoten**
 - Funktionale Komponente
-  • **Erweiterter Knoten**
 - Verfeinerte Funktion in einem Subgraphen
-  • **Wiederholungsknoten**
 - Schleife: n-malige Wiederholung einer Reihe von Knoten
- **Entscheidungsknoten**
 - Ausführung geknüpft an Bedingung
 - versehen mit Wahrscheinlichkeit
- **Parallelnoten**
 - parallele Ausführung von Knoten
 - Fortsetzung des Kontrollflusses erst nach Berechnung aller Knoten
- **Spaltungsknoten**
 - parallele Ausführung von Knoten
 - Fortsetzung des Kontrollflusses auch wenn nicht alle Knoten berechnet wurden

31.08.2004

Software Performance Engineering

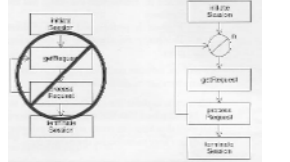
24

Ausführungsgraphen: Nicht erlaubte Konstrukte

- mehrere Anfangsknoten für einen Graphen



- Schleifen ohne Wiederholungsknoten



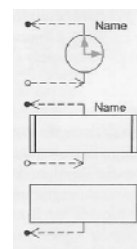
31.08.2004

Software Performance Engineering

25

Ausführungsgraphen: Synchronisation

Aufrufer Prozess:

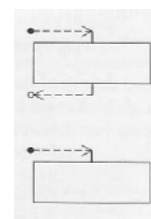


Synchroner Aufruf:
der Aufrufer wartet
auf eine Antwort

Verzögerter synchroner
Aufruf: Weiterrechnen,
danach Warten auf
Antwort

Asynchroner Aufruf:
keine Antwort

Aufgerufener Prozess:



Antwort

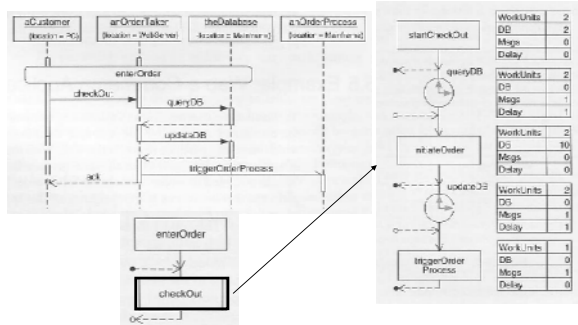
keine Antwort

31.08.2004

Software Performance Engineering

26

Ausführungsgraphen: Synchronisation - Beispiel



31.08.2004

Software Performance Engineering

27

Eingabedaten für SPE

- Performance-kritische Szenarios
 - modelliert als Sequenzdiagramme, Ausführungsgraphen
- Performance-Ziele
 - konkrete Vorgaben, die zu erfüllen sind
- Systemumgebung
 - Hardware-Konfiguration, Betriebssystem, Middleware
- Software Resource Requirements
 - Berechnungsanforderung für jeden Rechenschritt
- Computer Resource Requirements
 - Abbildung der Software Resource Requirements auf die Ausführungsumgebung

in der Übung
vorgegeben

31.08.2004

Software Performance Engineering

28

Software Resource Requirements

- Identifikation der Software Ressourcen
 - Beispiel: Berechnungseinheiten, Datenbankzugriffe, Netzwerknachrichten
- Schätzung der Werte für die Software-Ressourcen
 - best- und worst-case**, später detaillierter
 - Beispiel: Operation x braucht im best-case drei Datenbankzugriffe
- Angabe der CPU-Belastung
 - Beispiel: WorkUnits über einer Skala von 1 (niedrig) bis 5 (hoch)
- Beispiel: Autorisierung einer Transaktion mittels einer Datenbank
 - hier: Spezifikation der best-case Werte



31.08.2004

Software Performance Engineering

29

Software Resource Requirements

Software Resource Type	Einheit
CPU usage	Work Units, Anzahl der Instruktionen oder Zeit
SQL	Anzahl und Typ (read, write, update, ...)
File I/O	Anzahl der logischen oder physischen I/Os
Messages	Größe (z.B. in KByte) und Typ (LAN, Internet)
Log-Files	Anzahl der Log-Events
Middleware-calls	Anzahl und Typ (connectionOpen, queueGet, requestSend, ...)
Calls to different processes, threads or processor	Anzahl, Typ und Ziel des Aufrufs
Delay for remote processing	Zeit eines Requests

31.08.2004

Software Performance Engineering

30

Computer Resource Requirements

- Abbildung der Software Anforderungen auf Hardware-Elemente
- Typ, Anzahl und Einheit der relevanten Hardware-Elemente im System
- Spezifikation, wie hoch die Software Requirements auf den Computer Ressource Requirements sind
- Dauer einer Einheit für jede Ressource (z.B. durch Messungen)

Device	CPU	Disk	Network
Quantity	1	1	1
Service Unit	Kinstr.	Phys. I/O	Mbps.
WorkUnit	20	0	0
DB	500	2	0
Mbps	10	2	1
Service time	0.00001	0.02	0.01

31.08.2004

Software Performance Engineering

31

Computer Resource Requirements

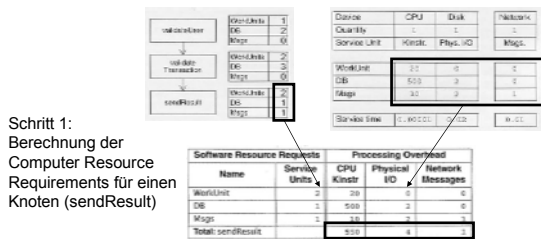
Computer Resource Type	Einheit
CPU	Zeit, Anzahl der Instruktionen
Disks, I/O	Anzahl der Operationen
Benutzer	Zeit
Netzwerk	Anzahl der Nachrichten
Verzögerung (ohne Queue, z.B. Bildschirm, Tastatur...)	Zeit
Box (mit Queue, z.B. Drucker)	Zeit

31.08.2004

Software Performance Engineering

32

Computer Resource Requirements: Berechnung der Antwortzeit



31.08.2004

Software Performance Engineering

33

Computer Resource Requirements: Berechnung der Antwortzeit

Schritt 2: Berechnung der Computer Resource Requirements für alle Knoten eines Graphen

Processing Step	CPU Konstr.	Physical I/O	Network Messages
validateUser	1,000	0	0
validateTransaction	1,000	0	0
sendResult	10	2	1
Total: authorizeTransaction	3,000	2	1

Schritt 3: Berechnung der best-case Antwortzeit (ggf. Einbeziehung von Iterationen und Wahrscheinlichkeiten bei Auswahlknoten)

$$(3110 * 0,00001) + (14 * 0,02) + (1 * 0,01) = 0,3211 \text{ Sekunden}$$

31.08.2004

Software Performance Engineering

34

System Execution Models

System Execution Models

- Software Execution Model:
 - zum **schnellen Feedback** von Performance-Problemen in frühen Entwurfsphasen
 - keine Berücksichtigung von anderen Arbeitslasten oder mehreren Benutzern
 - Vernachlässigung der Verzögerungen, die bei konkurrierender Ressourcennutzung auftreten
- System Execution Model:
 - Konstruktion erst nach Lösung aller Performance-Probleme im Software Execution Model
 - Modellierung von **konkurrierenden Ressourcenzugriffen** mittels **Queueing Networks**
 - zur Identifikation von Flaschenhälsen in der System-Architektur
 - automatisierte Berechnung durch Tools

31.08.2004

Software Performance Engineering

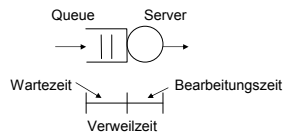
35

31.08.2004

Software Performance Engineering

36

Queueing Networks



Performance Metriken:

- Verweilzeit
- Auslastung
- Durchsatz
- Queue-Länge

31.08.2004

Software Performance Engineering

37

Queueing Networks

Annahme: Job Flow Balance

- Rate der fertiggestellten Jobs (Durchsatz) = Ankunftsrate

Beispiel:

Vorgabe:

- Ankunftsrate $\lambda = 0,4$ jobs/sec
- Mittlere Bearbeitungszeit $S = 2$ sec

Berechnung:

- Durchsatz, $X = \lambda$ 0,4 jobs/sec (job flow balance)
- Auslastung, $U = X * S$ 0,4/sec * 2 sec = 0,8
- Verweilzeit, $RT = S / (1-U)$ 2 sec / (1-0,8) = 10 sec
- Queue-Länge, $N = X * RT$ 0,4/sec * 10 sec = 4 jobs (Little's Formel)

31.08.2004

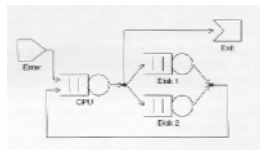
Software Performance Engineering

38

Queueing Networks

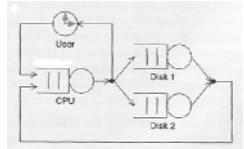
offenes QN:

- Eingang/Ausgang in das Netzwerk
- Anzahl der Jobs variiert mit der Zeit



geschlossenes QN:

- keine externen Ankünfte im Netzwerk
- Zirkulation fester Anzahl von Jobs



31.08.2004

Software Performance Engineering

39

Offenes Queueing Network: Berechnungen

Vorgabe:

- λ Ankunftsrate
- V_i Anzahl der Besuche (Visits) eines Gerätes
- S_i Mittlere Bearbeitungszeit (Service Time) eines Gerätes

Berechnung:

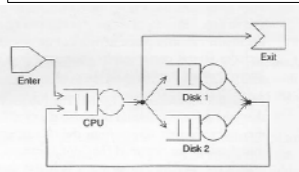
- Systemdurchsatz X_0 : $X_0 = \lambda$
- Durchsatz von Gerät i : $X_i = X_0 * V_i$
- Auslastung von Gerät i : $U_i = X_i * S_i$
- Verweilzeit von Gerät i : $RT_i = S_i / (1-U_i)$
- Länge der Queue Gerät i : $N_i = X_i * RT_i$
- Länge der Systemqueue: $N = \sum N_i$
- Systemantwortzeit: $RT = N / X_0$

31.08.2004

Software Performance Engineering

40

Offenes Queueing Network: Beispiel



Systemankunftsrate:

- $\lambda = 5$ jobs / sec

Anzahl der Besuche V_i :

- CPU 5
- Disk1 3
- Disk2 1

Mittlere Bearbeitungszeit S_i :

- CPU 0,01
- Disk1 0,03
- Disk2 0,02

Metrik	CPU	Disk1	Disk2
1. Durchsatz $X_i = X_0 * V_i$	25	15	5
2. Bearbeitungszeit S_i (Vorgabe)	0,01	0,03	0,02
3. Auslastung $U_i = X_i * S_i$	0,25	0,45	0,10
4. Verweilzeit $RT_i = S_i / (1-U_i)$	0,013	0,055	0,022
5. Queue Länge $N_i = X_i * RT_i$	0,325	0,825	0,111
Gesamte Jobs im System = 0,325 + 0,825 + 0,110 = 1,26			
Systemantwortzeit: 1,26 / 5 = 0,252 sec			

31.08.2004

Software Performance Engineering

41

System Execution Model

Eingaben für das Queueing Network

- Systemankunftsrate (ermittelt in einem Performance Walkthrough)
- Anzahl der Besuche an einem Gerät (berechnet mit dem Software Execution Model)

Processing Step	CPU Konstr.	Physical I/O	Network Messages
validateUser	1,024	4	0
validateTransaction	1,540	6	0
readInput	0,000	4	0
Total authorized transaction	2,564	14	0

- Mittlere Bearbeitungszeiten (ebenfalls im Software Execution Model festgelegt)

Service time: CPU=0,001 I/O=0,02 Network=0,001

- Berechnung der Ausgaben (Systemantwortzeit, Queue-Längen usw.) mit Tool

31.08.2004

Software Performance Engineering

42

Performance-orientiertes Design

31.08.2004

Software Performance Engineering

43

Performance-Prinzipien

Performance Objectives Principle	Define specific, quantitative, measurable performance objectives for performance scenarios.
Instrumenting Principle	Instrument systems as you build them to enable measurement and analysis of workload scenarios, resource requirements, and performance objectives compliance.
Centering Principle	Locally the dominant workload functions and minimize their processing.
Priority Point Principle	For responsiveness, bring critical activities connected to the earliest feasible point in time, such that restoring the connection is cost-effective.
Locality Principle	Create services, functions, and objects that are close to physical computer resources.
Processing Versus Frequency Principle	Minimize the product of processing times frequency.
Shared Resources Principle	Share resources when possible. When needed be aware to account, minimize the sum of the holding time plus the scheduling time.
Parallel Processing Principle	Execute processing in parallel to reduce the processing, speedup, offset communication overhead and resource contention delays.
Spread the Load Principle	Spread the load when possible by processing conflicting tasks at different times or in different places.

31.08.2004

Software Performance Engineering

44

Performance-Patterns

Pattern	Description	Principle(s)
Fast Path	Identify dominant workload functions and streamline the processing to do only what is necessary.	Centering
Fast Things First	Focus on the relative importance of processing tasks to ensure that the least important tasks will be the ones omitted if everything cannot be completed within the time available.	Centering
Coupling	Match the interfaces to objects with their most frequent usage.	Centering Locality Processing Versus Frequency
Bunching	Combine requests into bundles so the overhead processing is executed once for the entire batch instead of for each individual item.	Processing Versus Frequency
Alternate Routes	Spread the demand for high usage objects spatially, that is, to different objects or locations.	Spread-the-Load
Flex Time	Spread the demand for high usage objects temporally, that is, to different periods of time.	Spread-the-Load
Slender Cycle Functions	Minimize the amount of work that must execute at regular intervals.	Centering

31.08.2004

Software Performance Engineering

45

Performance-Antipatterns

Antipattern	Problem	Solution	Principle Violated
Global Clone	Occurs when a single object is replicated to all nodes in the system, causing unnecessary overhead. The overhead results in excessive message traffic that can degrade performance.	Restrict the design to distribute information, calculate, and store data and behavior locally.	Locality Processing Versus Frequency
Excessive Dynamic Allocation	Occurs when an application unnecessarily creates and destroys objects during its execution. The overhead incurred to create and destroy these objects has a significant impact on performance.	Use "Recycle" objects, that is, objects that are reused. Alternatively, use object pools. If the overhead is still too high, consider using a garbage collector.	Priority Point Locality Processing Versus Frequency
Calculated Resource Hunt	Occurs when an object must be located in order to find the information that it needs. If a large amount of processing is required for each "look," performance will suffer.	Reduce the design to provide information access paths that do not require a Calculated Resource Hunt (or to reduce the cost of each "look").	Locality
Overlaid Bridge	Occurs at a point in execution where only one, or a few, processes may continue to execute concurrently (e.g., when executing a database). Other processes are delayed while they wait for their turn.	To eliminate the congestion, use the Shared Resources Principle to minimize contention.	Shared Resources Spread-the-Load
Traffic Jam	Occurs when one process creates a backlog of jobs that processors must wait to process. This is not possible, possibly sufficient processing power to handle the worst case load.	Steps to eliminate the backlog of the backlog. If this is not possible, provide sufficient processing power to handle the worst case load.	Shared Resources Spread-the-Load

31.08.2004

Software Performance Engineering

46

Beispiel

- Web-Auftritt einer Fluggesellschaft
- Planung von Flugreisen im Web
- Ordern von Tickets

31.08.2004

Software Performance Engineering

47

31.08.2004

Software Performance Engineering

48

1. Eingrenzen des Performance-Risikos

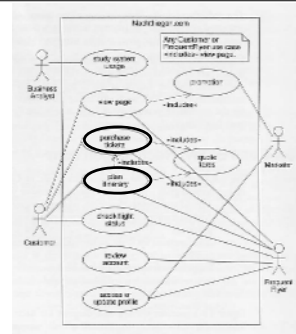
- mittleres Performance-Risiko
- bei schlechter Performance: keine Benutzung der Website -> Umsatzeinbußen

31.08.2004

Software Performance Engineering

49

2. Performance-kritische Use-Cases



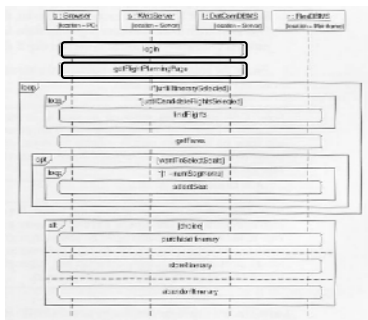
31.08.2004

Software Performance Engineering

50

3. Modellierung von Szenarien

plantinerary



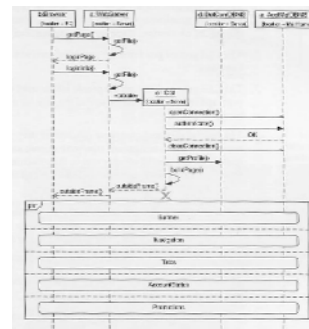
31.08.2004

Software Performance Engineering

51

3. Modellierung von Szenarien

login



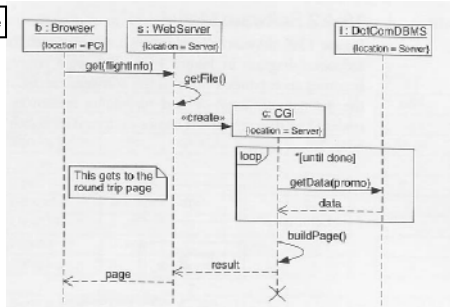
31.08.2004

Software Performance Engineering

52

3. Modellierung von Szenarien

getFlightPlanningPage



31.08.2004

Software Performance Engineering

53

4. Festlegung von Performance-Zielen

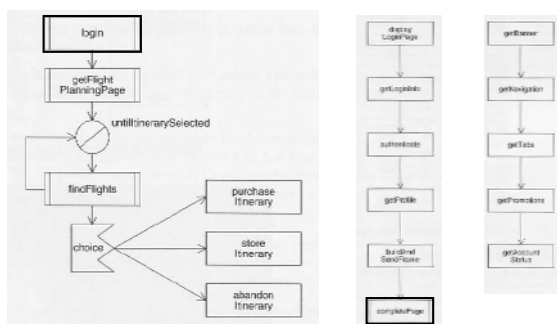
- maximale Login-Zeit: 8 Sekunden
- ansonsten zunächst keine Vorgaben

31.08.2004

Software Performance Engineering

54

5. Konstruktion eines Performance-Modells

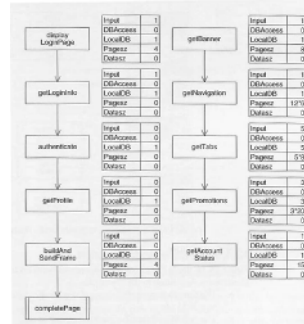


31.08.2004

Software Performance Engineering

55

6. Software Resource Requirements



31.08.2004

Software Performance Engineering

56

- Input: Größe der Eingabenachricht (KByte)
- DBAccess: Anzahl der Zugriffe auf die Mainframe-Datenbank
- LocalDB: Anzahl der Zugriffe auf das DotComDBMS
- Pagesz: Größe der Seite, die dem Benutzer gezeigt wird (KByte)
- Datasz: Größe der Daten, die aus dem Mainframe empfangen werden (KByte)

7. System Resource Requirements

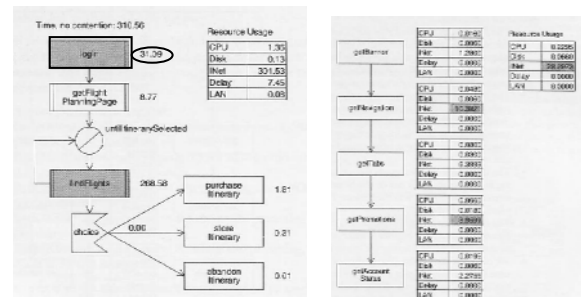
Devices	CPU	Disk	INet	Delay	LAN
Quantity	6	3	1	1	1
Service Units	sec	Phys I/O	KBytes	sec	Msgs
Input	0.002		1		
DBAccess	0.0005			0.25	1
LocalDB	0.01	2			
Pagesz	0.0005		1		
Datasz	0.0005		1		1
Service Time	1	0.003	1.14222	1	0.000164

31.08.2004

Software Performance Engineering

57

8. Auswertung des Performance-Modells



31.08.2004

Software Performance Engineering

58

Änderung des Entwurfs

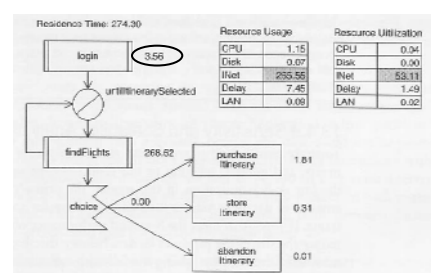
- Beispiele:
 - kleinere Webseiten verschicken
 - Verwendung von Tabellen anstatt von Frames um mehrfache Requests zu umgehen
 - Möglichkeit zur Festlegung einer Startseite durch den User zur schnelleren Navigation
 - Verwendung von weniger aufwendigen Graphiken
 - ...

31.08.2004

Software Performance Engineering

59

Änderung des Entwurfs



31.08.2004

Software Performance Engineering

60

Fazit: Lernziele der Übung



- Verstehen des SPE-Prozesses
- Wissen, wie ein Sequenzdiagramm in einen Ausführungsgraphen transformiert wird
- Wissen, wie Software Resource Requirements festgelegt werden
- Wissen, wie Computer Resource Requirements festgelegt werden, und wie damit Performance-Werte berechnet werden können

Capacity Planning

Menasce / Almeida

Inhalt

- Allgemeines
- Vorgehensmodell
- Workload-Modellierung
- Queueing Networks
 - qualitative Aspekte
 - quantitative Aspekte
- Beispiel

31.08.2004

Performance by Design

2

Allgemeines

- seit Anfang der 70er: Einsatz von Queueing Networks in der Performance-Analyse
- 1994: „Capacity Planning and Performance Modeling: from mainframes to client-server systems“
- 1998: „Capacity Planning for Web Performance: metrics, models, and methods“
- 2000: „Scaling for E-Business: technologies, models, performance, and capacity planning“
- 2002: „Capacity Planning for Web Services“
- 2004: „Performance by Design“



31.08.2004

Performance by Design

3

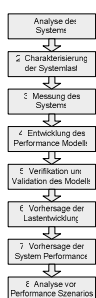
Vorgehensmodell

31.08.2004

Performance by Design

4

Vorgehensmodell



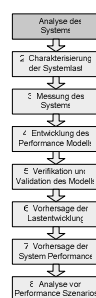
- Beispiel: Performance-Modellierung eines einfachen Web-Servers

31.08.2004

Performance by Design

5

Vorgehensmodell



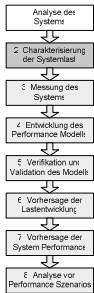
- Verstehen der Systemarchitektur
- Welche Arten von Hardware / Software werden benutzt?
- Beispiel: ein Web-Server bestehend aus einer CPU und einer Festplatte

31.08.2004

Performance by Design

6

Vorgehensmodell



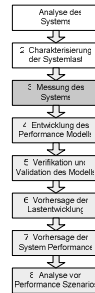
- Workload-Modellierung
- Ankunftsrate von Anfragen
- Einteilung der Systemlast in mehrere Klassen
- Beispiel: Beobachtung des Web-Server für eine Stunde
 - 14040 Anfragen für HTML-Dateien (durchschnittliche Größe: 3000 Bytes)
 - 1034 Anfragen für Image-Dateien (durchschnittliche Größe: 15000 Bytes)

31.08.2004

Performance by Design

7

Vorgehensmodell



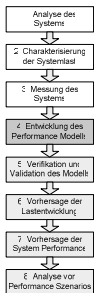
- Benchmarks
- Netzwerkgeschwindigkeit, Datenbankkapazität usw.
- Beispiel:
 - CPUDemand = $0,008 + 0,002 * \text{Größe der Anfrage}$
 - konstanter Anteil für das Öffnen einer TCP-Verbindung, die Analyse des HTTP-Requests und das Öffnen der Datei
 - variabler Anteil proportional zur Dateigröße, CPU ist an jeder I/O-Operation beteiligt
 - durchschnittliche Bearbeitungszeit der Festplatte für einen Block von 1000 Bytes: 12 msec

31.08.2004

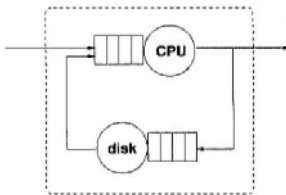
Performance by Design

8

Vorgehensmodell



- Erstellung eines Queueing Networks
- Beispiel:
 - Modellierung als offenes QN

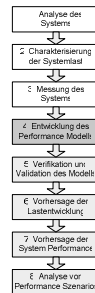


31.08.2004

Performance by Design

9

Vorgehensmodell



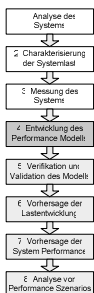
- Berechnung aller Eingabeparameter
- Beispiel:
 - Ankunftsraten:
 - $\lambda_{\text{HTML}} = 14040 / 3600 = 3,9 \text{ Anfragen / sec}$
 - $\lambda_{\text{Image}} = 1034 / 3600 = 0,29 \text{ Anfragen / sec}$
 - Berechnung der benötigten Rechenzeiten
 - $D_{\text{CPU,HTML}} = 0,008 + 0,002 * 3 = 0,014 \text{ sec}$
 - $D_{\text{CPU,Image}} = 0,008 + 0,002 * 15 = 0,038 \text{ sec}$
 - $D_{\text{Disk,HTML}} = 3 * 0,012 = 0,036 \text{ sec}$
 - $D_{\text{Disk,Image}} = 15 * 0,012 = 0,18 \text{ sec}$

31.08.2004

Performance by Design

10

Vorgehensmodell



- Eingabe der Parameter in ein Spreadsheet zur automatischen Berechnung des QN

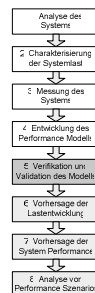
	HTML	Images
Arrival Rate (req / sec)	3,90	0,29
Service Demands (sec)		
CPU	0,014	0,038
Disk	0,036	0,180
Utilizations (%)		
CPU	5,5	1,1
Disk	14	5,2
Residence Times (sec)		
CPU	0,015	0,041
Disk	0,045	0,223
Response Times (sec)		
CPU	0,060	0,264

31.08.2004

Performance by Design

11

Vorgehensmodell



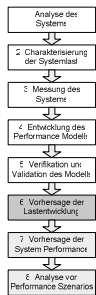
- Wurde das richtige Modell für das System erstellt?
- Verhält sich das Modell wirklich genauso wie das System?

31.08.2004

Performance by Design

12

Vorgehensmodell



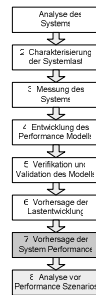
- Statistische Methoden zur Ermittlung zukünftiger Workloads
- Beispiel: Erhöhung der Systemlast um das 5-fache
 - Ankunftsrate:
 - $\lambda_{\text{HTML}} = 5 * 3,9 = 19,5$ Anfragen / sec
 - $\lambda_{\text{Image}} = 5 * 0,29 = 1,45$ Anfragen / sec

31.08.2004

Performance by Design

13

Vorgehensmodell



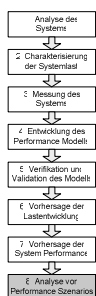
- erneute Berechnung des QN
 - Response Time:
 - HTML: 0,93 sec (das 15,5 fache)
 - Images: 4,61 sec (das 17,5 fache)
 - Utilization (disk):
 - 96 % (vorher: 19,2 %)
- Erreichung der Maximalkapazität bei 5-facher Systemlast

31.08.2004

Performance by Design

14

Vorgehensmodell



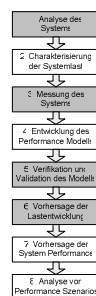
- Testen verschiedener Entwurfalternativen
- jeweils Modifikation des QN
- Beispiel: „Wie wirkt sich eine doppelt so schnelle CPU auf die Performance des Systems aus?“

31.08.2004

Performance by Design

15

Vorgehensmodell



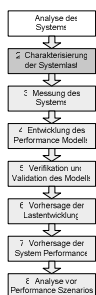
in der Übung
vorgegeben

31.08.2004

Performance by Design

16

Workload-Modellierung



31.08.2004

Performance by Design

17

Workload-Modellierung

- Typen von Workloads
 - Natürliche Modelle (Traces, Logs von Webservern)
 - Künstliche Modelle
 - Instruction Mixes
 - Kernels
 - Synthetic Programs
 - Artificial Benchmarks
 - Drivers
 - ausführbare / nicht-ausführbare Modelle

31.08.2004

Performance by Design

18

Workload-Modellierung

- Systemlast gekennzeichnet durch
 - Intensität
 - Ankunftsrate (z.B. 5 Anfragen / sec)
 - Anzahl der Clients und Think Times
 - Anzahl der gleichzeitig ausgeführten Prozesse und Threads
 - Service-Anforderungen ($D_{i1}, D_{i2}, \dots, D_{in}$), wobei D_{ij} die Anforderung der Last i an Ressource j ist (z.B. Anfrage i braucht 2 sec auf Ressource j)

31.08.2004

Performance by Design

19

Workload-Modellierung

- Anfragen an ein System
 - unterschiedliche Anforderungen bzgl. CPU-Zeit, I/O, Speicherverbrauch, Netzwerkbelastung
 - z.B. Abfrage statischer HTML-Seite gegenüber Datenbanktransaktion
 - Mittelung** der Anforderungen über alle Anfragen -> **ungenaue Modellbildung**

31.08.2004

Performance by Design

20

Workload-Modellierung

- Einteilung aller Anfragen in **Klassen**
 - z.B. einfache, mittlere, komplexe Anfragen
- Kriterien
 - Ressourcenverbrauch (hoch, niedrig)
 - Anwendungen (ftp, http)
 - Dokumenttyp (html, jpg, avi)
 - Herkunft (.de, .uk, .fr)
 - ...
- bei ähnlichen Objekten: **Mittelwertbildung**
- bei unterschiedlichen Objekten: **Cluster-Analyse**

31.08.2004

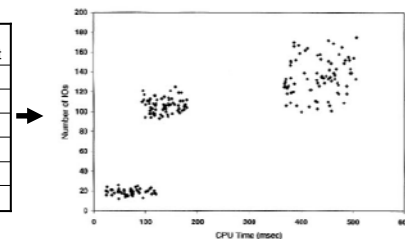
Performance by Design

21

Workload-Modellierung: Cluster-Analyse

- Darstellung der Anfragen eines Beispieldurchlaufs als Punkte in einem mehrdimensionalen Raum (z.B. bzgl. Anzahl der I/O Operationen und CPU-Zeit)

ID	CPU-Time	I/O-Count
...	...	...
...	...	...
005	25.4	12
006	32.8	18
007	107.6	10
...	...	...



31.08.2004

Performance by Design

22

Workload-Modellierung: Cluster-Analyse

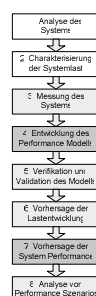
- Berechnung
 - Euklidischer Abstands der Punkte
 - Koordinaten der Mittelpunkte von Clustern
- k-means-Algorithmus
 - Setze die Anzahl der Cluster auf k .
 - Wähle k Startpunkte als initiale Mittelpunkte.
 - Ordne jedem weiteren Punkt den jeweils nächsten Mittelpunkt zu und berechne dabei den Mittelpunkt neu.
 - Wiederhole 3. bis sich kein Mittelpunkt mehr ändert oder eine maximale Anzahl an Durchläufen erfolgt ist.

31.08.2004

Performance by Design

23

Queueing-Networks



31.08.2004

Performance by Design

24

Queueing Networks: qualitative Aspekte

- offene/geschlossene Klassen
- Ressourcentypen
- Blocking
- Software Contention
- Gleichzeitige Ressourcennutzung
- Scheduling-Strategien

31.08.2004

Performance by Design

25

Queueing Networks

- Sicht auf ein Computersystem als eine Ansammlung von zusammenhängenden Warteschlangen
- einfaches Modell, das die Berechnung von Performance-Werten (wie Antwortzeiten, Durchsatz, Auslastung...) zulässt
- nicht unbedingt genaue absolute Werte, sondern relativer Vergleich verschiedener Architekturalternativen

31.08.2004

Performance by Design

26

QN: offene/geschlossene Klassen

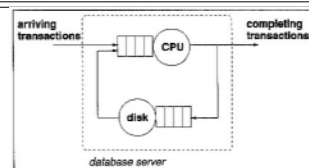
- Aufbau des Queueing Networks abhängig von der **Klassifizierung der Systemlast**
 - mehrere Lastklassen
 - offene, geschlossene, gemischte Systemlast

31.08.2004

Performance by Design

27

QN: offene Klassen



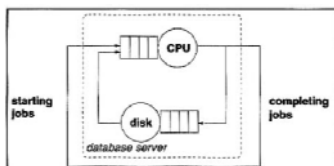
- Lastintensität abhängig von **Ankunftsrate** (z.B. 0,5 Transaktionen / Sekunde)
- Lastintensität hängt nicht von Anzahl der Benutzer im System ab
- unbegrenzte Anzahl von Benutzern
- Durchsatz = Ankunftsrate

31.08.2004

Performance by Design

28

QN: geschlossene Klassen



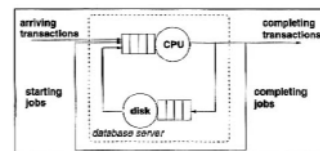
- Lastintensität anhängig von der **Benutzeranzahl** im System (z.B. 7 Benutzer im System)
- begrenzte Anzahl von Benutzern
- Durchsatz ist ein Ausgabeparameter

31.08.2004

Performance by Design

29

QN: gemischte Klassen



- beinhaltet **offene und geschlossene** Klassen
- z.B. Online-Transaktionen (offen) und gleichzeitig Batch-Generierung von Managementberichten (geschlossen)

31.08.2004

Performance by Design

30

QN: Ressourcentypen

- Load independent
 - Servicerate hängt nicht von der Systemlast ab
- Load dependent
 - Servicerate variiert mit der Systemlast (z.B. ist eine Ressource bei größerer Systemlast stärker ausgelastet)
- Delay
 - keine Warteschlange, sofortige Bearbeitung
 - konstante Bearbeitungszeit

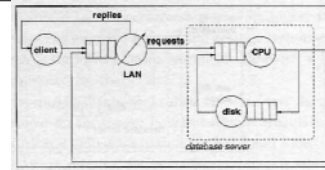


31.08.2004

Performance by Design

31

QN: Ressourcentypen Beispiel



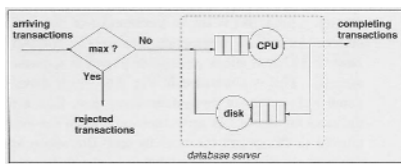
- Client (Delay):
 - entweder Warten auf Serverantwort oder Eingabe neuer Befehle
 - keine Warteschlange, einfache Verzögerung
- LAN (Load dependent):
 - Abnahme der effektiven Bandbreite bei Zunahme der Clientanfragen aufgrund von Paketkollisionen (Servicerate hängt von der Auslastung ab)
- CPU/Disk (Load independent):
 - konstante Servicerate

31.08.2004

Performance by Design

32

QN: Blocking



- zur Garantierung von Antwortzeiten: Begrenzung der maximalen Anzahl von Transaktionen
- Durchsatz = Ankunftsrate * (1 – Wahrscheinlichkeit der Ablehnung)

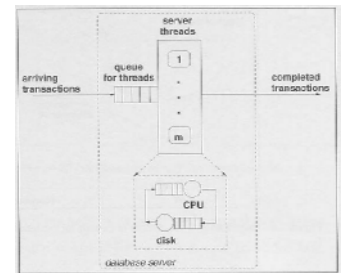
31.08.2004

Performance by Design

33

QN: Software Contention

- Software Contention: Konkurrenz um begrenzte Anzahl an Threads
- Physical Contention: Konkurrenz um begrenzte Anzahl an Ressourcen

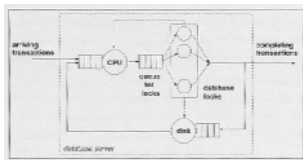


31.08.2004

Performance by Design

34

QN: gleichzeitige Ressourcennutzung



- Datenbank-Lock führt zu gleichzeitiger Ressourcennutzung (hier: CPU und Disk)
- Wahrscheinlichkeit, dass ein Lock durch eine andere Transaktion gehalten wird, erhöht sich mit der Systemlast

31.08.2004

Performance by Design

35

QN: Scheduling Strategien

- First Come First Serve
- Priority Queue
- Round Robin
- Processor Sharing
- Shortest Job First
- Random
- ...

31.08.2004

Performance by Design

36

QN: formale Definition

- K: Anzahl der Queues
- R: Anzahl der Systemlastklassen
- Eingabeparameter:
 - Lastintensität
 - offene Klassen: $\lambda = (\lambda_1, \dots, \lambda_R)$ (Ankunftsrate)
 - geschlossene Klassen: $N = (N_1, \dots, N_R)$ (Anfragen im System)
 - Rechenzeit $D_{i,r}$ (Queue i, Klasse r)

31.08.2004

Performance by Design

37

Queueing Networks: quantitative Aspekte

- Grundlegende Performance Ergebnisse
- Utilization Law
- Service Demand Law
- Forced Flow Law
- Little's Law
- Interactive Response Time Law
- Performance-Grenzen

31.08.2004

Performance by Design

38

QN: Grundlagen

- Beispiel: Eine CPU wird für eine Minute beobachtet. Während dieser Zeit finden 1800 Transaktionen statt und die CPU rechnet für 36 Sekunden. Alle Transaktionen werden innerhalb einer Minute fertig gestellt.
- Was ist die Performance des Systems?
 - Mittlere Bearbeitungszeiten pro Transaktion
 - Auslastung der Ressource
 - Durchsatz des Systems

31.08.2004

Performance by Design

39

QN: Grundlagen

	Beschreibung	Beispiel
T	Länge der Beobachtungszeit	60 sec
K	Anzahl der Ressourcen im System	1 (die CPU)
B_i	gesamte Rechenzeit der Ressource i in der Beobachtungszeit	36 sec
A_0	gesamte Anzahl der Anfragen an das System in der Beobachtungszeit	1800 Transaktionen
C_0	gesamte Anzahl der fertiggestellten Anfragen an das System in der Beobachtungszeit	1800 Transaktionen (job-flow-balance)

31.08.2004

Performance by Design

40

QN: Grundlagen

- S_i (= Service Time)
 - mittlere Bearbeitungszeit pro Anfrage an Ressource i
 - $S_i = B_i / C_i$
 - Beispiel: $S_1 = 36 / 1800 = 1 / 50$ sec pro Transaktion
- U_i (= Utilization)
 - Auslastung der Ressource i
 - $U_i = B_i / T$
 - Beispiel: $U_1 = 36 / 60 = 60\%$

31.08.2004

Performance by Design

41

QN: Grundlagen

- λ_i (= Arrival Rate)
 - Ankunftsrate pro Zeiteinheit an Ressource i
 - $\lambda_i = A_i / T$
 - Beispiel: $\lambda_1 = 1800 / 60 = 30$ tps
- X_0 (= Throughput)
 - Systemdurchsatz
 - $X_0 = C_0 / T$
 - Beispiel: $X_0 = 1800 / 60 = 30$ tps

31.08.2004

Performance by Design

42

QN: Utilization Law

$$U_i = S_i * X_i = S_i * \lambda_i$$

- Beispiel: Die Bandbreite eines Modems beträgt 56000 bps, es werden pro Sekunde 3 Pakete (1500 Byte) übertragen. Wie hoch ist die Auslastung?
- Mittlere Übertragungszeit $S_i = (1500 * 8) / 56000 = 0,214 \text{ sec / Paket}$
- Auslastung $U_i = 0,214 * 3 = 0,642 = 64,2 \%$

31.08.2004

Performance by Design

43

QN: Service Demand Law

$$D_i = U_i / X_0 = V_i * S_i$$

- Beispiel: Ein Web-Server wird für 10 Minuten beobachtet und ist zu 90% ausgelastet. Laut Web-Server-Log wurden in dieser Zeit 30000 Anfragen verarbeitet. Wie hoch ist der CPU-Bedarf einer Anfrage?
- Durchsatz des Servers $X_0 = 30000 / 600 = 50 \text{ Anfragen / sec}$
- CPU-Bedarf $D_i = 0,9 / 50 = 0,018 \text{ sec / Anfrage}$

31.08.2004

Performance by Design

44

QN: Forced Flow Law

$$X_i = V_i * X_0$$

- V_i : Anzahl der Besuche einer Anfrage auf einer Ressource
- Beispiel: Während einer einstündigen Beobachtung eines Datenbank-Servers werden 7200 Transaktionen durchgeführt, die jeweils 4,5 I/O Operationen brauchen. Wie hoch ist der Durchsatz der Festplatte?
- Durchsatz des Servers $X_0 = 7200 / 3600 = 2 \text{ tps}$
- Durchsatz der Festplatte $X_i = 4,5 * 2 = 9 \text{ tps}$

31.08.2004

Performance by Design

45

QN: Little's Law

$$N = R * X$$

- N: Länge der Warteschlange
- Beispiel: Ein NFS-Server wurde 30 Minuten lang beobachtet und es wurden 32400 I/O Anfragen durchgeführt. Bei durchschnittlich 9 offenen Anfragen (N_{req}), wie hoch war die Antwortzeit für eine Anfrage?
- $X_{\text{Server}} = 32400 / 1800 = 18 \text{ Anfragen / sec}$
- $R_{\text{req}} = N_{\text{req}} / X_{\text{Server}} = 9 / 18 = 0,5 \text{ sec}$

31.08.2004

Performance by Design

46

QN: Interactive Response Time Law

$$R = (M / X_0) - Z$$

- M: Anzahl der Clients, die eine Ressource gleichzeitig bearbeiten
- Z: think time der Clients
- Beispiel: Wenn 7200 Anfragen von 40 Clients während einer Stunde bearbeitet werden und die durchschnittliche think time 15 Sekunden beträgt, wie hoch ist die Antwortzeit?
- $R = (40 / (7200/3600)) - 15 = 5 \text{ sec}$

31.08.2004

Performance by Design

47

Beispiel: Datenbank-Server

31.08.2004

Performance by Design

48

Beispiel

- Analyse der Systemlast
- Cluster-Einteilung der Systemlast
- Erstellen eines Performance-Modells
- Berechnung der Performance-Daten
- Testen von Entwurfsalternativen
 - Hinzufügen einer dritten Festplatte
 - Nutzung eines Mehrprozessorsystems
 - Nutzung schnellerer Festplatten

31.08.2004

Performance by Design

49

Analyse der Systemlast

	CPU	Disk 1	Disk 2	TR ID
1	116.824	9	9	18
2	64.383	7	9	37
3	35.403	7	9	58
4	104.409	8	12	77
5	119.793	9	8	19
6	47.956	5	7	1
7	72.453	9	8	99
8	33.800	6	15	121
9	70.900	6	14	38
10	86.329	6	13	2
11	39.710	5	13	139
12	76.018	6	11	20
13	59.795	7	10	100
14	58.789	9	12	3
15	76.889	7	12	78
...	...	...	...	...

- Log vom DBMS Performance Monitor
 - 200 Transaktionen innerhalb von 150 Sekunden
- Durchsatz
 - $X_0 = 200 / 150 = 1,33$ tps

31.08.2004

Performance by Design

50

Analyse der Systemlast

- Messung der Auslastung der Ressourcen mit OS Performance Monitor

Ressource	Auslastung (%)
CPU	45
Disk 1	75
Disk 2	65

31.08.2004

Performance by Design

51

Analyse der Systemlast

	CPU Time (msec)	No. I/Os Disk 1	No. I/Os Disk 2
Mean	238.2	51.38	44.85
Standard Deviation	165.9	27.0	26.4
Sample Variance	27510.4	728.7	698.1
Coeff. of Variation	0.696	0.525	0.677
Minimum	23.6	5	7
First Quartile (Q1)	104.4	33	26
Median (Q2)	151.6	63	39
Third Quartile (Q3)	418.1	72	68
Maximum	507.5	85	92
Range	483.9	80	85
Largest	507.5	85	92
Smallest	23.60	5	7
Sum	47640.8	10275	8969

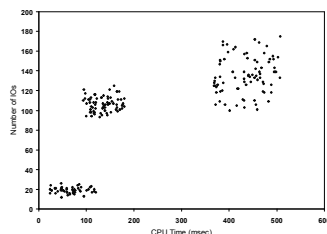
- Berechnungen z.B. mit Excel
- Coefficient of Variation
- CV = Standard Deviation / Mean
 - relatives Maß der Variabilität
 - Faustregel: Wenn $CV < 0,25$ gehören die Anfragen in eine Lastklasse

31.08.2004

Performance by Design

52

Analyse der Systemlast



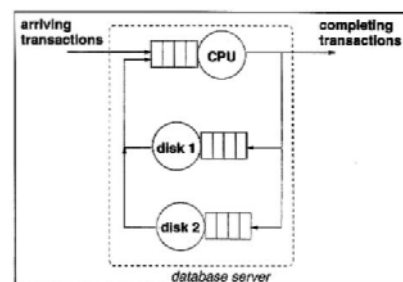
Cluster Number	CPU Time	I/Os on disk 1	I/Os on disk 2	No. of points
1	67.5	8.0	11.0	50
2	434.2	62.4	73.1	80
3	136.1	69.8	36.7	70

31.08.2004

Performance by Design

53

Erstellung eines Performance Modells



31.08.2004

Performance by Design

54

Erstellen eines Performance Modells



- Gegeben:
 - Ankunftsrate: 1,33 tps (vom DBMS Performance Monitor)
 - Auslastung (vom OS Performance Monitor)
 - Durchsatz (= Ankunftsrate)
 - Einteilung der Systemlast in drei Klassen
- Gesucht für das QN:
 - Rechenzeiten für jede Klasse (Service Demands)

31.08.2004

Performance by Design

55

Erstellen eines Performance Modells



- Service Demand Law
 - $D_{i,r} = U_{i,r} / X_0$
 - (i ist die Ressource, r die Lastklasse)
- Berechnung der Auslastung für die einzelnen Lastklassen:
 - $U_{i,r} = U_i * f_{i,r}$
 - $f_{i,r}$ ist ein Faktor, der von Ressourcentyp und Lastklasse abhängt

31.08.2004

Performance by Design

56

Erstellen eines Performance Modells



- Faktoren für die CPU
 - $f_{CPU,r}$ = Gesamte CPU-Zeit für Klasse r / Gesamte CPU Zeit für alle Klassen
 - $f_{CPU,1} = (67,5 * 50) / 47640,8 = 0,071$
 - $f_{CPU,2} = (434,2 * 80) / 47640,8 = 0,729$
 - $f_{CPU,3} = (136,1 * 70) / 47640,8 = 0,200$

31.08.2004

Performance by Design

57

Erstellen eines Performance Modells



- Faktoren für Disk1
 - $f_{disk i,r}$ = Gesamte Anzahl an I/Os für Klasse r / Gesamte Anzahl an I/Os für alle Klassen
 - $f_{disk 1,1} = (8,0 * 50) / 10275 = 0,039$
 - $f_{disk 1,2} = (62,4 * 80) / 10275 = 0,486$
 - $f_{disk 1,3} = (69,8 * 70) / 10275 = 0,475$

31.08.2004

Performance by Design

58

Erstellen eines Performance Modells



- Faktoren für Disk2
 - $f_{disk i,r}$ = Gesamte Anzahl an I/Os für Klasse r / Gesamte Anzahl an I/Os für alle Klassen
 - $f_{disk 2,1} = (11,0 * 50) / 8969 = 0,061$
 - $f_{disk 2,2} = (73,1 * 80) / 8969 = 0,652$
 - $f_{disk 2,3} = (36,7 * 70) / 8969 = 0,287$

31.08.2004

Performance by Design

59

Erstellen eines Performance Modells



- Auslastung einer Lastklasse: $U_{i,r} = U_i * f_{i,r}$
- $U_{CPU,1} = U_{CPU} * f_{CPU,1} = 0,45 * 0,071 = 0,032$
- $U_{CPU,2} = U_{CPU} * f_{CPU,2} = 0,45 * 0,729 = 0,328$
- $U_{CPU,3} = U_{CPU} * f_{CPU,3} = 0,45 * 0,200 = 0,090$
- analog für Disk1 und Disk2

31.08.2004

Performance by Design

60

Erstellen eines Performance Modells

- Auslastungen für alle Ressourcen und Klassen ($U_{i,r}$)

	Class 1	Class 2	Class 3	Total
CPU	0,032	0,328	0,090	0,45
Disk 1	0,029	0,365	0,356	0,75
Disk 2	0,040	0,424	0,187	0,65

- Rechenzeiten (in Sekunden)

	Class 1	Class 2	Class 3
CPU	0,096	0,615	0,193
Disk 1	0,088	0,683	0,763
Disk 2	0,119	0,795	0,400

- $D_{i,r} = U_{i,r} / X_0$

31.08.2004

Performance by Design

61

Erstellen eines Performance Modells

- alle Eingabeparameter vorhanden
- Eingabe in Spreadsheet
- Berechnung der Antwortzeiten für das System

	Class 1	Class 2	Class 3
CPU	0,17427	1,1835	0,35065
Disk 1	0,35100	2,73375	3,0535
Disk 2	0,33986	2,27036	1,14214
Response Time	0,86513	6,12246	4,54636

- Disk1 ist 'Bottleneck'

31.08.2004

Performance by Design

62

Testen von Entwurfsalternativen

- Vorhersage der Systemlast-Entwicklung:

	Month	Arrival Rate (tps)
1	January	1,33
2	February	1,45
3	March	1,68
4	April	2,26
5	May	2,68
6	June	3,25
7	July	3,98
8	August	4,78
9	September	5,74
10	October	6,76

31.08.2004

Performance by Design

63

Testen von Entwurfsalternativen

- Hinzufügen einer dritten Festplatte
- Load Balancing
- $D_{neu\ i,r} = (D_{alt\ 1,r} + D_{alt\ 2,r}) / 3$ für $i = 1,2,3$

Service Demand	Class 1	Class 2	Class 3
CPU	0,096	0,615	0,193
Disk 1	0,069	0,493	0,388
Disk 2	0,069	0,493	0,388
Disk 3	0,069	0,493	0,388

31.08.2004

Performance by Design

64

Testen von Entwurfsalternativen

- neue Antwortzeiten:

Arrival Rate (tps)	R1	R2	R3
1,33	0,56	3,89	2,53
1,45	0,61	4,21	2,75
1,68	0,73	5,02	3,28
2,26	1,39	9,65	6,37
2,68	4,34	30,28	20,78

31.08.2004

Performance by Design

65

Weitere Alternativen

Arrival Rate (tps)	Response Time (sec)		
	Class 1	Class 2	Class 3
Scenario: third disk added.			
1,33	0,56	3,89	2,53
1,45	0,61	4,21	2,75
1,68	0,73	5,02	3,28
2,26	1,39	9,65	6,37
2,68	4,34	30,28	20,78
Scenario: 3 disks and a dual CPU.			
2,26	1,11	7,86	5,81
2,68	3,47	24,71	19,08
Scenario: 3 disks 3x faster and a dual CPU.			
2,68	0,24	1,59	0,84
3,25	0,27	1,78	0,93
3,98	0,32	2,16	1,11
4,78	0,45	3,01	1,47
5,74	1,78	11,60	4,34
Scenario: 3 disks 3x faster and a quad CPU.			
5,74	0,35	2,25	1,41
6,76	0,45	3,15	2,09

31.08.2004

Performance by Design

66

Fazit: Lernziele der Übung



- Verstehen wie das Vorgehensmodell aufgebaut ist
- Wissen, wie man Workloads in Klassen einteilt
- Wissen, wie man die Eingabeparameter für ein Queueing Network berechnet

Simulation von UML-Modellen

Balsamo/Marzolla

Inhalt

- Allgemeines
- Vorgehensmodell
- Attributierung von UML-Modellen
- Simulationen von UML-Modellen mit umIPSI
- Ergebnisse der Simulation
- Beispiel

31.08.2004

umIPSI

2

Allgemeines

- Ziel:
 - nahe Integration von Performance-Analysen in den Entwicklungsprozess
- Verwendung von Standard-Notationen zur Modellierung
 - UML (1997)
 - UML Performance Profile (2002)
- umIPSI-Tool
 - Thema einer Dissertation Februar 2004
 - Attributierung von UML-Diagrammen mit anschließender automatischer **Simulation**

UML- Ψ

31.08.2004

umIPSI

3

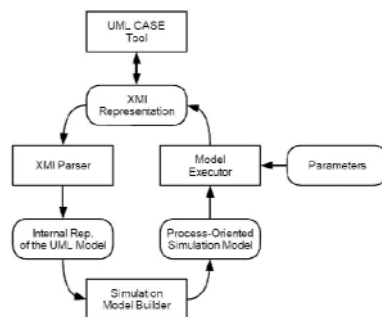
Vorgehensmodell

31.08.2004

umIPSI

4

Vorgehensmodell



31.08.2004

umIPSI

5

Vorgehensmodell



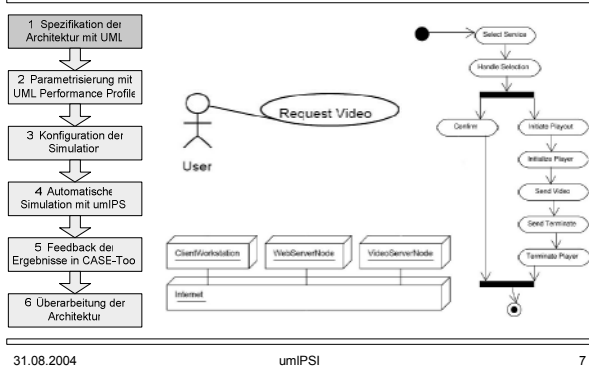
- Beispiel: Modellierung eines Videostreams im Internet

31.08.2004

umIPSI

6

Vorgehensmodell

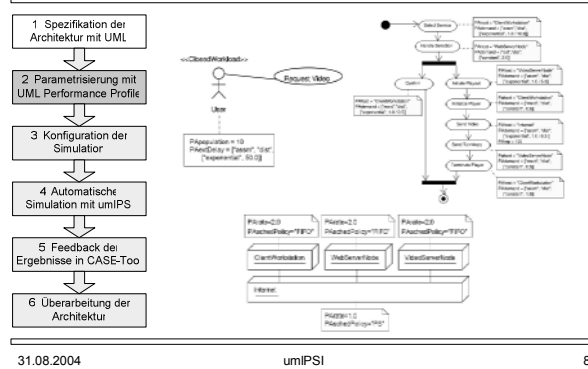


31.08.2004

umlPSI

7

Vorgehensmodell

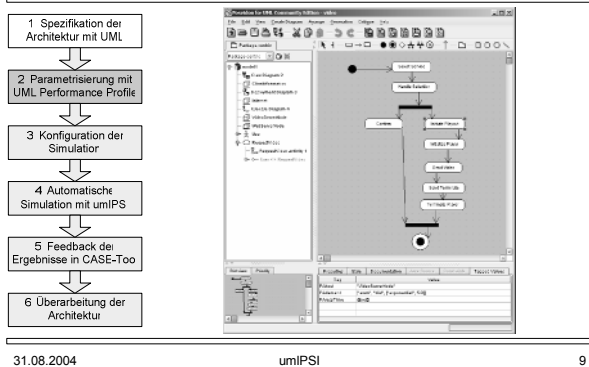


31.08.2004

umlPSI

8

Vorgehensmodell

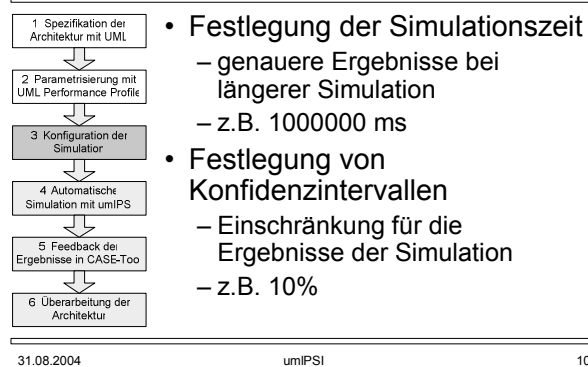


31.08.2004

umlPSI

9

Vorgehensmodell

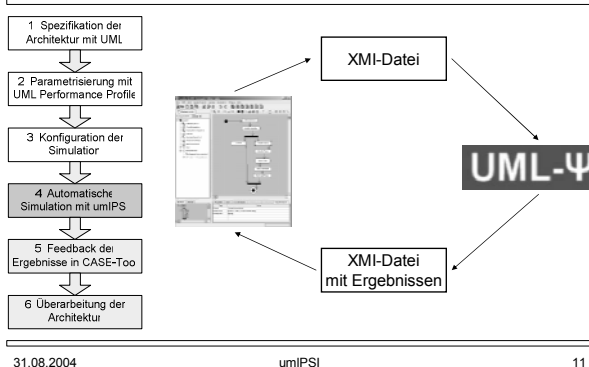


31.08.2004

umlPSI

10

Vorgehensmodell

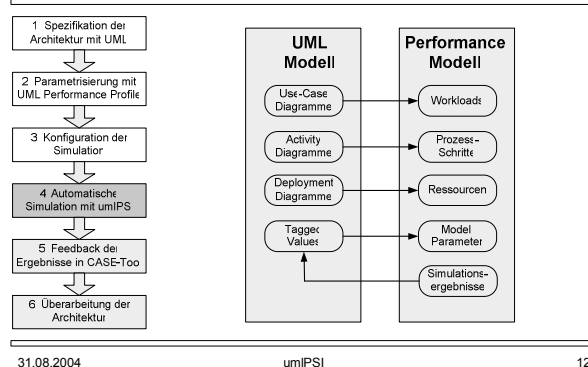


31.08.2004

umlPSI

11

Vorgehensmodell

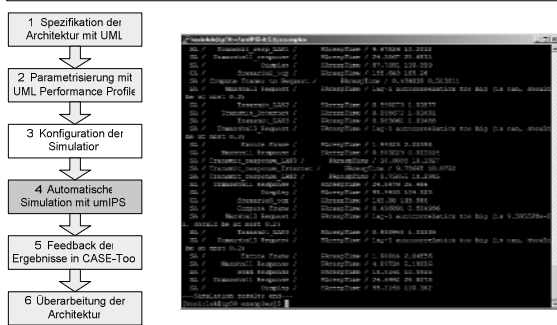


31.08.2004

umlPSI

12

Vorgehensmodell

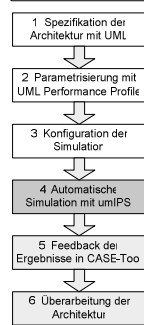


31.08.2004

umlPSI

13

Vorgehensmodell



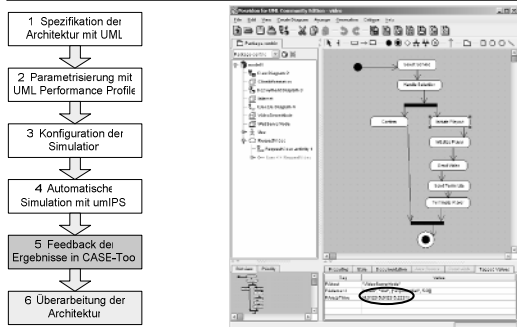
- **Ergebnisse:**
 - Auslastungen von Ressourcen
 - Durchsatz von Ressourcen
 - Mittlere Ausführungszeiten für Aktionen und Use-Cases
- spezielle Attribute zur Speicherung der Ergebnisse im UML-Modell

31.08.2004

umlPSI

14

Vorgehensmodell

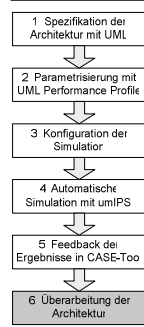


31.08.2004

umlPSI

15

Vorgehensmodell



- **Ergebnisse:**

Utilization	Throughput (requests/sec)
ClientWorkstation	
[0.183559, 0.183619]	[0.0611551, 0.0611941]
WebServerNode	
[0.0152827, 0.0153439]	[0.0152827, 0.0153439]
VideoServerNode	
[0.0432813, 0.0433357]	[0.030573, 0.0305853]
Internet	
[0.764314, 0.764321]	[1.5285, 1.52852]

- Internet-Ressource am stärksten belastet (~76%)
- Testen verschiedener Entwurfsalternativen

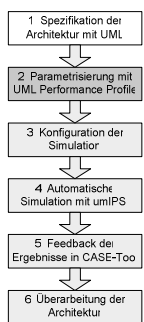
31.08.2004

umlPSI

16

Attributierung von UML-Diagrammen

UML Performance Profile



31.08.2004

umlPSI

17

Erweiterungsmechanismen für UML

- **Stereotypes**
 - Definition von Subklassen für existierenden Metamodell-Elemente
 - Gleiche Struktur wie Original, Erweiterungen
 - graphische Repräsentierung: <<stereotype>>
- **Tagged-Values**
 - Attribute für Modell-Elemente
 - Beispiel: PAAhost = „Workstation“
- **Constraints**
 - Einschränkungen für ein Attribut
 - Beispiel: PAArespTime < 1.5
- **Profile**
 - Paket mit speziell für eine Domäne angepassten Modell-Elementen
 - Definition von Stereotypes, Tagged Values und Constraints

31.08.2004

umlPSI

18

UML Performance Profile

- „UML Profile for Schedulability, Performance and Time specification“
 - Standardisierung März 2002 durch die OMG
 - Einheitliche Schnittstelle zwischen Software-Modell (in UML) und Performance-Modell
 - Möglichkeit zur Verwendung unterschiedlichster Performance Modelle wie Queueing Networks, Petri-Netze, Simulationsmodelle...
 - Vorbereitung von UML-Modellen zur toolgestützten Auswertung

31.08.2004

umIPSI

19

UML Performance Profile

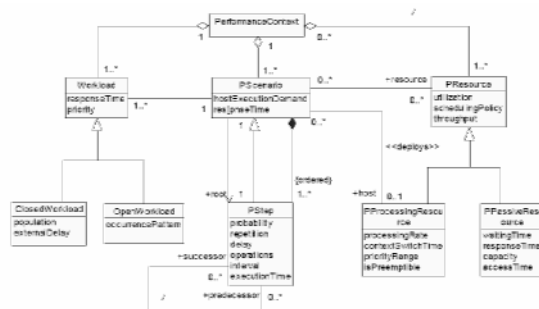
- Möglichkeit zur:
 - Aufnahme von Performance-Anforderungen in den Entwurfskontext
 - Assoziation von QoS-Charakteristiken mit ausgewählten Elemente eines UML-Modells
 - Spezifikation von Ausführungsparametern zur Nutzung durch Modellierungstools für die Berechnung von Performance-Charakteristiken
 - Präsentation von Performance-Ergebnissen, die durch Tools ermittelt wurden

31.08.2004

umIPSI

20

UML Performance Profile Domain Viewpoint



31.08.2004

umIPSI

21

umIPSI-Ansatz

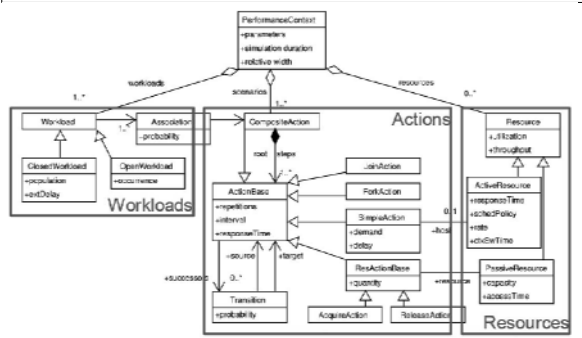
- Modifikationen des UML Performance Profiles zur leichteren Erstellung der Simulation
- Erweiterungen:
 - Composite Pattern für Actions
 - Zuweisungen von Wahrscheinlichkeiten an Transitionen
 - Join/Fork-Actions zur Modellierung von nebenläufigen Threads
 - Modellierung von Workloads mit Use-Case Diagrammen

31.08.2004

umIPSI

22

Performance-Modell



31.08.2004

umIPSI

23

Abbildung auf UML- Erweiterungen

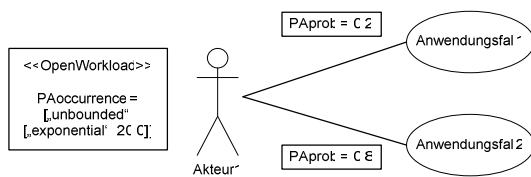
- Performance-Modell → UML Extensions
 - Klasse → Stereotype
 - Attribut → Tagged Values

31.08.2004

umIPSI

24

Modellierung von Workloads: offene Systemlastklassen



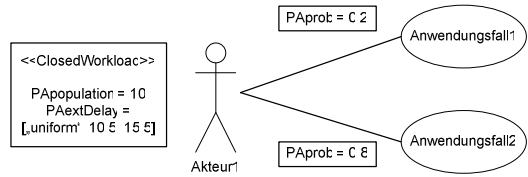
PAoccurrence	Ankunftsrate für Anfragen an das System
PAprob	Wahrscheinlichkeit, dass eine Transition genommen wird

31.08.2004

umlPSI

25

Modellierung von Workloads: geschlossene Systemlastklassen



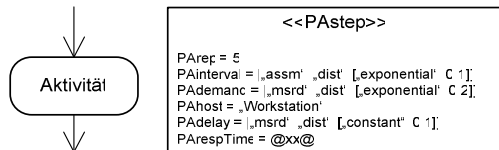
PApopulation	Anzahl der Anfragen im System
PAextDelay	externe Verzögerung zwischen zwei Anfragen (think time)
PAprob	Wahrscheinlichkeit, dass eine Transition genommen wird

31.08.2004

umlPSI

26

Modellierung von Szenarien: einfache Zustände



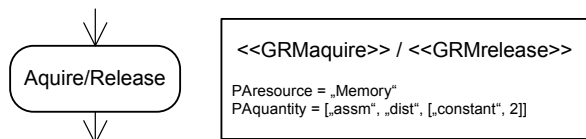
PArep	Anzahl der Wiederholungen
PAinterval	Intervallzeit zwischen zwei Wiederholungen
PAdemand	Service Demand: die benötigte Rechenzeit für diesen Schritt
PAhost	Ressource, auf der dieser Schritt ausgeführt wird (diese muss im Deployment-Diagramm enthalten sein)
PAdelay	Verzögerung (z.B. für eine Benutzerinteraktion)
Paresptime	Antwortzeit für diesen Schritt (berechnet umlPSI)

31.08.2004

umlPSI

27

Modellierung von Szenarien: Akquirierung von passiven Ressourcen



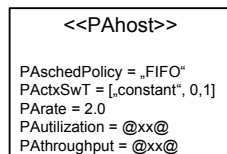
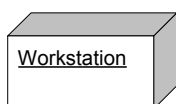
PAresource	Ressource, die besetzt oder freigegeben wird
PAquantity	Anzahl der Ressourcen, die besetzt oder freigegeben werden

31.08.2004

umlPSI

28

Modellierung von Ressourcen: Aktive Ressourcen



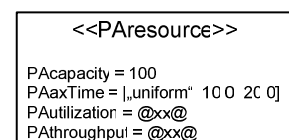
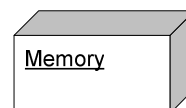
PASchedPolicy	Scheduling-Strategie: (FIFO, LIFO, PS(=Processor Sharing))
PActxSwT	Zeit für einen Kontextwechsel (nur falls Scheduling-Strategie PS)
PArate	Rechengeschwindigkeit im Verhältnis zum Referenz-Prozessor (benötigt 1 sec Rechenzeit für 1 sec Service Demand)
PAutilization	Auslastung der Ressource (berechnet umlPSI)
PAtthroughput	Durchsatz der Ressource (berechnet umlPSI)

31.08.2004

umlPSI

29

Modellierung von Ressourcen: Passive Ressourcen



PACapacity	maximale Anzahl von Instanzen
PAaxTime	physikalische Zugriffszeit, Zeit zwischen Besetzung der Ressource und ihrer Einsatzfähigkeit (Freigabe verbraucht keine Zeit)
PAutilization	Auslastung der Ressource (berechnet umlPSI)
PAtthroughput	Durchsatz der Ressource (berechnet umlPSI)

31.08.2004

umlPSI

30

Performance Context



- Konfiguration der Simulation im Root-Knoten des Modells

simDuration	Maximale Länge der Simulation in ms
paramFileName	Name einer externen Parameter-Datei (z.B. „parameter.pl“)
confRelWidth	relative width of confidence intervals (default: 90% confidence level)

31.08.2004

umlPSI

31

Tagged Values (1/3)



Tag	Type	Multiplicity	Domain Attribute Name
paramFileName	String	[0..1]	PerformanceContext:parameters
simDuration	Real	[0..1]	PerformanceContext:simulation duration
confRelWidth	Real	[0..1]	PerformanceContext:confidence width
PAprob	Real	1	Association:probability
PAprob	Real	1	Transition:probability
PAoccurrence	RTarrivalPattern	[0..1]	OpenWorkload:occurrence
PApopulation	Integer	[0..1]	ClosedWorkload:population
PAextDelay	PAperfValue	[0..1]	ClosedWorkload:extDelay

31.08.2004

umlPSI

32

Tagged Values (2/3)



Tag	Type	Multiplicity	Domain Attribute Name
PAutilization	mean	[0..1]	Resource::utilization
PAthroughput	mean	[0..1]	Resource::throughput
PAshedPolicy	Enumeration {LIFO, FIFO, PS}	[0..1]	ActiveResource::schedPolicy
PActxSwT	PAperfValue	[0..1]	ActiveResource:ctxSwTime
PArate	Real	[0..1]	ActiveResource::rate
PAcapacity	Real	[0..1]	PassiveResource::capacity
PAaxTime	PAperfValue	[0..1]	PassiveResource::accessTime

31.08.2004

umlPSI

33

Tagged Values (3/3)



Tag	Type	Multiplicity	Domain Attribute Name
PArep	Integer	[0..1]	ActionBase::repetitions
PAinterval	PAperfValue	[0..1]	ActionBase::interval
PAdemand	PAperfValue	[0..1]	SimpleAction::demand
PAhost	String	[0..1]	SimpleAction::host
PAdelay	PAperfValue	[0..1]	SimpleAction::delay
PArespTime	mean	[0..1]	ActionBase::responseTime
PAresource	String	1	ResActionBase::resource
PAquantity	PAperfValue	[0..1]	ResActionBase::quantity

31.08.2004

umlPSI

34

Tagged Value Types



- PAperfValue:

< PAperfValue > := [< SourceModifier > „dist“, < PDFstring >]
 < SourceModifier > := „assm“ | „pred“ | „msrd“

< PDFstring > := [< constantPDF > | < uniformPDF > | < exponentialPDF > | < normalPDF >]

< constantPDF > := < real > | „constant“, < real >
 < uniformPDF > := „uniform“, < real >, < real >
 < exponentialPDF > := „exponential“, < real >
 < normalPDF > := „normal“, < real >, < real >

31.08.2004

umlPSI

35

Tagged Value Types



- PAperfValue:

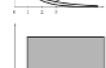
– konstanter Wert:
 • < constantPDF > := < real > | „constant“, < real >



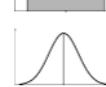
– Exponentialverteilung mit Mittelwert:
 • < exponentialPDF > := „exponential“, < mean >



– Gleichverteilung über Intervall [a,b]:
 • < uniformPDF > := „uniform“, < a >, < b >



– Normalverteilung mit Mittelwert und Standardabweichung:
 • < normalPDF > := „normal“, < mean >, < stddev >



31.08.2004

umlPSI

36

Tagged Value Types



- PArpfValue Beispiele:
 - PAdemand = [„assm“, „dist“, [„exponential“, 2.5]]
 - (angenommene Exponentialverteilung um den Mittelwert 2,5 Sekunden)
 - PActxSwT = [„msrd“, „dist“, [„constant“, 0.1]]
 - (gemessener konstanter Wert von 0.1 Sekunden)
 - PAdelay = [„pred“, „dist“, [„uniform“, 0.2, 0.5]]
 - (vorhergesagter, über dem Intervall 0,2-0,5 Sekunden gleichverteilter Wert)

31.08.2004

umIPSI

37

Tagged Value Types



• RTarrivalPattern:

< RTarrivalPattern > := [< bounded > | < unbounded > | < bursty >]

< bounded > := "bounded", < real >, < real >
 < bursty > := "bursty", < PDFstring >, < integer >
 < unbounded > := "unbounded", < PDFstring >

- Bounded: Zeit zwischen zwei aufeinanderfolgenden Anfragen ist gleichverteilt über einem Intervall
 - 1. Parameter: obere Grenze
 - 2. Parameter: untere Grenze

31.08.2004

umIPSI

38

Tagged Value Types



• RTarrivalPattern:

< RTarrivalPattern > := [< bounded > | < unbounded > | < bursty >]
 < bounded > := "bounded", < real >, < real >
 < bursty > := "bursty", < PDFstring >, < integer >
 < unbounded > := "unbounded", < PDFstring >

- Bursty: Anfragen kommen in Schüben an.
 - 1. Parameter: mittlere Zeit zwischen zwei Schüben
 - 2. Parameter: maximale Größe eines Schubs
 - Berechnung der eigentlich Schubgröße als eine gleichverteilte Zufallsvariable über dem Intervall [0...maximale Schubgröße]
- Unbounded: Zeit zwischen zwei aufeinanderfolgenden Anfragen mit entsprechender Verteilungsfunktion
 - 1. Parameter: Zeit zwischen zwei Anfragen

31.08.2004

umIPSI

39

Tagged Value Types



• RTarrivalPattern Beispiele:

- PAoccurrence = [„unbounded“, [„exponential“, 0.15]]
 - Intervall zwischen zwei Anfragen 0,15 Sekunden

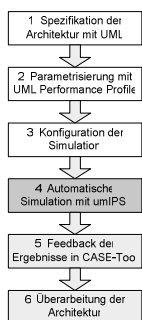
31.08.2004

umIPSI

40

Simulation von UML-Modellen

umIPSI



31.08.2004

umIPSI

41

Vergleich von Performance-Modellen



- analytisch
 - Menge von Formeln oder Algorithmen
 - Berechnung von Performance-Werten als eine Funktion der Eingabeparameter
 - begrenzter Detaillierungsgrad aufgrund der mathematischen Komplexität
 - effiziente Ausführung
 - einfache, billige Konstruktion
- simulationsbasiert
 - Emulation von dynamischen Aspekten eines Systems durch Computerprogramme
 - beliebiger Detaillierungsgrad
 - meist hoher Rechenaufwand für die Ausführung
 - Ausgabe: statistische Werte (mehrere Ausführungen notwendig)

31.08.2004

umIPSI

42

Beispiel

31.08.2004

umIPSI

49

NICE

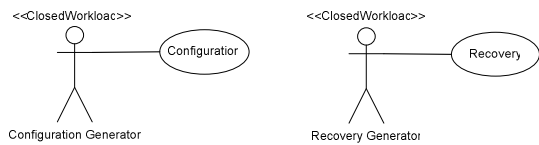
- Naval Integrated Communication Environment (NICE)
- Sprach-, Daten- und Videokommunikation

31.08.2004

umIPSI

50

Use-Cases



- neue Parameter-Einstellung für eine oder mehrere Equipments
- Performance-Vorgabe: mittlere Ausführungszeit < 5 sec ± 1sec
- Systemwiederherstellung nach dem Ausfall eines Equipments
- Performance-Vorgabe: mittlere Ausführungszeit < 5 sec ± 1sec

31.08.2004

umIPSI

51

Szenarien



Configuration Scenario

Recovery Scenario

31.08.2004

umIPSI

52

Mittlere Ausführungszeiten

Configuration Scenario:

Action	Mean Execution time (sec.)
PrepareSetParameter	1.00
SendSetParameter	0.01
SetParameter	2.00
GetParameter	0.01
TrapSetParameter	0.01
StopNewConfiguration	1.00

Recovery Scenario:

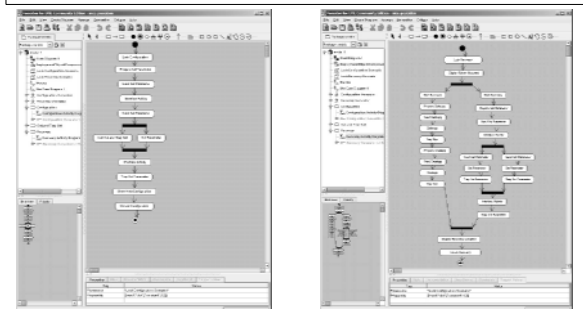
Action	Mean Execution time (sec.)
Display Failure Occurred	0.00
Start Recovery	0.00
PrepareSetParameter	1.00
SendSetParameter	0.01
SetParameter	2.00
GetParameter	0.01
TrapSetParameter	0.01
TrapCreatepp	0.01
TrapDeletepp	0.01
PrepareDeletepp	1.00
SendDeletepp	0.01
Deletepp	1.00
PrepareCreatepp	1.00
SendCreatepp	0.01
Createpp	2.00
Recovery Completed	0.00
Display Recovery Completed	0.50

31.08.2004

umIPSI

53

Aktivitätsdiagramme



Configuration Scenario

Recovery Scenario

31.08.2004

umIPSI

54

Ergebnisse der Simulation



N	Configuration Scenario		Recovery Scenario		Simulation Exec. Time
	Mean Exec. Time (s)	Requirement Satisfied?	Mean Exec. Time (s)	Requirement Satisfied?	
1	4.02	yes	6.03	no	36s
2	4.68	yes	7.04	no	1m09s
3	5.50	yes	10.26	no	1m37s
4	6.87	no	11.70	no	1m29s
5	8.35	no	17.60	no	2m44s
6	10.54	no	23.97	no	2m51s

- N: Anzahl der Equipments

31.08.2004

umlPSI

55

Fazit: Lernziele



- Verstehen, wie das Vorgehensmodell aufgebaut ist
- Wissen, wie man UML-Diagramme mit Performance-Attributen versieht
- Verstehen des Unterschiedes zwischen einem analytischen und simulationsbasierten Verfahren

31.08.2004

umlPSI

56

C. Übungszettel

Die folgenden Übungszettel wurden jeweils nach der Vorstellung eines Verfahrens ausgegeben, damit die Teilnehmer die Verfahren auch praktisch anwendeten und erste Erfahrungen mit den Werkzeugen sammelten. Die Studenten hatten jeweils eine Woche Zeit für die Bearbeitung der Aufgaben.

Übungsblatt 1

Software Performance Engineering

Hinweise:

- Für diese Übung wird ein Windows-Rechner benötigt. Das SPE-ED-Tool ist für andere Betriebssysteme leider nicht verfügbar.
- Abgabe der Lösung per Mail (s.u.) bis zum 22.06.
- Die Übung soll als Einzelperson und nicht in Gruppen bearbeitet werden.

Aufgabe 1 (Einarbeitung)

- Installieren Sie das SPE-ED-Tool (kann von Stud-IP heruntergeladen werden).
- Arbeiten Sie die „Quick-Start-Anleitung“ gründlich durch.

Aufgabe 2 (Beispielarchitektur)

Das folgende Sequenzdiagramm modelliert die Recherche in einem Bibliotheksverzeichnis. Der Benutzer kann Suchanfragen stellen, die an das lokale DBMS weitergeleitet werden. Alternativ kann über das Internet die Zentraldatenbank eines Verbundes mehrerer anderer Bibliotheken abgefragt werden. Der Web-Server und das lokale DBMS werden gemeinsam auf einem Rechner mit einer CPU und einer Festplatten ausgeführt.

- Legen Sie in SPE-ED ein neues Projekt an, das Sie nach ihrem eigenen Namen benennen („Markus Mustermann“). Erstellen Sie darin ein Szenario „bookSearch“.
- Erstellen Sie für das Sequenzdiagramm einen Execution-Graphen (Synchronisation kann zunächst vernachlässigt werden). Beachten Sie, dass in der Demo-Version von SPE-ED nur maximal 4 Subgraphen erstellt werden können. Gehen Sie davon aus, dass die Suche nach einem Buchtitel 10 mal wiederholt wird (Schleife im Sequenzdiagramm), bevor der Benutzer „viewDetails“ aufruft.
- Bestimmen Sie die Software Resource Requirements. Wählen Sie dazu den Menüpunkt (Software Model / Assign Template Spec) aus und weisen Sie aus den Beispiel-Projekten (Button „Show All“) die Vorlage „Distr sys spec“ zu.
- Tragen Sie nun für jeden Schritt im Execution Graph die jeweiligen geschätzten Werte ein (Software Model / Specify values). Versuchen Sie möglichst realistische Werte anzugeben und modifizieren Sie dazu diese Werte ggf. nachträglich.
- Weisen Sie die Computer Resource Requirements zu. Wählen Sie dazu den Menüpunkt (Overhead / Facility template) aus und weisen Sie aus den Beispiel-Projekten (Button „Show All“) die Facility „WebServer“ zu.

- Tragen Sie folgende Werte ein:

Facility template:				
WebServer				
Devices	CPU	Disk	Delay	GLNet
Quantity	1	1	1	1
Service units	KInstr	I/Os	Visits	Msgs
WorkUnit	25			
DB	500	4		
Msgs	25	1		1
Delay			1	
Service time	1e-006	0.05	0.5	0.1

- Berechnen Sie die Antwortzeiten des Szenarios (No Contention-Solution).
- Spezifizieren Sie den Service-Level der Architektur (Software Execution Model / Service level...). Geben Sie als Performance-Ziel 120 Sekunden an und probieren Sie verschiedene Ankunftsrate zwischen 0-1 aus. Stellen Sie fest, bis zu welcher Ankunftsrate das Performance-Ziel eingehalten werden kann (Contention-Solution).
- In den Web-Server werden eine Dual-CPU und eine zweite (identische) Festplatte eingebaut. Wie verändert sich die Antwortzeit des Szenarios, wenn Sie die in der vorherigen Aufgabe ermittelte maximale Ankunftsrate beibehalten?
- Machen Sie Vorschläge zur Überarbeitung der Architektur.

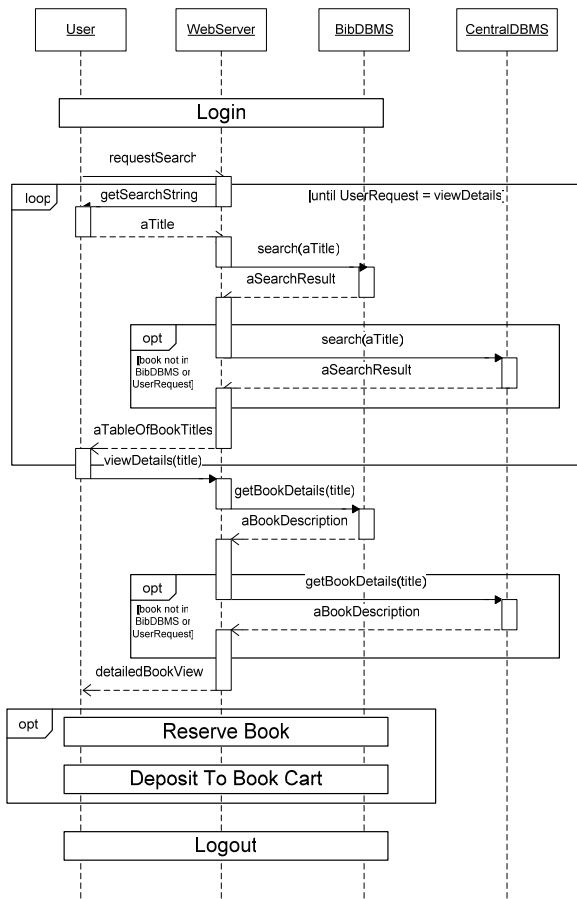
Zur Abgabe: Schreiben Sie eine E-Mail mit dem Betreff „SPE-Übung“ an

heiko.koziolek@informatik.uni-oldenburg.de

Geben Sie im Body der E-Mail an:

- die Antwortzeit des Szenarios (No Contention-Solution),
- die ermittelte maximale Ankunftsrate bei einem Performance-Ziel von 120 Sekunden,
- die Antwortzeit des aufgerüsteten Web-Servers bei der vorherigen maximalen Ankunftsrate und
- ihre Verbesserungsvorschläge für die Architektur.

Fügen Sie als Anhang die vier Dateien (spedb0, spedb1, spedb2, spedb3) hinzu, nachdem Sie Szenario und Projekt gespeichert haben. (In der Demo-Version von SPE-ED ist die Exportfunktionalität deaktiviert, so dass die ganze SPE-ED-Datenbank geschickt werden muss.)



Übungsblatt 2

Capacity Planning

Hinweise:

- Für die Übung wird ein Spreadsheet-Programm wie MS Excel, OpenOffice Calc oder ähnliches benötigt.
- Die benötigten Dateien können von Stud-IP bezogen werden.
- Tragen Sie ihre Ergebnisse in die Datei „Capacity_Planning.xls“ ein und hängen Sie diese an eine E-Mail an heiko.koziolek@informatik.uni-oldenburg.de
- Bitte bearbeiten Sie diesen Übungszettel als Einzelperson!
- Abgabetermin: 22.06.

Aufgabe 1) (Workload-Modellierung)

Die Excel-Datei „Capacity_Planning.xls“ enthält eine Liste von Transaktionen mit CPU-Zeiten und entsprechenden I/O-Werten.

- Berechnen Sie die Grundstatistiken, die auf dem Arbeitsblatt angegeben sind.
- Zeichnen Sie einen x-y scatter plot für diesen Datensatz und benutzen Sie dabei die CPU-Zeit für die X-Achse und die Anzahl der I/O-Operationen als y-Achse.
- Teilen Sie die Systemlast visuell in Cluster ein und geben Sie für jeden Cluster die Nummer, geschätzten Koordinaten des Mittelpunkts (CPU Time, I/Os) und Anzahl der Punkte an.

Aufgabe 2) (Berechnung der Eingabeparameter für QN)

Nehmen Sie an, dass die Daten in „Capacity_Planning.xls“ innerhalb von 15 Sekunden aufgenommen wurden, dass die CPU-Auslastung 35% beträgt und die Auslastung der Festplatte 80%. Jeder Cluster, der in Aufgabe 1 gefunden wurde gehört zu einer Klasse in einem QN-Modell.

- Was ist der Gesamtdurchsatz des Systems in tps (maximal 4 Nachkommastellen, auf- bzw. abrunden)?
- Was ist der Durchsatz für jede Klasse in tps?
- Berechnen Sie die Service Demands jeweils für die CPU und die Festplatte.

Aufgabe 3) (Analyse des QN)

Benutzen Sie „OpenQN.xls“ um das QN-Modell für das vorherige Beispiel zu lösen und nehmen Sie dabei an, dass die Ankunftsraten genauso groß sind, wie die Durchsätze der Klassen in der vorherigen Aufgabe. Geben Sie die Verweilzeiten und Auslastungen an.

Übungsblatt 3

Simulation von UML-Modellen

Hinweise:

- Zur Performance-Analyse werden die Tools umlPSI und libcppsim (beide Unix only) benötigt (gibt es auch auf StudIP).
- <http://www.dsi.unive.it/~marzolla/software/libcppsim-0.2.2.tar.gz>
- <http://www.dsi.unive.it/~marzolla/software/umlPSI-0.5.1.tar.gz>
- Zur Modellierung muss Poseidon 1.41 verwendet werden, da umlPSI momentan nur XMI-Dateien parsen kann, die von dieser Version erzeugt wurden.
- <ftp://ftp.gentleware.org/1.4.1/PoseidonCE-1.4.1.zip> (Windows, Linux)
- <ftp://ftp.gentleware.org/1.4.1/PoseidonCE-1.4.1.dmg.tar.gz> (Mac)
- Poseidon 1.4.1 ist in der ARBI unter /usr/local/java/poseidon1.4.1 installiert und kann mit startPoseidon141 gestartet werden.
- Abgabe per E-Mail bis zum 22.06.

Aufgabe 1 (Modellierung der Architektur mit UML)

Im Folgenden soll das Szenario eines E-Commerce Systems modelliert werden. Erstellen Sie dazu mit den folgenden Angaben unter Poseidon 1.4 ein Use Case-Diagramm, für jeden der 4 Anwendungsfälle ein Aktivitätsdiagramm und ein Deployment Diagramm.

Das System besteht aus einem Web-Server, der über LAN mit einem Datenbank-Server verbunden ist. Der Web-Server ist multi-threaded und kann maximal 5 Threads gleichzeitig ausführen. Vor jeder Anfrage an das System muss ein Thread akquiriert werden. Falls alle Threads belegt sind, werden die Anfragen in einer FIFO-Reihenfolge in eine Warteschlange eingereiht. Web-Server, LAN und Datenbank-Server können als aktive Ressourcen modelliert werden, während der Thread-Pool innerhalb des Web-Servers als passive Ressource mit einer Kapazität äquivalent zur maximalen Threadanzahl (5) modelliert wird. Weitere Vorgaben müssen für die Ressourcen nicht gemacht werden.

Für dieses Szenario gibt es zwei Benutzerklassen: „User“ und „Service“. Für den Actor „User“ gibt es drei Anwendungsfälle: „Browse Product“ (Wahrscheinlichkeit: 80%), „Select Product“ (10%) und „Checkout Cart“ (10%). Diese Benutzerklasse kann als Open Workload modelliert werden, wobei alle 60 Sekunden eine Anfrage an das System gestellt wird. Für den Actor „Service“ gibt es einen Anwendungsfall („Purge Database“). Dieser Vorgang findet in Intervallen von jeweils einer halben Stunde (1800 Sekunden) statt und wird nur von einem festen Benutzer durchgeführt (Modellierung als Closed Workload).

Für alle Anwendungsfälle des Aktors „User“ muss zunächst ein Thread akquiriert (Action1), die Anfrage geparkt (Action2) und dann über das LAN eine Anfrage an den DB-Server gestellt werden (Action3). Im Fall „Browse Product“ kann nach der Übermittlung der Anfrage an den DB-Server bereits mit dem Formatieren der Seite auf dem Web-Server begonnen werden. Gleichzeitig läuft die Abfrage der Datenbank und die anschließende Übermittlung der Daten vom Datenbank-Server zum Web-Server. In einem weiteren Schritt wird nach Übermittlung der Daten das Formatieren

der Seite auf dem Web-Server beendet. Anschließend kann der belegte Thread wieder freigegeben werden.

Im Fall „Select Product“ wird nach Übermittlung der geparkten Anfrage an den DB-Server lediglich die Datenbank aktualisiert, danach wird das Ergebnis zurück an den Web-Server transferiert. Schließlich erfolgt die Freigabe des belegten Threads. Im Fall „Checkout Cart“ werden wieder ein Thread eingeholt, die Anfrage geparkt und die Daten zur Datenbank übertragen. Anschließend können die Bezahlmethode und die Bestelldaten gleichzeitig auf dem DB-Server kontrolliert werden (Validate Payment einmal, Validate Order für jeden Artikel). Die Ergebnisse werden wieder über das LAN zurück an den Web-Server geliefert, worauf im Anschluss der Thread freigegeben wird.

Für den Anwendungsfall „Purge Database“ des Aktors „Service“ muss kein Thread akquiriert werden, da dieser Vorgang nur auf dem DB-Server stattfindet. Zunächst wird die Datenbank für diesen Vorgang initialisiert, dann werden gleichzeitig die aktuellen Sessions invalidiert (Invalidate Sessions) und die Tabellen gesäubert (Empty Charts). Anschließend folgt noch ein Schritt Cleanup.

Aufgabe 2 (Attributierung der UML-Elemente mit Tagged-Values)

Nun sollen alle Elemente des UML-Modells mit Stereotypes bzw. Tagged-Values versehen werden, so dass sie von umlPSI analysiert werden können. Stereotypes können im unteren rechten Abschnitt von Poseidon unter „Properties“ definiert werden. Für Tagged Values gibt es ein eigenes Tab.

- Deployment Diagramm: Geben Sie jeweils an, ob es sich um eine aktive (PAhost) oder passive (PAresource) Ressource handelt. Fügen Sie Tagged Values für Auslastung (PAutilization) und Durchsatz (PAthroughput) jeweils mit dem Wert 0 ein.
- Use Case Diagramm: Geben Sie bei den Aktoren jeweils an, ob es sich um einen offenen (OpenWorkload) oder geschlossenen (ClosedWorkload) Workload handelt und wie hoch die Ankunftsrate bzw. die externe Verzögerungsrate ist. Legen Sie für die einzelnen Anwendungsfälle die Wahrscheinlichkeiten (PAprob) jeweils an den Transitionen fest.
- Activity Diagramme: Geben Sie für die Schritte jeweils an, ob es sich um einen GRMquire/GRMrelease oder PASTep handelt. Spezifizieren Sie für jede Aktivität, auf welcher Ressource sie stattfindet (PAhost, PAresource) wie lange sie dauert (PAdemand, hier bitte Schätzungen!) und ggf. wie oft sie wiederholt wird. Fügen Sie zusätzlich Tagged Values für die Antwortzeiten (PArespTime) jeweils mit dem Wert 0 ein.
- Legen Sie im Wurzelknoten des gesamten Modells die Simulationszeit auf 4000000 Zeiteinheiten fest (simulation = 4000000). Geben Sie als Konfidenzintervall 10% an (confrelwidth = 0.1).

Aufgabe 3 (Simulation mit umlPSI)

Nach der Modellierung des Systems mit Poseidon und Spezifikation der Performance-Werte über Tagged Values kann nun die Simulation mit umlPSI erfolgen.

- Extrahieren Sie aus Ihrer Poseidon Projektdatei (example.zargo) die gleichnamige XMI-Datei (unzip example.zargo example.xmi). Diese dient umlPSI als Eingabe.

- Starten Sie umlPSI mit (umlPSI example.xmi). Die Simulation kann eine Weile dauern, zwischenzeitlich wird nach jeweils 1000000 Zeiteinheiten abgefragt, ob die Simulation fortgesetzt werden soll. Bestätigen Sie dies mindestens dreimal.
- Verläuft die Simulation erfolgreich, so wird eine Ausgabedatei (example.xmi.result) erzeugt. In dieser Datei sind die Tagged Values für z.B. Auslastung oder Durchsatz mit den Ergebnissen der Simulation beschrieben. Benennen Sie diese Datei in example.xmi um, und integrieren Sie sie wieder in ihre Projektdatei (zip example.zargo example.xmi). Sie können sich nun die Ergebnisse unter Poseidon anschauen.
- Hängen Sie ihre Projektdatei an eine E-Mail und senden Sie sie mit dem Betreff „umlPSI-Übung“ bis zum 22.06. an heiko.koziolek@informatik.uni-oldenburg.de

ANHANG:

- Legende für Stereotypes

Stereotype	Base Class	Tagged Values
<<PAContext>>	Model	paramFileName simDuration confRelWidth
<<OpenWorkload>>	Actor	PAoccurrence
<<ClosedWorkload>>	Actor	PApopulation PAextDelay
<<PAhost>>	Node	PAutilization PAthroughput PASchedPolicy PActxSwT PArate
<<PAresource>>	Node	PAutilization PAthroughput PAaxTime
<<PAStep>>	ActionState	PArep PAinterval PAdemand PAhost PAdelay PArespTime
<<PACompositeStep>>	ActionState	PArep PAinterval PArespTime
<<GRMquire>>	ActionState	PAresource PAquantity
<<GRMrelease>>	ActionState	PAresource PAquantity

- Legende für Tagged Values

Tag	Type	Multiplicity
paramFileName	String	[0..1]
simDuration	Real	[0..1]

confRelWidth	Real	[0..1]
PAprob	Real	1
PAprob	Real	1
PAoccurrence	RTArrivalPattern	[0..1]
PApopulation	Integer	[0..1]
PAextDelay	PAperValue	[0..1]
PAutilization	mean	[0..1]
PAthroughput	mean	[0..1]
PASchedPolicy	Enumeration(LIFO, FIFO, PS)	[0..1]
PActxSwT	PAperValue	[0..1]
PArate	Real	[0..1]
PACapacity	Real	[0..1]
PAaxTime	PAperValue	[0..1]
PArep	Integer	[0..1]
PAinterval	PAperValue	[0..1]
PAdemand	PAperValue	[0..1]
PAhost	String	[0..1]
PAdelay	PAperValue	[0..1]
PArespTime	mean	[0..1]
PAresource	String	1
PAquantity	PAperValue	[0..1]

- Grammatiken von Tagged Value Types

PAperValue

< PAperValue > := [< SourceModifier > , "dist", < PDFstring >]
< SourceModifier > := "assm" | "pred" | "msrd"

RTArrivalPattern

< RTArrivalPattern > := [< bounded > | < unbounded > | < bursty >]
< bounded > := "bounded", < real > , < real >
< bursty > := "bursty", < PDFstring > , < integer >
< unbounded > := "unbounded", < PDFstring >

PDFstring

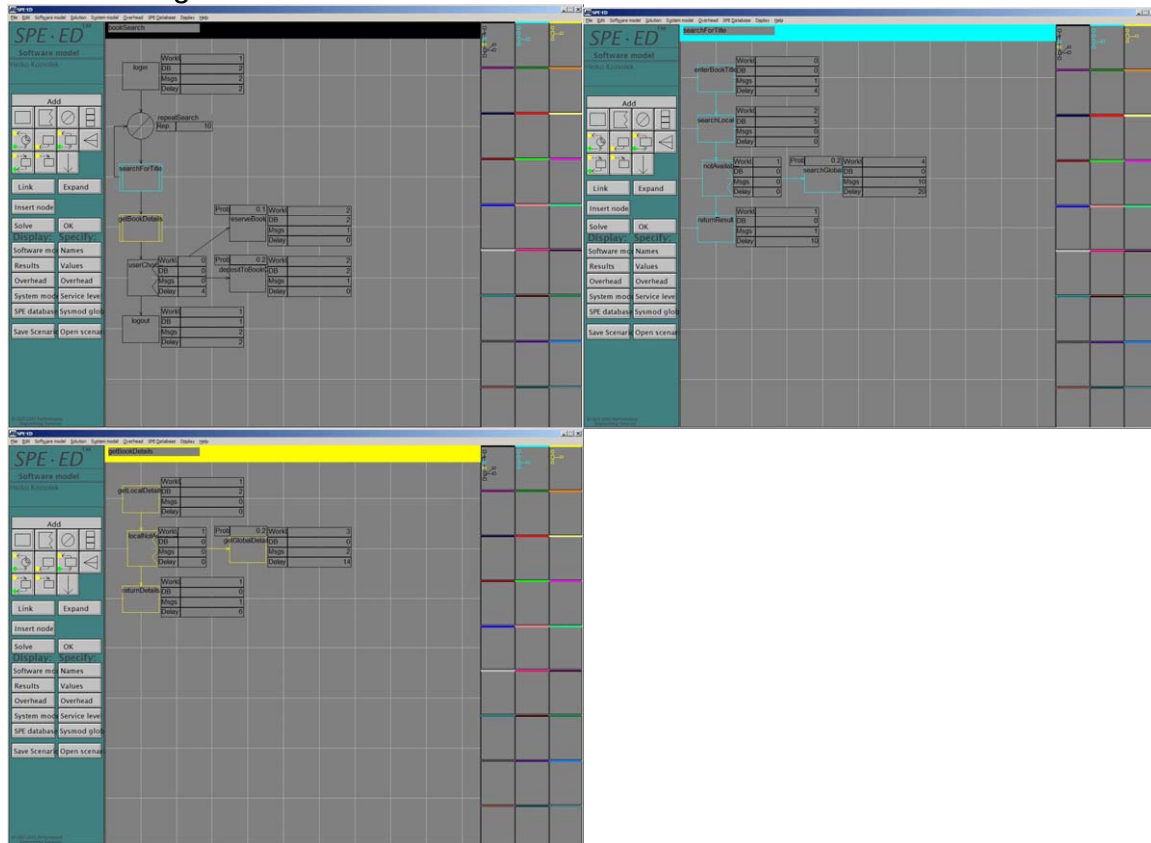
< PDFstring > := [< constantPDF > | < uniformPDF > | < exponentialPDF > | < normalPDF >]
< constantPDF > := < real > | "constant", < real >
< uniformPDF > := "uniform", < real > , < real >
< exponentialPDF > := "exponential", < real >
< normalPDF > := "normal", < real > , < real >

D. Musterlösung Übungen

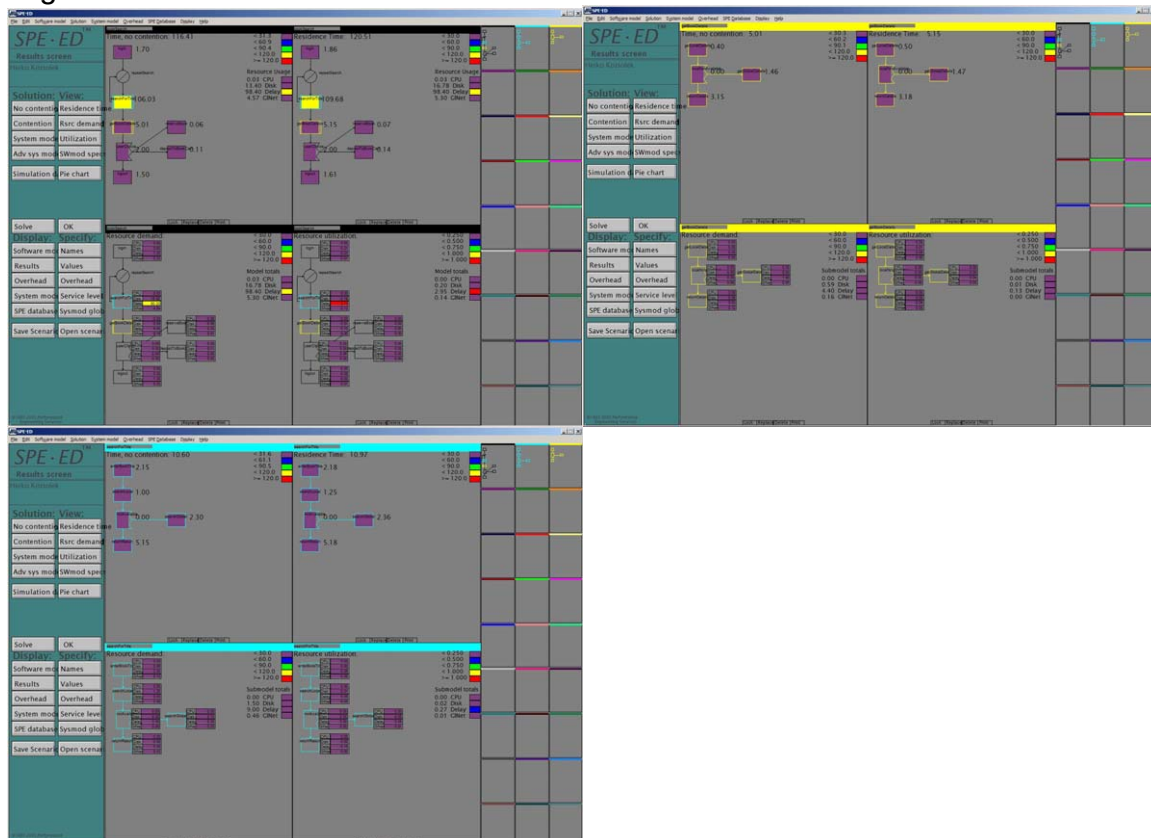
Die folgende Musterlösung wurde den Studenten nach der Rückgabe ihrer korrigierten Aufgabenzettel vorgestellt. Beim SPE-Verfahren wurden Screenshots der Ergebnisse einer möglichen Modellierung des untersuchten Verfahrens gemacht. Für das Capacity-Planning finden sich die Berechnung in Form von PowerPoint-Folien. Die UML-Diagramme für das umlPSI-Verfahren wurden mit dem CASE-Werkzeug Poseidon der Firma Gentleware erstellt. Näheres zur Auswertung der Ergebnisse der Übungen findet sich in Abschnitt [5.4.2](#).

SPE-Übung Musterlösung

Modellierung:



Ergebnisse:



Übungsblatt 2

Capacity Planning

Aufgabe 1) (Workload-Modellierung)

- Die Excel-Datei „Capacity_Planning.xls“ enthält eine Liste von Transaktionen mit CPU-Zeiten und entsprechenden I/O-Werten.
- Berechnen Sie die Grundstatistiken, die auf dem Arbeitsblatt angegeben sind.

	CPU Time (msec)	No. I/Os
Mean	29.94	37.08
Standard Deviation	33.46	27.98
Sample Variance	1119.85	782.61
Coeff. of Variation	1.12	0.75
Minimum	1.56	7.00
First Quartile (Q1)	7.42	17.00
Median (Q2)	12.61	23.00
Third Quartile (Q3)	28.27	45.25
Maximum	101.18	99.00
Range	99.62	92.00
Sum	2275.73	2818.00

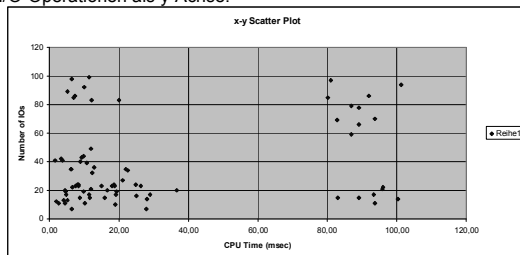
06.09.2004

Performance by Design

2

Aufgabe 1) (Workload-Modellierung)

- Zeichnen Sie einen x-y scatter plot für diesen Datensatz und benutzen Sie dabei die CPU-Zeit für die X-Achse und die Anzahl der I/O-Operationen als y-Achse.



06.09.2004

Performance by Design

3

Aufgabe 1) (Workload-Modellierung)

- Teilen Sie die Systemlast visuell in Cluster ein und geben Sie für jeden Cluster die Nummer, geschätzten Koordinaten des Mittelpunkts (CPU Time, I/Os) und Anzahl der Punkte an.

Cluster Nummer	CPU Time (msec)	No. I/Os	Anzahl der Punkte
1	15	25	51
2	10	90	8
3	95	15	7
4	90	80	10

06.09.2004

Performance by Design

4

Aufgabe 2) (Berechnung der Eingabeparameter für QN)

- Was ist der Gesamtdurchsatz des Systems in tps?
 - $X_0 = C_0 / T = 76 / 15 = 5,0667$ tps
- Was ist der Durchsatz für jede Klasse in tps?
 - $X_{0,r}$ = Anzahl der Transaktionen in Klasse r / T
 - $X_{0,1} = 51 / 15 = 3,4000$ tps
 - $X_{0,2} = 8 / 15 = 0,5333$ tps
 - $X_{0,3} = 7 / 15 = 0,4667$ tps
 - $X_{0,4} = 10 / 15 = 0,6667$ tps

06.09.2004

Performance by Design

5

Aufgabe 2) (Berechnung der Eingabeparameter für QN)

- Berechnen Sie die Service Demands jeweils für die CPU und die Festplatte.
 - Service Demand Law: $D_i = U_i / X_0$
 - mehrere Klassen: $D_{i,r} = U_{i,r} / X_{0,r}$
 - Gegeben: $X_{0,r}$, Gesucht $U_{i,r}$
 - Faktorisierung der Auslastung: $U_{i,r} = U_i * f_{i,r}$
 - Gegeben: U_i , Gesucht $f_{i,r}$
 - $f_{i,r}$ = Gesamtzeit in Klasse r / Gesamtzeit in allen Klassen

06.09.2004

Performance by Design

6

Aufgabe 2) (Berechnung der Eingabeparameter für QN)

- $f_{i,r}$ = Gesamtzeit in Klasse r / Gesamtzeit in allen Klassen
- Faktoren für die CPU
 - $f_{CPU,1} = (15 * 51) / 2275,73 = 0,3362$
 - $f_{CPU,2} = (10 * 8) / 2275,73 = 0,0352$
 - $f_{CPU,3} = (95 * 7) / 2275,73 = 0,2922$
 - $f_{CPU,4} = (90 * 10) / 2275,73 = 0,3955$
- Faktoren für die Festplatte
 - $f_{Disk,1} = (25 * 51) / 2818 = 0,4524$
 - $f_{Disk,2} = (90 * 8) / 2818 = 0,2555$
 - $f_{Disk,3} = (15 * 7) / 2818 = 0,0373$
 - $f_{Disk,4} = (80 * 10) / 2818 = 0,2839$

06.09.2004

Performance by Design

7

Aufgabe 2) (Berechnung der Eingabeparameter für QN)

- Auslastung der CPU
 - $U_{CPU,1} = U_{CPU} * f_{CPU,1} = 0,35 * 0,3362 = 0,1177$
 - $U_{CPU,2} = 0,35 * 0,0352 = 0,0123$
 - $U_{CPU,3} = 0,35 * 0,2922 = 0,1023$
 - $U_{CPU,4} = 0,35 * 0,3955 = 0,1384$
- Auslastung der Festplatte
 - $U_{Disk,1} = U_{Disk} * f_{Disk,1} = 0,8 * 0,4524 = 0,3619$
 - $U_{Disk,2} = 0,8 * 0,2555 = 0,2044$
 - $U_{Disk,3} = 0,8 * 0,0373 = 0,0298$
 - $U_{Disk,4} = 0,8 * 0,2839 = 0,2271$

06.09.2004

Performance by Design

8

Aufgabe 2) (Berechnung der Eingabeparameter für QN)

- Service Demand für CPU
 - $D_{CPU,1} = U_{CPU,1} / X_{0,1} = 0,1177 / 3,4 = 0,0346$
 - $D_{CPU,2} = 0,0123 / 0,5333 = 0,0231$
 - $D_{CPU,3} = 0,1023 / 0,4667 = 0,2192$
 - $D_{CPU,4} = 0,1384 / 0,6667 = 0,2076$
- Service Demand für die Festplatte
 - $D_{Disk,1} = U_{Disk,1} / X_{0,1} = 0,3619 / 3,4 = 0,1064$
 - $D_{Disk,2} = 0,2044 / 0,5333 = 0,3832$
 - $D_{Disk,3} = 0,0298 / 0,4667 = 0,0639$
 - $D_{Disk,4} = 0,2271 / 0,6667 = 0,3406$

06.09.2004

Performance by Design

9

Aufgabe 3) Analyse des QN

- Benutzen Sie „OpenQN.xls“ um das QN-Modell für das vorherige Beispiel zu lösen und nehmen Sie dabei an, dass die Ankunftsraten genauso groß sind, wie die Durchsätze der Klassen in der vorherigen Aufgabe.

Open Multiclass Queuing Networks

This workbook comes with the books "Performance by Design," "Capacity Planning for Web Services," and "Scaling for E-Business" by D. A. Brenasch and V. A. F. Almeida. Prentice Hall, 2004, 2002 and 2000.

Initialize		Solve		Help	
No. Queues:	2				
No. of Classes:	4				
Arrival Rates:	3,400	0,533	0,457	0,667	
Service Demand Matrix:					
Classes →					
Queues ↓	Type ↓ (L/U/D/M/Pn)	1	2	3	4
1 L		0,0346	0,0231	0,2192	0,2076
2 L		0,1064	0,3832	0,0639	0,3406

06.09.2004

Performance by Design

10

Aufgabe 3) Analyse des QN

- Benutzen Sie „OpenQN.xls“ um das QN-Modell für das vorherige Beispiel zu lösen und nehmen Sie dabei an, dass die Ankunftsraten genauso groß sind, wie die Durchsätze der Klassen in der vorherigen Aufgabe.

Residence Times:

	Class 1	Class 2	Class 3	Class 4
CPU	0,05499	0,03671	0,34837	0,32994
Disk	0,60122	2,16530	0,36107	1,92459
Response Time	0,65621	2,20202	0,70944	2,25452

Utilizations:

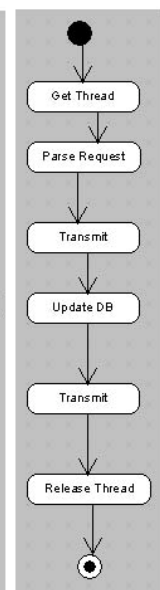
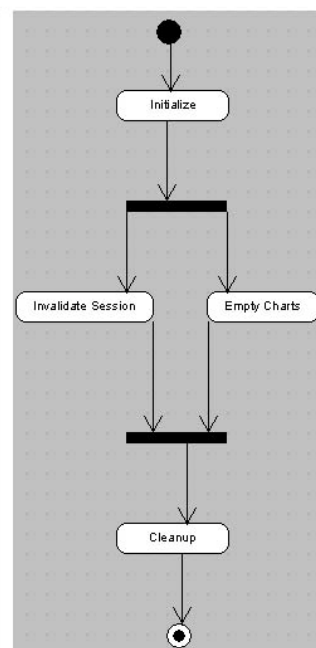
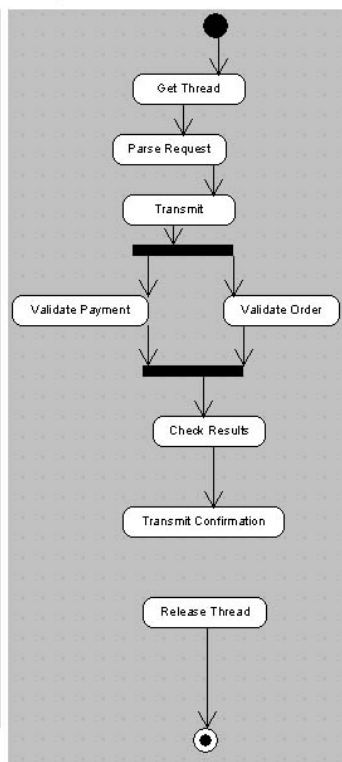
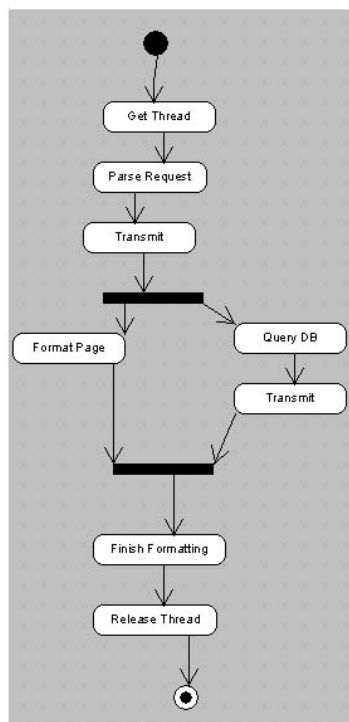
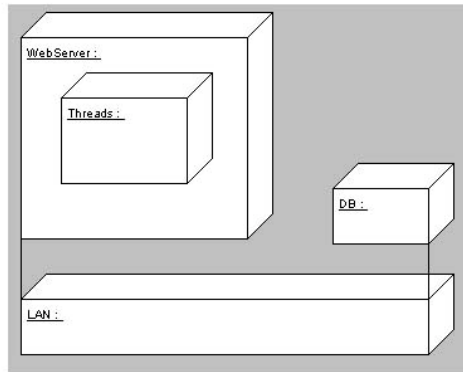
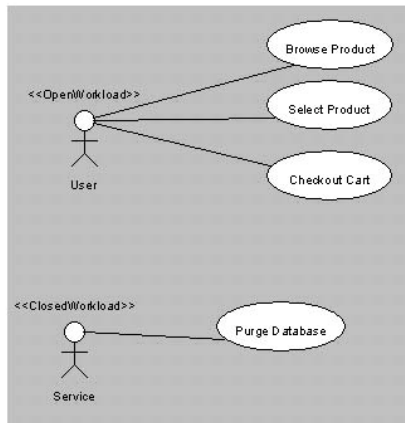
	Class 1	Class 2	Class 3	Class 4	Total
CPU	0,11764	0,01231	0,10237	0,13847	0,37079
Disk	0,36176	0,20425	0,02984	0,22718	0,82303

06.09.2004

Performance by Design

11

umlPSI Musterlösung



E. Aufgabenstellungen des Experiment

Die Aufgabenstellung des Experiments enthält Entwurfsdokumente für den untersuchten prototypischen Webserver. Es wurde versucht, die Eingaben für die Modellierung bei allen Verfahren möglichst gleich zu gestalten, jedoch sind z.B. die UML-Diagramme in der Aufgabenstellung für das Capacity Planning nicht enthalten, da diese für dieses Verfahren nutzlos sind. Weitere Ausführungen zur Aufgabenstellung des Experiment sind in Abschnitt [5.5](#) enthalten.

Aufgabenstellung

Ein neuartiger komponentenbasierter Web-Server soll entwickelt werden. Während des Entwurfs diskutieren die Entwickler verschiedene Alternativen für die Architektur. Da Sie mit Performance-Analyseverfahren für Software-Architekturen vertraut sind, werden Sie gebeten, die vorgeschlagenen Entwurfsalternativen auf ihre Performance-Eigenschaften hin zu untersuchen.

Die Entwickler legen Ihnen für die Entwurfsalternativen jeweils ein Komponenten- und ein Sequenzdiagramm vor. Ein Systemadministrator liefert Ihnen darüber hinaus Daten für die Hardware, auf der der Web-Server zum Einsatz gebracht werden soll.

Ihre Aufgabe besteht nun darin, mittels des kennen gelernten Verfahrens des Software Performance Engineering (SPE) die Performance-Werte für die vorgeschlagenen Entwurfsalternativen zu bestimmen. Anschließend sollen Sie eine Empfehlung für eine aus Performance-orientierter Sicht günstige Entwurfsalternative geben. Ihre Aufgaben sind im Einzelnen:

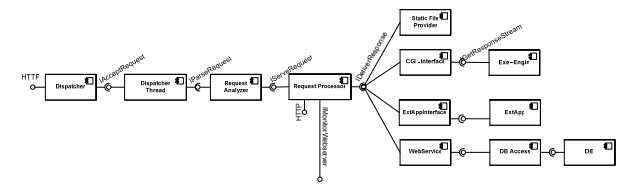
- Erstellen Sie anhand der Sequenzdiagramme mit dem Tool SPE-ED die entsprechenden Execution Graphen.
- Wählen Sie im Menü mit „Overhead/Specify Overhead“ aus, wählen Sie unter im Dialog „Select spec template“ den Button „New“ und im Dialog „Select facility template“ den Button „Show all“. Wählen Sie aus der Liste das Facility template „NachtF Server“ aus. Tragen Sie in die nun angezeigte Tabelle die unter Processing Overhead (s.u.) stehenden Daten ein.
- Bestimmen Sie für jeden Knoten der Execution-Graphen die „Software Resource Requirements“. Sofern keine Vorgaben für die Werte gegeben sind, versuchen Sie die Daten zu schätzen. Versuchen Sie möglichst realistische Schätzungen vorzunehmen.
- Berechnen Sie die Residence Times, Resource Usages und Utilizations für die verschiedenen Entwurfsalternativen und tragen Sie ihre Ergebnisse in die vorgegebene Tabelle ein. Notieren Sie zusätzlich weitere Vor- und Nachteile jeder Entwurfsalternative.
- Favorisieren Sie eine der modellierten Entwurfsentscheidungen und machen Sie den Entwicklern eine begründete Empfehlung für diese Entscheidung. Orientieren Sie sich nicht nur an den ermittelten Performance-Werten, sondern ziehen sie alle verfügbaren Faktoren in Betracht.

Weitere Angaben

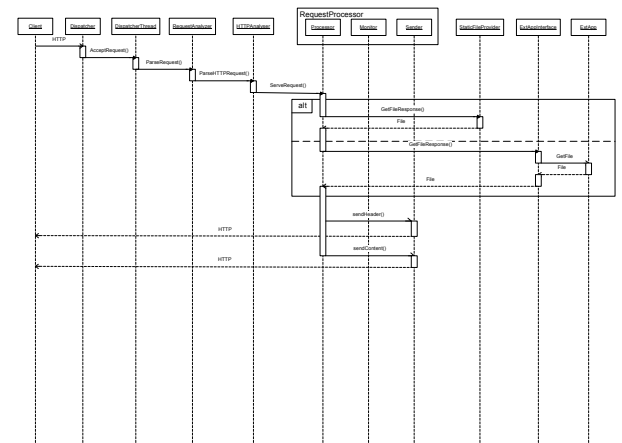
- Die Aufrufe „HTTP“ im Sequenzdiagramm modellieren HTTP-Get Anfragen, die vom Client gestellt bzw. die Antworten des Servers, die jeweils über das Internet übertragen werden.
- Der Web-Server greift bei Anfragen für statische HTML-Seiten (Wahrscheinlichkeit: 40%) auf seine lokale Festplatte zu.
- Bei Anfragen für dynamische HTML-Seiten (Wahrscheinlichkeit: 60%) werden die dazu benötigten Daten aus einer externen Applikation bezogen, die auf einem weiteren Server ausgeführt wird, welcher über ein lokales Netzwerk mit dem Web-Server verbunden ist. Die externe Applikation braucht im Schnitt etwa eine halbe Sekunde, um die benötigten Daten zu sammeln.
- Einen Delay für User Eingaben gibt es in dieser Modellierung *nicht* (!), es wird nur die Performance einzelner Anfragen und nicht ganzer Eingabesequenzen gemessen. Der im Processing Overhead angegebene Delay soll zur Angabe der Rechendauer der externen Applikation verwendet werden.
- Mittlere Größe der versendeten Dateien: ca. 5 KByte
- Die Ankunftsrate für neue Anfragen liegt nach Schätzungen der Entwickler bei einer Anfrage pro Sekunde.
- Der Web-Server ist multithreaded und startet in der Komponente „DispatcherThread“ für jede Anfrage einen neuen Thread ab.

Ausgangsarchitektur

Komponentendiagramm



Sequenzdiagramm



Software Resources

- Workload:** die Zeit für eine Berechnung auf der CPU (Beispiel: Wert „5“ bedeutet, dass 500 Mikrosekunden auf der CPU gerechnet werden muss)
- DiskAccess:** die Anzahl der Festplattenzugriffe
- INetMsgSize:** die Größe einer Anfrage, die über das Internet verschickt wird in KByte
- LANMsgSize:** die Größe einer Anfrage, die über das LAN verschickt wird in KByte
- ExtApp:** die Zeit für die Berechnung der externen Applikation in Sekunden

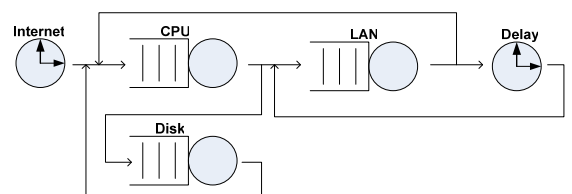
Processing Overhead

Devices	CPU	Disk	INet	Delay	LAN
Quantity	1	1	1	1	1
Service Units	sec	Phys I/O	KBytes	sec	KBytes
Workload	0.0001				
DiskAccess	0.01	2			
INetMsgSize	0.0005		1		
LANMsgSize	0.0005				1
ExtApp				1	
Service Time	1	0.003	0.125	1	0.0008

Die Service Times wurden folgendermaßen bestimmt:

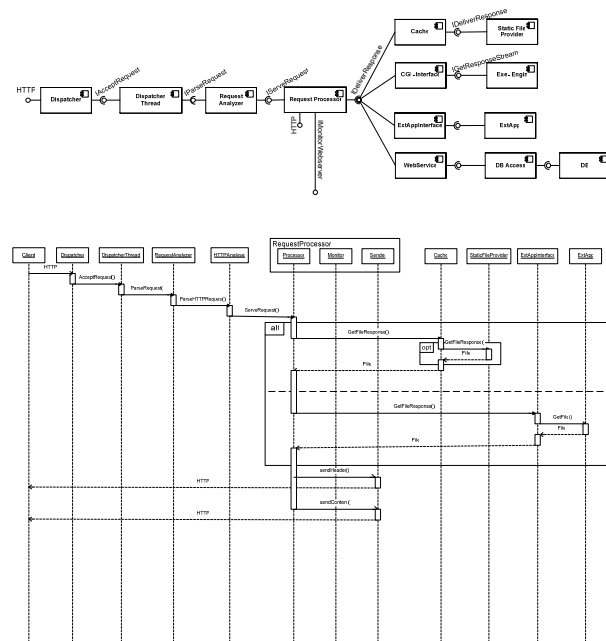
- CPU: durch Messungen
- Disk: Spezifikation 3 ms pro I/O
- INet: 64 KBit/sec (ISDN) -> 8 KB/sec -> 0,125 sec/KB
- LAN: 10 MBit/sec (Ethernet) -> 1250 KB/sec -> 0,0008 sec/KB
- Delay: willkürliche Festlegung, kann durchaus geändert werden

Ressourcenverteilung (nur zur Illustration)



Entwurfsalternativen

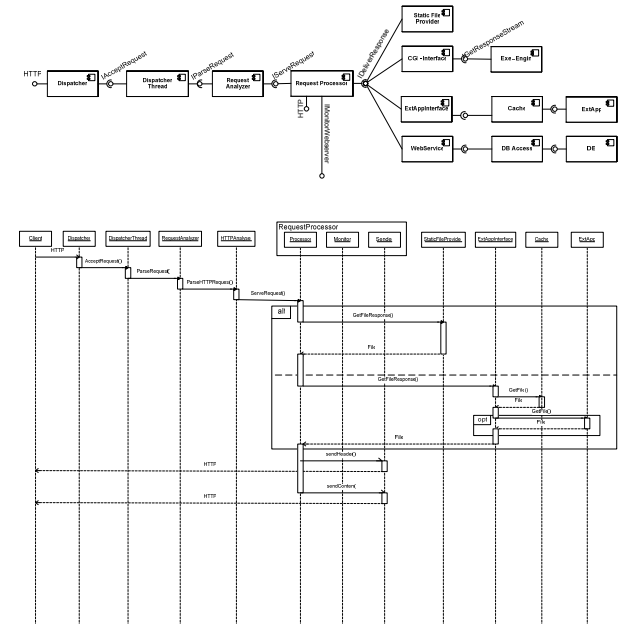
Alternative 1a (Server-Cache für statische Seiten)



Beschreibung:

- Ein Cache im Hauptspeicher des Web-Servers spart Festplattenzugriffe ein.
- Im Mittel könnten schätzungsweise ca. 70% aller Anfrage auf statische Seiten direkt aus dem Cache beantwortet werden.

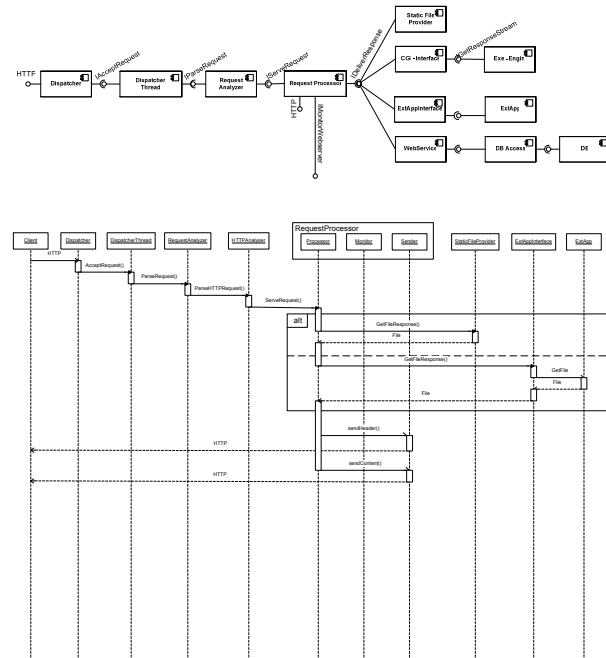
Alternative 1b (Server-Cache für dynamische Seiten)



Beschreibung:

- Ein Cache im Hauptspeicher des Web-Servers spart Zugriffe auf die externe Applikation ein. Dies funktioniert nur bei Inhalten, die sich selten ändern und nicht bei jeder Anfrage neu erzeugt werden müssen. (Beispiele: Artikelbeschreibungen, tägliche Wetterdaten, Lagerbestände usw.)
- Im Mittel könnten schätzungsweise ca. 20% aller Anfragen an die externe Applikationen direkt aus dem Cache beantwortet werden.
- Anmerkung: Die beiden Cache-Varianten schließen sich gegenseitig aus, da nicht genügend Hauptspeicher zur Verfügung steht.

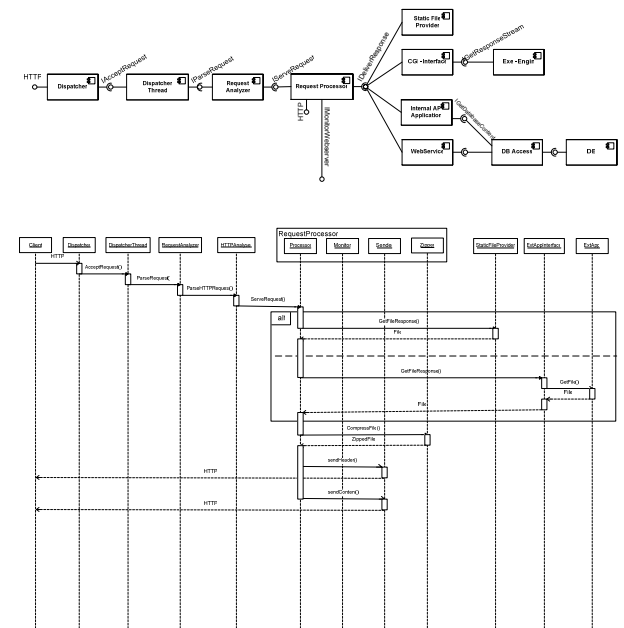
Alternative 2 (Singlethreaded)



Beschreibung:

- Der Server wird als nicht als multithreaded implementiert. Dadurch fällt die Rechenzeit für das Starten bzw. den Kontextwechsel der Threads weg. Jedoch bildet sich vor dem Server eine größere Warteschlange, da immer nur eine Anfrage gleichzeitig bearbeitet werden kann.
- Die Architektur bzw. die Anordnung der Komponenten ändert sich bei dieser Alternative nicht.

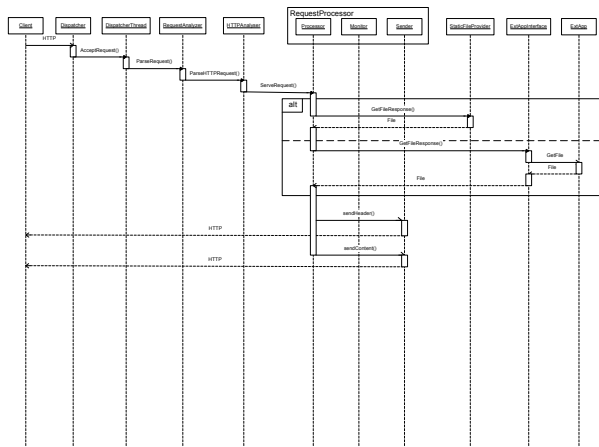
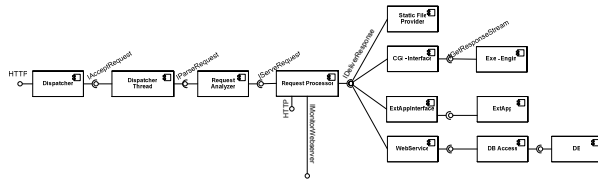
Alternative 3 (Komprimierung)



Beschreibung:

- Die Dateien werden vor dem Versenden auf dem Server komprimiert, dies wird auf der Client-Seite von allen modernen Browsern unterstützt.
- Eine Kompressionsrate für die Dateien muss abgeschätzt werden.
- Durch die Kompression reduziert sich die Internet-Auslastung, es entsteht jedoch eine höhere CPU-Last für die Ausführung des Kompressionsalgorithmus.
- Die Rechenzeit des Clients für die Dekompression der Dateien wird in diesem Modell nicht berücksichtigt.

Alternative 4 (Clustering)



Beschreibung:

- Der Web-Server wird auf einem Cluster von zwei Rechnern installiert. Ein Scheduler verteilt die Anfragen gleichmäßig. Die externe Applikation läuft weiterhin auf einem einzelnen über das LAN angebundenen Server.
- Durch die verdoppelte Hardware entstehen Kosten, zusätzlich muss ein Scheduler implementiert werden.
- Die Verteilung der Last verursacht einen gewissen Rechenaufwand.

Bewertung der Alternativen

Alternative	Vorteile	Nachteile
1a		
1b		
2		
3		
4		

Ergebnisse

Alternative	Residence Time	Resource Usage	Utilization
1a		CPU: DISK: INet: Delay: LAN:	CPU: DISK: INet: Delay: LAN:
1b		CPU: DISK: INet: Delay: LAN:	CPU: DISK: INet: Delay: LAN:
2		CPU: DISK: INet: Delay: LAN:	CPU: DISK: INet: Delay: LAN:
3		CPU: DISK: INet: Delay: LAN:	CPU: DISK: INet: Delay: LAN:
4		CPU: DISK: INet: Delay: LAN:	CPU: DISK: INet: Delay: LAN:

Ihre Entwurfsempfehlung: _____

Begründung für ihre Entwurfsempfehlung:

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Aufgabenstellung

Ein neuartiger komponentenbasierter Web-Server soll entwickelt werden. Während des Entwurfs diskutieren die Entwickler verschiedene Alternativen für die Architektur. Da Sie mit Performance-Analyseverfahren für Software-Architekturen vertraut sind, werden Sie gebeten die vorgeschlagenen Entwurfsalternativen auf ihre Performance-Eigenschaften hin zu untersuchen.

Die Entwickler legen Ihnen eine beispielhafte Log-Datei (Workload.xls) vor, die mittels eines Prototyps aufgenommen wurde. Diese Log-Datei kann für die Modellierung der Systemlast verwendet werden. Für die geplante Ressourcenverteilung wurde ein Queueing-Network angegeben, außerdem konnte am Prototyp bereits die Auslastungen von CPU, Festplatte und lokalem Netzwerk während des Testlaufs mit einem Performance-Monitor gemessen werden.

Ihre Aufgabe besteht nun darin, mittels des kennen gelernten Verfahrens des „Capacity Planning“ die Performance-Werte für die vorgeschlagene Architektur und verschiedene Entwurfsalternativen zu bestimmen. Anschließend sollen Sie den Entwicklern eine Empfehlung für eine aus Performance-orientierter Sicht günstige Entwurfsalternative geben. Ihre Aufgaben sind im Einzelnen:

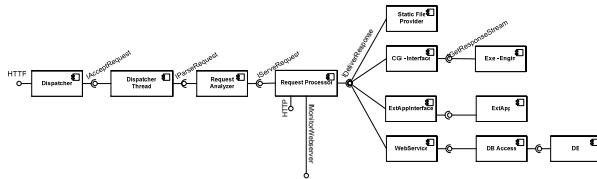
- Teilen Sie die Systemlast in Cluster ein. Bei den Anfragen an den Web-Server bietet sich eine Einteilung in Abfragen auf statische bzw. auf dynamische Seiten an. Sortieren Sie dazu das Log-File (Menü: Daten/Sortieren...) nach der ersten Spalte und bestimmen Sie die Mittelwerte je Ressource (CPU, Disk, LAN) und Systemlastklasse (dynamisch, statisch).
- Berechnen Sie vorhanden die Eingabeparameter für das Queueing Network.
 - Ankunftsrate: das Log-File wurde innerhalb von 1000 Sekunden aufgenommen, geben Sie die Ankunftsrate für die beiden Systemlastklassen an.
 - Service-Demands: diese Werte können mittels der angegebenen Ressourcenauslastungen (Utilizations) über das Service Demand Law bestimmt werden. Führen Sie dazu vorher eine Faktorisierung der Auslastung nach Lastklassen durch.
 - Da das Internet und die externe Applikation als Delay-Knoten modelliert wurden (s.u.), sind die Verzögerungen an diesen Ressourcen konstant. Gehen beim Internet von einer ISDN-Verbindung (64KBit/s) aus und berechnen Sie, wie lange das Versenden von 7 KByte (1 KByte Client-Request, 1 KByte Header, 5 KByte mittlere Größe der versandten Dateien) dauert. Geben Sie für den ExtApp-Knoten einen festen Wert von 0,5 Sekunde vor (natürlich nur bei der Systemlastklasse, die auch auf diese Ressource zugreift).
- Benutzen Sie die Datei OpenQN.xls um das Queueing-Network zu lösen. Modellieren Sie CPU, LAN und Disk als LI-Knoten (Load Independent) und Internet und ExtApp jeweils als D-Knoten (Delay). Tragen Sie ihre

berechneten Werte ein, lösen Sie das Queueing-Network und speichern Sie die Datei ab.

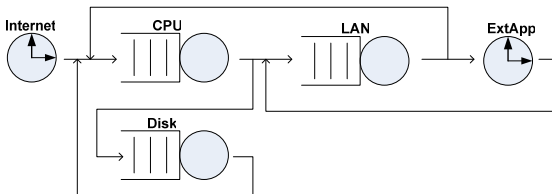
- Berechnen Sie die Antwortzeiten und Auslastungen für die verschiedenen Entwurfsalternativen und speichern Sie ihre Ergebnisse jeweils in separaten Dateien (OpenQN1a.xls, OpenQN1b.xls, OpenQN2.xls usw.). Überlegen Sie sich dazu, wie die Beschreibungen der Entwurfsalternativen durch das Verfahren abgebildet werden können und welche Eingabeparameter für das Queueing-Network in welcher Form geändert werden müssen.
- Notieren Sie zusätzlich weitere Vor- und Nachteile der Entwurfsalternative.
- Favorisieren Sie eine der modellierten Entwurfsentscheidungen und machen Sie den Entwicklern eine begründete Empfehlung für diese Entscheidung. Orientieren Sie sich nicht nur an den ermittelten Performance-Werten, sondern ziehen sie alle verfügbaren Faktoren in Betracht.

Ausgangsarchitektur

Komponentendiagramm (nur zur Illustration)



Queueing Network zur Ressourcenverteilung



Gemessene Auslastungen der Ressourcen

	Utilization
CPU	0,082
Disk	0,012
LAN	0,004

Weitere Angaben

- Mittlere Größe der versendeten Dateien: ca. 5 KByte
- Die Ankunftsrate für neue Anfragen ist aus dem Logfile ersichtlich (dieses wurde innerhalb von 1000 Sekunden aufgenommen) und beträgt für alle Arten von Anfragen eine Anfrage pro Sekunde.
- Der Web-Server ist multithreaded und startet in der Komponente „DispatcherThread“ für jede Anfrage einen neuen Thread ab.

Entwurfsalternativen

Alternative 1a (Server-Cache für statische Seiten)

Beschreibung:

- Ein Cache im Hauptspeicher des Web-Servers spart Festplattenzugriffe ein.
- Im Mittel könnten schätzungsweise ca. 70% aller Anfrage auf statische Seiten direkt aus dem Cache beantwortet werden.

Alternative 1b (Server Cache für dynamische Seiten)

Beschreibung:

- Ein Cache im Hauptspeicher des Web-Servers spart Zugriffe auf die externe Applikation ein. Dies funktioniert nur bei Inhalten, die sich selten ändern und nicht bei jeder Anfrage neu erzeugt werden müssen. (Beispiele: Artikelbeschreibungen, tägliche Wetterdaten, Lagerbestände usw.)
- Im Mittel könnten schätzungsweise ca. 20% aller Anfragen an die externe Applikationen direkt aus dem Cache beantwortet werden.

- Anmerkung: Die beiden Cache-Varianten schließen sich gegenseitig aus, da nicht genügend Hauptspeicher zur Verfügung steht.

Alternative 2 (Singlethreaded)

Beschreibung:

- Der Server wird als nicht als multithreaded implementiert. Dadurch fällt die Rechenzeit für das Starten bzw. den Kontextwechsel der Threads weg. Jedoch bildet sich vor dem Server eine größere Warteschlange, da immer nur eine Anfrage gleichzeitig bearbeitet werden kann.
- Die Architektur bzw. die Anordnung der Komponenten ändert sich bei dieser Alternative nicht.

Alternative 3 (Komprimierung)

Beschreibung:

- Die Dateien werden vor dem Versenden auf dem Server komprimiert, dies wird auf der Client-Seite von allen modernen Browsern unterstützt.
- Eine Kompressionsrate für die Dateien muss abgeschätzt werden.
- Durch die Kompression reduziert sich die Internet-Auslastung, es entsteht jedoch eine höhere CPU-Belastung für die Ausführung des Kompressionsalgorithmus.
- Die Rechenzeit des Clients für die Dekompression der Dateien wird in diesem Modell nicht berücksichtigt.

Alternative 4 (Clustering)

Beschreibung:

- Der Web-Server wird auf einem Cluster von zwei Rechnern installiert. Ein Scheduler verteilt die Anfragen gleichmäßig. Die externe Applikation läuft weiterhin auf einem einzelnen über das LAN angebundenen Server.
- Durch die verdoppelte Hardware entstehen Kosten, zusätzlich muss ein Scheduler implementiert werden.
- Die Verteilung der Last verursacht einen gewissen Rechenaufwand.
- Hinweis: die LI-Knoten im QN für CPU und Disk können in MP2-Knoten umgewandelt werden

Ergebnisse

Alternative	Vorteile	Nachteile
1a		
1b		
2		
3		
4		

Bewertung der Alternativen

Ihre Entwurfsempfehlung: _____

Begründung für ihre Entwurfsempfehlung:

[illegible]

Aufgabenstellung

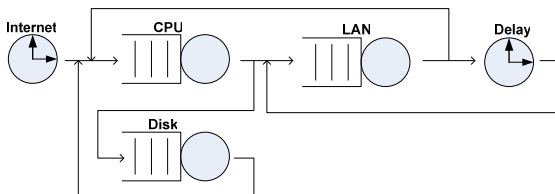
Ein neuartiger komponentenbasierter Web-Server soll entwickelt werden. Während des Entwurfs diskutieren die Entwickler verschiedene Alternativen für die Architektur. Da Sie mit Performance-Analyseverfahren für Software-Architekturen vertraut sind, werden Sie gebeten die vorgeschlagenen Entwurfsalternativen auf ihre Performance-Eigenschaften hin zu untersuchen.

Die Entwickler legen Ihnen eine Datei für Poseidon vor (webserver.zargo), in der der grobe Entwurf mittels UML dokumentiert wurde. Zusätzlich wurden Angaben zu den verschiedenen Entwurfsalternativen gemacht (s.u.).

Ihre Aufgabe besteht nun darin, die vorgelegten Diagramme mittels des UML Performance Profiles mit sinnvollen Performance-Werten zu attributieren. Danach erstellen Sie Modelle für die einzelnen Entwurfsalternativen und speichern diese jeweils in separaten Dateien ab. Die Simulation der Modelle kann im Rahmen dieses Versuchs nicht durchgeführt werden und erfolgt nachträglich. Ihre Aufgaben sind im Einzelnen:

- Tragen Sie die noch fehlenden Werte in den Tagged-Values der Diagramme ein. Schätzen Sie die Dauer für die PAdemand-Tags (in der Eingabedatei ist überall 0.0 eingetragen!). Versuchen Sie möglichst realistische Werte zu schätzen.
- Überlegen Sie sich wie Sie die Beschreibungen der Entwurfsalternativen jeweils durch das Verfahren modellieren können. Nehmen Sie die Datei „webserver.zargo“ als Ausgangsbasis und speichern Sie die modellierten Entwurfsalternativen jeweils in einer separaten Datei („webserver1a.zargo“, „webserver1b.zargo“, webserver2.zargo usw.).
- Notieren Sie zusätzlich weitere Vor- und Nachteile jeder Entwurfsalternative.

Ressourcenverteilung (nur zur Illustration)

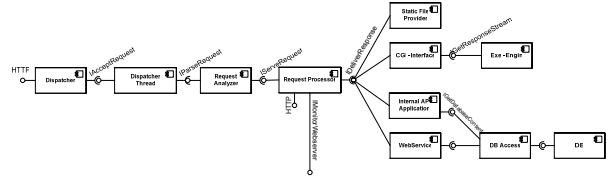


Weitere Angaben

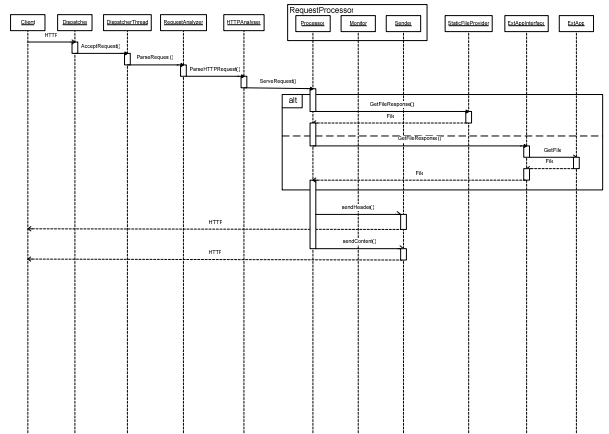
- Die Aufrufe „HTTP“ in den Activity-Diagrammen modellieren HTTP-Get Anfragen, die vom Client gestellt bzw. die Antworten des Servers, die jeweils über das Internet mit einer ISDN-Leitung übertragen werden.
- Der Web-Server greift bei Anfragen für statische HTML-Seiten (Wahrscheinlichkeit: 40%) auf seine lokale Festplatte zu.
- Bei Anfragen für dynamische HTML-Seiten (Wahrscheinlichkeit: 60%) werden die dazu benötigten Daten aus einer externen Applikation bezogen, die auf einem weiteren Server ausgeführt wird, welcher über ein lokales Netzwerk mit dem Web-Server verbunden ist. Die externe Applikation braucht im Schnitt etwa eine halbe Sekunde, um die benötigten Daten zu sammeln.
- Mittlere Größe der versendeten Dateien: ca. 5 KByte
- Die Ankunftsrate für neue Anfragen liegt nach Schätzungen der Entwickler bei einer Anfrage pro Sekunde.
- Der Web-Server ist multithreaded und startet in der Komponente „DispatcherThread“ für jede Anfrage einen neuen Thread (maximal 10) ab.

Ausgangsarchitektur

Komponentendiagramm (nur zur Illustration)



Sequenzdiagramm (nur zur Illustration)



Entwurfsalternativen

Alternative 1a (Server-Cache für statische Seiten)

Beschreibung:

- Ein Cache im Hauptspeicher des Web-Servers spart Festplattenzugriffe ein.
- Im Mittel könnten schätzungsweise ca. 70% aller Anfrage auf statische Seiten direkt aus dem Cache beantwortet werden.
- Hinweis: Modellierung z.B. durch das Hinzufügen eine Aktivität für Cachezugriffe

Alternative 1b (Server-Cache für dynamische Seiten)

Beschreibung:

- Ein Cache im Hauptspeicher des Web-Servers spart Zugriffe auf die externe Applikation ein. Dies funktioniert nur bei Inhalten, die sich selten ändern und nicht bei jeder Anfrage neu erzeugt werden müssen. (Beispiele: Artikelbeschreibungen, tägliche Wetterdaten, Lagerbestände usw.)
- Im Mittel könnten schätzungsweise ca. 20% aller Anfragen an die externe Applikationen direkt aus dem Cache beantwortet werden.
- Anmerkung: Die beiden Cache-Varianten schließen sich gegenseitig aus, da nicht genügend Hauptspeicher zur Verfügung steht.

Alternative 2 (Singlethreaded)

Beschreibung:

- Der Server wird als nicht als multithreaded implementiert. Dadurch fällt die Rechenzeit für das Starten bzw. den Kontextwechsel der Threads weg. Jedoch bildet sich vor dem Server eine größere Warteschlange, da immer nur eine Anfrage gleichzeitig bearbeitet werden kann.
- Die Architektur bzw. die Anordnung der Komponenten ändert sich bei dieser Alternative nicht.

Alternative 3 (Komprimierung)

Beschreibung:

- Die Dateien werden vor dem Versenden auf dem Server komprimiert, dies wird auf der Client-Seite von allen modernen Browsern unterstützt.
- Eine Kompressionsrate für die Dateien muss abgeschätzt werden.
- Durch die Kompression reduziert sich die Internet-Auslastung, es entsteht jedoch eine höhere CPU-Last für die Ausführung des Kompressionsalgorithmus.
- Die Rechenzeit des Clients für die Dekompression der Dateien wird in diesem Modell nicht berücksichtigt.

Beschreibung:

- Beschreibung:**

Alternative	Antwortzeiten	Auslastung
1a	Static: Dynamic:	CPU: Disk: Internet: ExtNode: LAN:
1b	Static: Dynamic:	CPU: Disk: Internet: ExtNode: LAN:
2	Static: Dynamic:	CPU: Disk: Internet: ExtNode: LAN:
3	Static: Dynamic:	CPU: Disk: Internet: ExtNode: LAN:
4	Static: Dynamic:	CPU: Disk: Internet: ExtNode: LAN:

[illegible]

Begründung für ihre Entwurfsempfehlung:

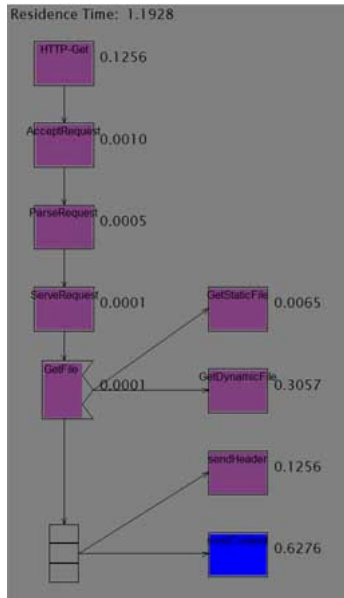
[illegible]

F. Lösung des Experiments

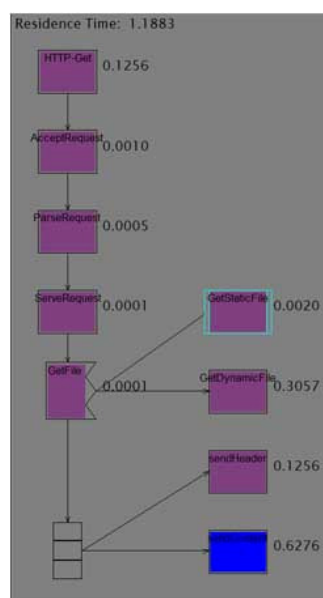
Im Folgenden findet sich für jedes Verfahren eine *mögliche* Lösung der Aufgabenstellung des Experiments. Leicht abweichende Modellierungen können ebenfalls gültig sein, denn es fließen z.T. individuelle Schätzungen des Entwicklers ein. Die Lösung wurde vor dem Experiment erstellt, um die Durchführbarkeit der Aufgabenstellung zu testen. Enthalten sind hier die Ausführungsgraphen des SPE-ED-Werkzeugs jeweils mit Durchlaufzeiten, die Berechnungen für das CP-Verfahren und die UML-Diagramme, die in Poseidon verwendet wurden.

SPE - Lösung

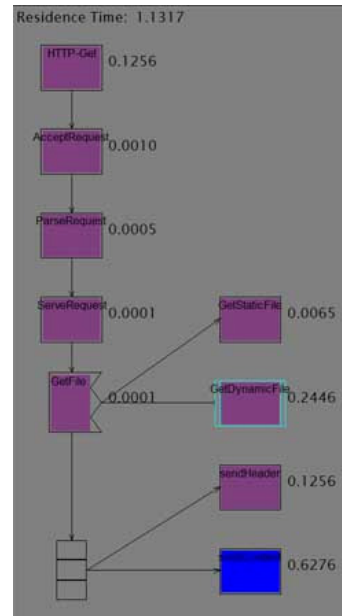
Alternative 0



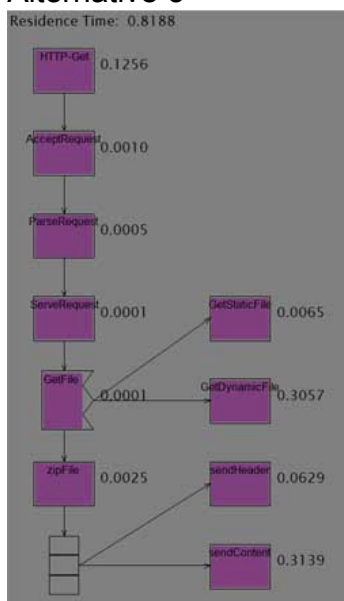
Alternative 1a



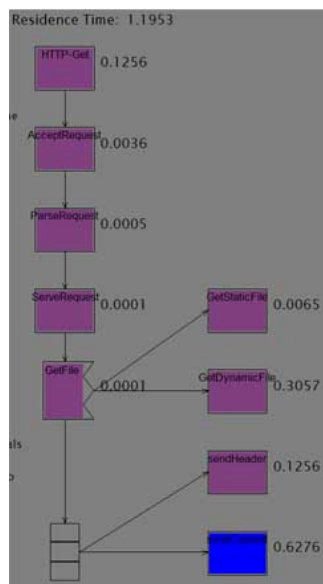
Alternative 1b



Alternative 3



Alternative 4



Studie Performance-Evaluationstechniken

Capacity Planning

Workload-Modellierung

- Teilen Sie die Systemlast in Cluster ein. Bei den Anfragen an den Web-Server bietet sich eine Einteilung in Abfragen auf statische bzw. auf dynamische Seiten an. Sortieren Sie dazu das Log-File (Menü: Daten/Sortieren...) nach der ersten Spalte und bestimmen Sie die Mittelwerte je Ressource (CPU, Disk, LAN) und Systemlastklasse (dynamisch, statisch).

File Type	CPU (sec)	Disk (IO)	LAN (sec)
Dynamic	0,0123	0	0,0058
Dynamic	0,0108	0	0,0051
Dynamic	0,0124	0	0,0058
Dynamic	0,0114	0	0,0054
Dynamic	0,0120	0	0,0056
Dynamic	0,0119	0	0,0056
Dynamic	0,0123	0	0,0058
Dynamic	0,0111	0	0,0052
Dynamic	0,0123	0	0,0058
	...	...	...

Mittelwert	Dynamic	Static
CPU	0,0116	0,0038
Disk	0	17,837
LAN	0,0054	0

Summe aller Anfragen:		
CPU	8,3576	
Disk	7331	
LAN	3,208	

21.10.2004

Performance by Design

2

Eingabeparameter QN

- Berechnen Sie die Eingabeparameter für das Queueing Network.
 - Ankunftsrate: das Log-File wurde innerhalb von 1000 Sekunden aufgenommen, geben Sie die Ankunftsrate für die beiden Systemlastklassen an.

	Ankunftsrate
Dynamic	0,589
Static	0,411

21.10.2004

Performance by Design

3

Eingabeparameter QN

- Berechnen Sie die Eingabeparameter für das Queueing Network.
 - Service-Demands: diese Werte können mittels der angegebenen Ressourcenauslastungen (Utilizations) über das Service Demand Law bestimmt werden. Führen Sie dazu vorher eine Faktorisierung der Auslastung nach Lastklassen durch.
- Service Demand Law: $D_{i,r} = U_{i,r} / X_{0,r}$
 - gegeben $X_{0,r}$ (eben berechnet) und U_i (Aufgabenblatt)
 - gesucht: $U_{i,r}$

21.10.2004

Performance by Design

4

Eingabeparameter QN

- $U_{i,r} = U_i * f_{i,r}$
- $f_{i,r}$ = Gesamtzeit in Klasse r / Gesamtzeit in allen Klassen
 - $f_{CPU,dynamic} = (0,0116 * 589) / 8,3576 = 0,8144$
 - $f_{CPU,static} = (0,0038 * 411) / 8,3576 = 0,1856$
 - $f_{Disk,dynamic} = (0 * 589) / 7331 = 0$
 - $f_{Disk,static} = (17,8370 * 411) / 7331 = 1$
 - $f_{LAN,dynamic} = (0,054 * 589) / 3,2088 = 1$
 - $f_{LAN,static} = (0 * 411) / 3,2088 = 0$

21.10.2004

Performance by Design

5

Eingabeparameter QN

- Berechnen Sie die Eingabeparameter für das Queueing Network.
 - Auslastung: $U_{i,r} = U_i * f_{i,r}$

Auslastung	Dynamic	Static
CPU	0,066781	0,015219
Disk	0	0,012
LAN	0,004	0

21.10.2004

Performance by Design

6

Eingabeparameter QN

- Berechnen Sie die Eingabeparameter für das Queueing Network.

– Service Demands: $D_{i,r} = U_{i,r} / X_{0,r}$

	Dynamic	Static
CPU	0,1134	0,037
Disk	0	0,0292
LAN	0,0068	0

21.10.2004

Performance by Design

7

Eingabeparameter QN

- Berechnen Sie die Eingabeparameter für das Queueing Network.

– Da das Internet und die externe Applikation als Delay-Knoten modelliert wurden (s.u.), sind die Verzögerungen an diesen Ressourcen konstant. Gehen beim Internet von einer ISDN-Verbindung (64KBit/s) aus und berechnen Sie, wie lange das Versenden von 7 KByte (1 KByte Client-Request, 1 KByte Header, 5 KByte mittlere Größe der versandten Dateien) dauert.

- Internet Delay:

– 64 KBit/s = 8 KByte/s

– $x = (7 \text{ KByte} * 1 \text{ s}) / 8 \text{ KByte} = 0,875 \text{ s}$

21.10.2004

Performance by Design

8

Eingabeparameter QN

- Berechnen Sie die Eingabeparameter für das Queueing Network.

– Geben Sie für den ExtApp-Knoten einen festen Wert von 0,5 Sekunde vor (natürlich nur bei der Systemlastklasse, die auch auf diese Ressource zugreift)

	Dynamic	Static
CPU	0,1134	0,037
Disk	0	0,0292
LAN	0,0068	0
Delay	0,5	0
Internet	0,875	0,875

21.10.2004

Performance by Design

9

Lösung des QN

- Benutzen Sie die Datei OpenQN.xls um das Queueing-Network zu lösen. Modellieren Sie CPU, LAN und Disk als LI-Knoten (Load Independent) und Internet und ExtApp jeweils als D-Knoten (Delay). Tragen Sie ihre berechneten Werte ein, lösen Sie das Queueing-Network und speichern Sie die Datei ab.

Open Multiclass Queueing Networks				
This workbook comes with the books "Performance by Design," "Capacity Planning for Web Services," and "Scaling for E-Business" by D. A. Menascé and V. A. F. Almeida, Prentice Hall, 2004, 2002 and 2000.				
	Initialize	Solve	Help	
No. Queues:	5			
No. of Classes:	2			
Arrival Rates:		Classes →		
		0,589	0,411	
		Service Demand Matrix:		Restore
		Classes →		
Queues ↓	Type ↓ (LI/D/MPn)	1	2	
1 LI		0,1134	0,037	
2 LI		0	0,0292	
3 LI		0,0068	0	
4 D		0,5	0	
5 D		0,875	0,875	

21.10.2004

Performance by Design

10

Lösung des QN

- Utilizations
- Residence Time

	Classes →		
Queues ↓	1	2	Total
1	0,06679	0,01521	0,08200
2	0,00000	0,01200	0,01200
3	0,00401	0,00000	0,00401
4	0,29450	0,00000	0,29450
5	0,51538	0,35963	0,87500

	Classes →		
Queues ↓	1	2	
1	0,12353	0,04030	
2	0,00000	0,02955	
3	0,00683	0,00000	
4	0,50000	0,00000	
5	0,87500	0,87500	
Response Time	1,50538	0,94486	

21.10.2004

Performance by Design

11

Lösung des QN

- Berechnen Sie die Antwortzeiten und Auslastungen für die verschiedenen Entwurfsalternativen.
- Überlegen Sie sich dazu, wie die Beschreibungen der Entwurfsalternativen durch das Verfahren abgebildet werden können und welche Eingabeparameter für das Queueing-Network in welcher Form geändert werden müssen.

21.10.2004

Performance by Design

12

Alternative 1a (statischer Cache)

• Service Demands:

Service Demands (Alternative 1a)		
	Dynamic	Static
CPU	0,1134	0,037
Disk	0	0,00876 = $0,0292 * 0,3$
LAN	0,0068	0
Delay	0,5	0
INet	0,875	0,875

Residence Time		
	dynamic	static
CPU	0,12353	0,04030
Disk	0,00000	0,00879
LAN	0,00683	0,00000
Delay	0,50000	0,00000
Internet	0,87500	0,87500
Response Time	1,50536	0,92410

Utilization			
	dynamic	static	Total
CPU	0,06679	0,01521	0,08200
Disk	0,00000	0,00360	0,00360
LAN	0,00401	0,00000	0,00401
Delay	0,29450	0,00000	0,29450
Internet	0,51538	0,35963	0,87500

21.10.2004

Performance by Design

13

Alternative 1b (dynamischer Cache)

• Service Demands:

Service Demands (Alternative 1a)		
	Dynamic	Static
CPU	0,1134	0,037
Disk	0	0,0292
LAN	0,00544 = $0,0068 * 0,8$	0
Delay	0,4 = $0,5 * 0,8$	0
INet	0,875	0,875

Residence Time		
	dynamic	static
CPU	0,12353	0,04030
Disk	0,00000	0,02955
LAN	0,00546	0,00000
Delay	0,40000	0,00000
Internet	0,87500	0,87500
Response Time	1,40399	0,94486

Utilization			
	dynamic	static	Total
CPU	0,06679	0,01521	0,08200
Disk	0,00000	0,01200	0,01200
LAN	0,00320	0,00000	0,00320
Delay	0,23560	0,00000	0,23560
Internet	0,51538	0,35963	0,87500

21.10.2004

Performance by Design

14

Alternative 3 (Komprimierung)

• Service Demands:

Service Demands (Alternative 1a)		
	Dynamic	Static
CPU	0,2268	0,0407
Disk	0	0,0292
LAN	0,0068	0
Delay	0,5	0
INet	0,4375	0,4375 = $0,875 * 0,5$

Residence Time		
	dynamic	static
CPU	0,26692	0,04790
Disk	0,00000	0,02955
LAN	0,00683	0,00000
Delay	0,50000	0,00000
Internet	0,43750	0,43750
Response Time	1,21125	0,51495

Utilization			
	dynamic	static	Total
CPU	0,13359	0,01673	0,15031
Disk	0,00000	0,01200	0,01200
LAN	0,00401	0,00000	0,00401
Delay	0,29450	0,00000	0,29450
Internet	0,25769	0,17981	0,43750

21.10.2004

Performance by Design

15

Alternative 4 (Clustering)

• Service Demands:

Service Demands (Alternative 1a)		
	Dynamic	Static
CPU (MP2)	0,13608	0,0444
Disk (MP2)	0	0,0292
LAN	0,0068	0
Delay	0,5	0
INet	0,875	0,875

Residence Time		
	dynamic	static
CPU	0,07196	0,02335
Disk	0,00000	0,01469
LAN	0,00680	0,00000
Delay	0,50000	0,00000
Internet	0,87500	0,87500
Response Time	1,45336	0,91304

Utilization			
	dynamic	static	Total
CPU	0,06679	0,01521	0,08200
Disk	0,00000	0,01200	0,01200
LAN	0,00320	0,00000	0,00320
Delay	0,23560	0,00000	0,23560
Internet	0,51538	0,35963	0,87500

21.10.2004

Performance by Design

16

Entwurfsempfehlung

- (bei mir:) Alternative 3, Komprimierung
 - es bleibt zu messen wie lange die Komprimierung der Dateien tatsächlich dauert und wie stark die Dateien komprimiert werden können

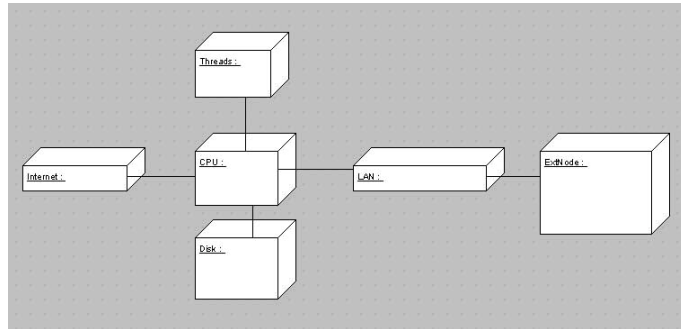
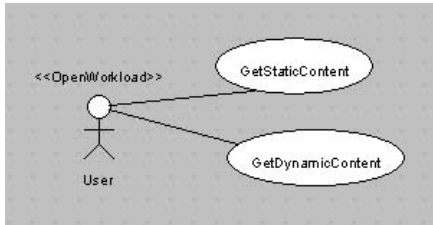
21.10.2004

Performance by Design

17

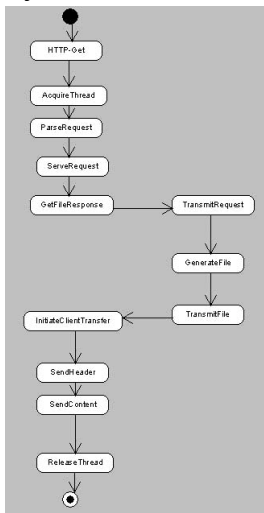
umlPSI-Lösung

Anwendungsfall- und Deploymentdiagramm (bei allen Alternativen gleich):

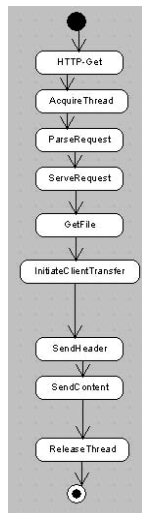


Aktivitätsdiagramme:

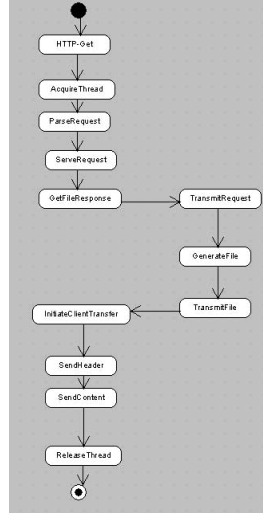
Alternative 0:
dynamisch



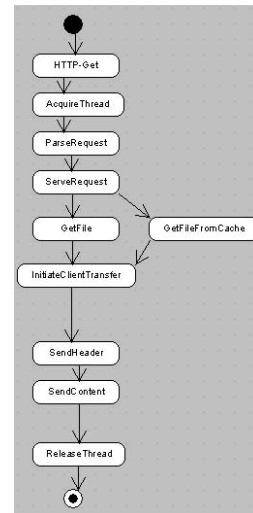
statisch



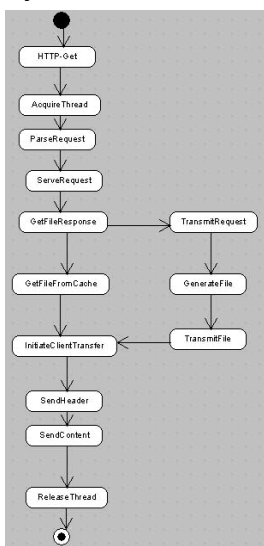
Alternative 1a:
dynamisch



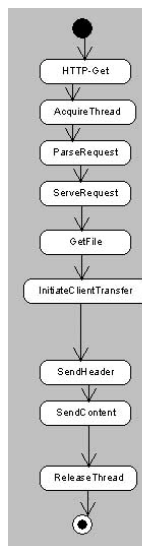
statisch



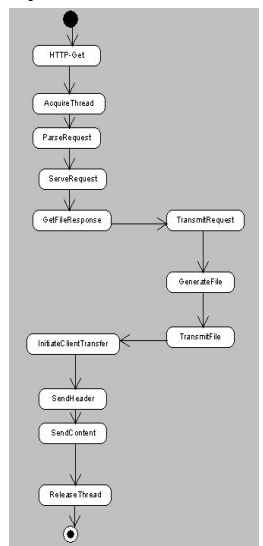
Alternative 1b:
dynamisch



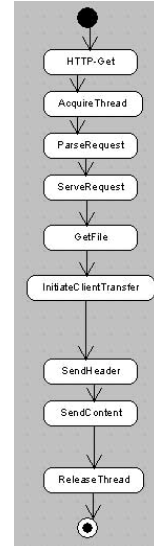
statisch



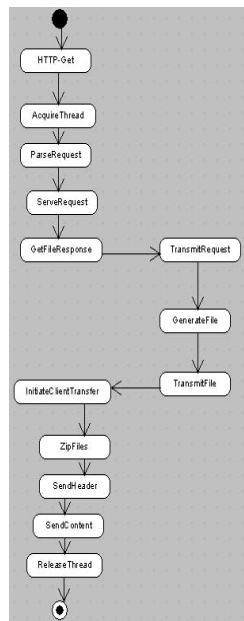
Alternative 2:
dynamisch



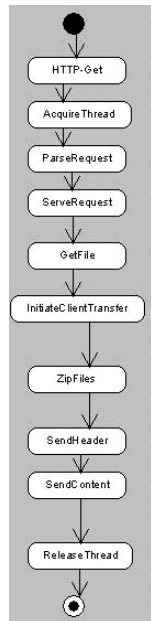
statisch



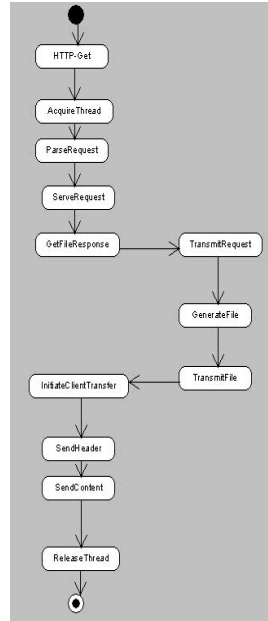
Alternative 3: dynamisch



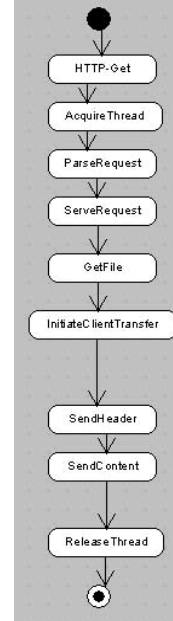
statisch



Alternative 4: dynamisch



statisch



G. Rohdaten der Ergebnisse

Um eine nachträgliche, unabhängige Auswertung der Ergebnisse des Experimente zu ermöglichen, wurden die Rohdaten jedes Teilnehmers in diese Ausarbeitung integriert. Für jedes Verfahren werden die Durchlaufzeiten und Auslastungen für jede Alternative angegeben. Beim SPE-Verfahren wird genau eine Durchlaufzeit angegeben, beim CP-Verfahren wird zusätzlich die Zeit angegeben, die auf jeder Ressource verbracht wird. Die Ausgaben des umlPSI-Werkzeugs zur Simulation der UML-Diagramme schließen sich an. Dabei wird für jede Aktivität eine Durchlaufzeit angegeben. Farblich hervorgehoben sind dort die Gesamtdurchlaufzeiten (blau) für die beiden Anwendungsfälle (statische bzw. dynamische Seitenabfragen) und die Ressourcenauslastungen (gelb). Diese Werte werden auch in den Auswertungsdiagrammen im Hauptteil der Ausarbeitung verwendet.

SPE-Ergebnisse

Alternative	Residence Time					Utilization					
	Total	CPU	Disk	INet	Delay	LAN	CPU	Disk	INet	Delay	LAN
Teilnehmer 1	0	1,1928	0,0120	0,0024	0,8750	0,3000	0,0034	0,0100	0,8800	0,3000	0,0000
	1a	1,1883	0,0092	0,0007	0,8750	0,3000	0,0034	0,0100	0,8800	0,3000	0,0000
	1b	1,1317	0,0116	0,0024	0,8750	0,2400	0,0027	0,0100	0,8800	0,2400	0,0000
	2										
	3	0,8188	0,0130	0,0024	0,5000	0,3000	0,0034	0,0100	0,5000	0,3000	0,0000
Teilnehmer 2	4	1,1952	0,0145	0,0024	0,8750	0,4000	0,0034	0,0100	0,8800	0,3000	0,0000
	0	2,9415	0,9060	0,0412	2,5000	0,3000	0,0097	0,0800	0,0400	2,5000	0,0100
	1a	2,9338	0,0848	0,0373	2,5000	0,3000	0,0177	0,0800	0,0400	2,5000	0,0100
	1b	2,8670	0,1294	0,0619	2,4250	0,2400	0,0107	0,0110	0,0600	2,4200	0,0000
	2										
Teilnehmer 3	3	1,6856	0,0846	0,0412	1,2500	0,3000	0,0097	0,0800	0,0400	1,2500	0,0100
	4	2,9367	0,0866	0,0404	2,5000	0,3000	0,0097	0,0400	0,0200	2,5000	0,0100
	0	0,9562	0,0187	0,0096	0,6250	0,3000	0,0029	0,1000	0,0500	3,1300	1,5000
	1a	0,9499	0,0157	0,0062	0,6250	0,3000	0,0029	0,0800	0,0300	3,1300	1,5000
	1b	0,8981	0,0211	0,0096	0,6250	0,2400	0,0024	0,1100	0,0500	3,1300	1,2000
Teilnehmer 4	2	0,9546	0,0177	0,0096	0,6250	0,3000	0,0029	0,1000	0,0500	3,1300	1,5000
	3	0,7096	0,0221	0,0096	0,3750	0,3000	0,0029	0,1200	0,0500	1,8800	1,5000
	4	0,9588	0,0213	0,0096	0,6250	0,3000	0,0029	0,0500	0,0200	3,1300	1,5000
	0	1,3168	0,0120	0,0024	1,0000	0,3000	0,0024	0,0100	0,0000	1,0000	0,0024
	1a	1,3108	0,0077	0,0007	1,0000	0,3000	0,0024	0,0100	0,0000	1,0000	0,0000
	1b	1,2545	0,0102	0,0024	1,0000	0,2400	0,0019	0,0100	0,0000	1,0000	0,0000
	2	1,3482	0,0312	0,0146	1,0000	0,3000	0,0024	0,0300	0,0100	1,0000	0,0000
	3	0,8766	0,0093	0,0024	0,5625	0,3000	0,0024	0,0100	0,0000	0,5600	0,0000
	4	1,3152	0,0104	0,0024	1,0000	0,3000	0,0024	0,0100	0,0000	1,0000	0,0000

Alternative	Residence Time					Utilization					
	Total	CPU	Disk	Inet	Delay	LAN	CPU	Disk	Inet	Delay	LAN
Teilnehmer 5	0	1,5503	0,1633	0,0846	1,0000	0,3000	0,0024	0,0800	1,0000	0,3000	0,0000
	1a	1,5196	0,1424	0,0748	1,0000	0,3000	0,0024	0,0700	1,0000	0,3000	0,0000
	1b	1,4868	0,1603	0,0846	1,0000	0,2400	0,0019	0,0800	1,0000	0,2400	0,0000
	2	1,5477	0,1607	0,0846	1,0000	0,3000	0,0024	0,0800	1,0000	0,3000	0,0000
	3	1,2979	0,1629	0,0846	0,7500	0,3000	0,0024	0,0800	0,7500	0,3000	0,0000
	4	1,5323	0,1488	0,0812	1,0000	0,3000	0,0024	0,0400	1,0000	0,3000	0,0000
Teilnehmer 6	0	1,8476	0,0351	0,0120	1,5000	0,3000	0,0005	0,0000	0,0200	0,0000	0,0000
	1a	1,8204	0,0163	0,0036	1,5000	0,3000	0,0005	0,0000	0,0200	0,0000	0,0000
	1b	1,7826	0,0302	0,0120	1,5000	0,2000	0,0004	0,0000	0,0200	0,0000	0,0000
	2	1,8428	0,0303	0,0120	1,5000	0,3000	0,0005	0,0000	0,0200	0,0000	0,0000
	3	1,1221	0,0476	0,0240	0,7500	0,3000	0,0005	0,0000	0,0100	0,0000	0,0000
	4	1,8421	0,0296	0,0120	1,5000	0,3000	0,0005	0,0000	0,0200	0,0000	0,0000
Teilnehmer 7	0	2,3094	0,0424	0,0198	1,1250	1,1200	0,0022	0,0200	1,1300	1,2000	0,0000
	1a	2,3030	0,0376	0,0182	1,1250	1,1200	0,0022	0,0200	1,1300	1,2000	0,0000
	1b	2,1478	0,0411	0,0198	1,1250	0,9600	0,0020	0,0200	1,1300	0,9600	0,0000
	2	2,3944	0,0450	0,0221	1,1250	1,2000	0,0022	0,0200	1,1300	1,2000	0,0000
	3	1,8822	0,0533	0,0271	1,0000	0,8000	0,0018	0,0300	1,0000	0,8000	0,0000
	4	1,9892	0,0416	0,0208	1,1250	0,8000	0,0018	0,0200	1,1300	0,8000	0,0000
Teilnehmer 8	0	0,6845	0,0132	0,0024	0,3515	0,3164	0,0010	0,0020	0,3510	0,3160	0,0010
	1a	0,6200	0,0132	0,0007	0,3515	0,2588	0,0010	0,0010	0,3510	0,2590	0,0010
	1b	0,5800	0,0186	0,0024	0,3410	0,2200	0,0080	0,0020	0,3410	0,2290	0,0010
	2										
	3	0,6089	0,0220	0,0024	0,2671	0,3164	0,0010	0,0020	0,2620	0,3160	0,0010
	4	0,6878	0,0038	0,0011	0,3656	0,3156	0,0017	0,0040	0,0010	0,3280	0,0020

Teilnehmer 4	Alternative 0		Alternative 1a		Alternative 1b		Alternative 2		Alternative 3		Alternative 4	
	Residence Time	dynamic static	Residence Time	dynamic static	Residence Time	dynamic static	Residence Time	dynamic static	Residence Time	dynamic static	Residence Time	dynamic static
CPU	0.12342	0.04050	0.12342	0.04050	0.12342	0.04050			0.19379	0.06359	0.05907	0.01938
Disk	0.00000	0.02935	0.00000	0.00873	0.00000	0.02935			0.00000	0.02935	0.00000	0.01459
LAN	0.00682	0.00000	0.00682	0.00000	0.00545	0.00000			0.00682	0.00000	0.00682	0.00000
Delay	0.50000	0.00000	0.50000	0.00000	0.40000	0.00000			0.50000	0.00000	0.50000	0.00000
Internet	0.87500	0.87500	0.87500	0.87500	0.87500	0.87500			0.70000	0.70000	0.87500	0.87500
Response	1.50524	0.94485	1.50524	0.92423	1.40387	0.94485			1.40061	0.79294	1.44089	0.90897
	Utilization		Utilization		Utilization		Utilization		Utilization		Utilization	
	dynamic static	Total	dynamic static	Total	dynamic static	Total	dynamic static	Total	dynamic static	Total	dynamic static	Total
CPU	0.06673	0.01528	0.06673	0.01528	0.06673	0.01528	0.08201		0.10010	0.02292	0.03337	0.00764
Disk	0.00000	0.01192	0.00000	0.00358	0.00000	0.01192	0.01192		0.00000	0.01192	0.00000	0.00596
LAN	0.00400	0.00000	0.00400	0.00000	0.00400	0.00000	0.00320		0.00400	0.00000	0.00400	0.00000
Delay	0.29450	0.00000	0.29450	0.00000	0.23560	0.00000	0.23560		0.29450	0.00000	0.29450	0.00000
Internet	0.51538	0.35963	0.51538	0.35963	0.51538	0.35963	0.87500		0.41230	0.28770	0.51538	0.35963
	0.87500	0.87500	0.87500	0.87500	0.87500	0.87500	0.87500		0.70000	0.70000	0.87500	0.87500
Teilnehmer 5	Alternative 0		Alternative 1a		Alternative 1b		Alternative 2		Alternative 3		Alternative 4	
	Residence Time	dynamic static	Residence Time	dynamic static	Residence Time	dynamic static	Residence Time	dynamic static	Residence Time	dynamic static	Residence Time	dynamic static
CPU	0.12329	0.04043	0.12329	0.04043	0.12329	0.04043	0.10998	0.03607	0.05901	0.01935		
Disk	0.00000	0.02962	0.00000	0.00881	0.00000	0.02962	0.00000	0.02663	0.00000	0.02962	0.00000	0.01472
LAN	0.00681	0.00000	0.00681	0.00000	0.00544	0.00000	0.00612	0.00000	0.00681	0.00000	0.00681	0.00000
Delay	0.50000	0.00000	0.50000	0.00000	0.40000	0.00000	0.00000	0.00000	0.50000	0.00000	0.50000	0.00000
Internet	0.87500	0.87500	0.87500	0.87500	0.87500	0.87500	0.87500	0.87500	0.51625	0.51625	0.87500	0.87500
Response	1.50510	0.94506	1.50510	0.92424	1.40373	0.94506	1.49110	0.93770	1.98167	0.86023	1.44082	0.90907
	Utilization		Utilization		Utilization		Utilization		Utilization		Utilization	
	dynamic static	Total	dynamic static	Total	dynamic static	Total	dynamic static	Total	dynamic static	Total	dynamic static	Total
CPU	0.06667	0.01526	0.06667	0.01526	0.06667	0.01526	0.06000	0.01373	0.33334	0.07628	0.03333	0.00763
Disk	0.00000	0.01203	0.00000	0.00361	0.00000	0.01203	0.00000	0.01083	0.00000	0.01203	0.00000	0.00601
LAN	0.00399	0.00000	0.00399	0.00000	0.00319	0.00000	0.00359	0.00000	0.00399	0.00000	0.00399	0.00000
Delay	0.29450	0.00000	0.29450	0.00000	0.23560	0.00000	0.29450	0.00000	0.29450	0.00000	0.29450	0.00000
Internet	0.51538	0.35963	0.51538	0.35963	0.51538	0.35963	0.87500		0.30407	0.21218	0.51538	0.35963
	0.87500	0.87500	0.87500	0.87500	0.87500	0.87500	0.87500		0.51625	0.51625	0.87500	0.87500
Teilnehmer 6	Alternative 0		Alternative 1a		Alternative 1b		Alternative 2		Alternative 3		Alternative 4	
	Residence Time	dynamic static	Residence Time	dynamic static	Residence Time	dynamic static	Residence Time	dynamic static	Residence Time	dynamic static	Residence Time	dynamic static
CPU	0.12341	0.04019	0.12341	0.04019	0.12341	0.04019	0.11008	0.03585	0.16484	0.05369	0.05907	0.01924
Disk	0.00000	0.02945	0.00000	0.00876	0.00000	0.02945	0.00000	0.03244	0.00000	0.02945	0.00000	0.01464
LAN	0.00682	0.00000	0.00682	0.00000	0.00545	0.00000	0.00682	0.00000	0.00682	0.00000	0.00682	0.00000
Delay	0.50000	0.00000	0.50000	0.00000	0.40000	0.00000	0.50000	0.00000	0.50000	0.00000	0.50000	0.00000
Internet	0.87500	0.87500	0.87500	0.87500	0.87500	0.87500	0.87500	0.87500	0.43750	0.43750	0.87500	0.87500
Response	1.50522	0.94464	1.50522	0.92395	1.40386	0.94464	1.49190	0.94329	1.10916	0.52064	1.44089	0.90888
	Utilization		Utilization		Utilization		Utilization		Utilization		Utilization	
	dynamic static	Total	dynamic static	Total	dynamic static	Total	dynamic static	Total	dynamic static	Total	dynamic static	Total
CPU	0.06673	0.01517	0.06673	0.01517	0.06673	0.01517	0.06006	0.01365	0.08675	0.01972	0.03337	0.00758
Disk	0.00000	0.01196	0.00000	0.00359	0.00000	0.01196	0.00000	0.01316	0.00000	0.01196	0.00000	0.00598
LAN	0.00400	0.00000	0.00400	0.00000	0.00320	0.00000	0.00400	0.00000	0.00400	0.00000	0.00400	0.00000
Delay	0.29450	0.00000	0.29450	0.00000	0.23560	0.00000	0.29450	0.00000	0.29450	0.00000	0.29450	0.00000
Internet	0.51538	0.35963	0.51538	0.35963	0.51538	0.35963	0.87500		0.25769	0.17981	0.51538	0.35963
	0.87500	0.87500	0.87500	0.87500	0.87500	0.87500	0.87500		0.43750	0.43750	0.87500	0.87500

umlPSI - Simulationsergebnisse

Teilnehmer 1, Ankunftsrate: 2,0 s, = Auslastung, = Durchlaufzeit

= Durchlaufzeit

= Durchlaufzeit

[illegible]

H. Diagramme zur Veranschaulichung

Einige Visualisierungen der Ergebnisse wurden zur besseren Übersichtlichkeit nicht in den Hauptteil der Ausarbeitung integriert und sollen nun folgen.

Zur Darstellung werden u.a. so genannte *Boxplots* verwendet. Diese eignen sich gut, um die Verteilung von Werten einer Stichprobe zu illustrieren. Auf der x-Achse der Boxplots befinden sich die verschiedenen Entwurfsalternativen und auf der y-Achse die Durchlaufzeiten, die von den einzelnen Teilnehmern berechnet wurden. Dabei sind die Punkte mit gleicher Farbe die Vorhersagen eines einzelnen Teilnehmers. Ein schwarzer Balken mit einer rechts daneben stehenden Zahl gibt bei jedem Datensatz den Median an, also die Grenze zwischen der oberen und der unteren Hälfte der Datenpunkte. Kästen um den Median herum markieren jeweils den Bereich, in dem die mittlere Hälfte der Daten liegt, d.h. ein Viertel der Datenpunkte liegt unterhalb der Kästen und ein Viertel oberhalb.

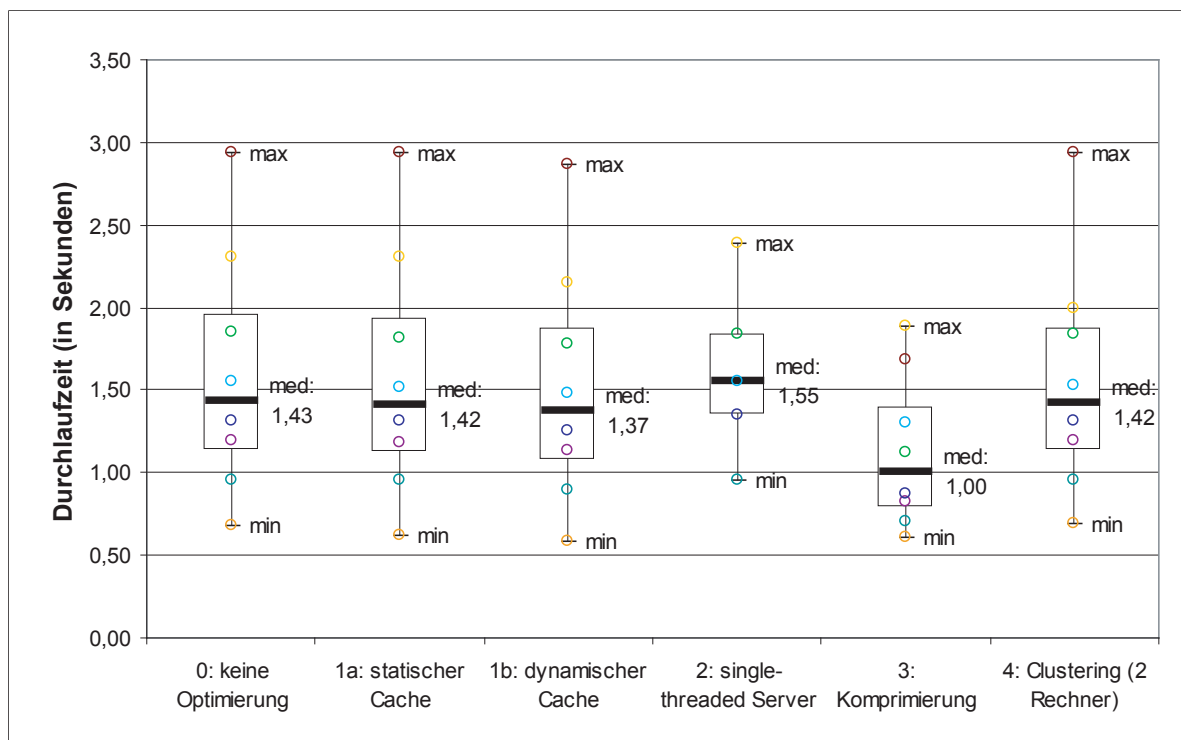


Abbildung H.1.: Boxplot der Durchlaufzeiten beim SPE-Verfahren

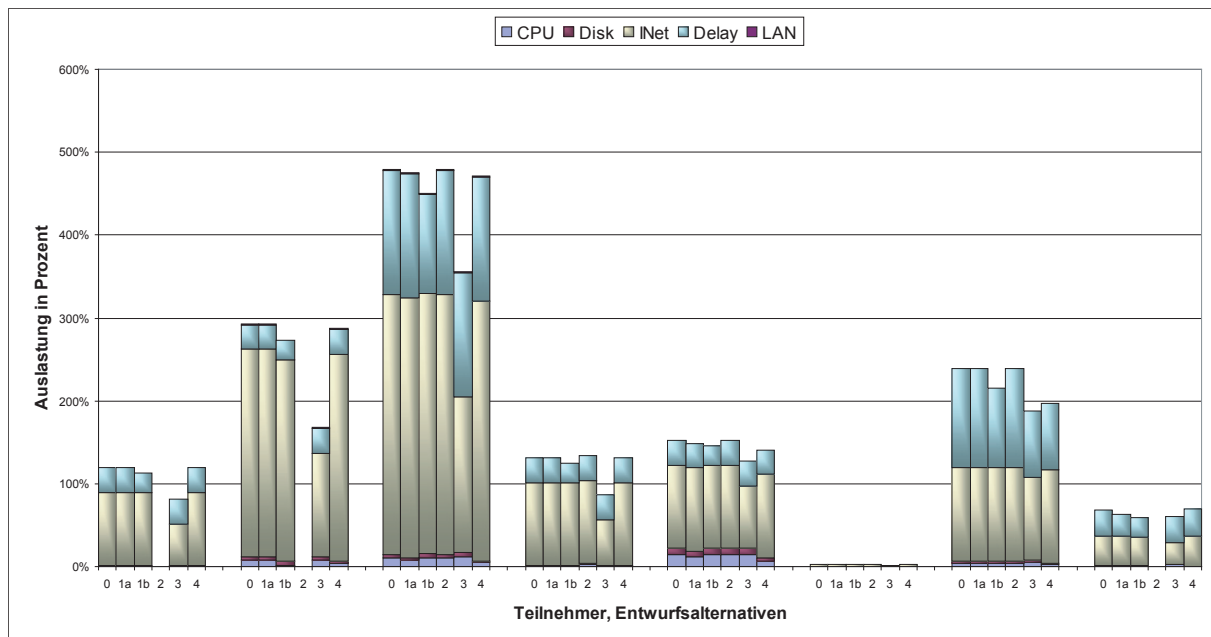


Abbildung H.2.: Vorhergesagte Auslastungen beim SPE-Verfahren

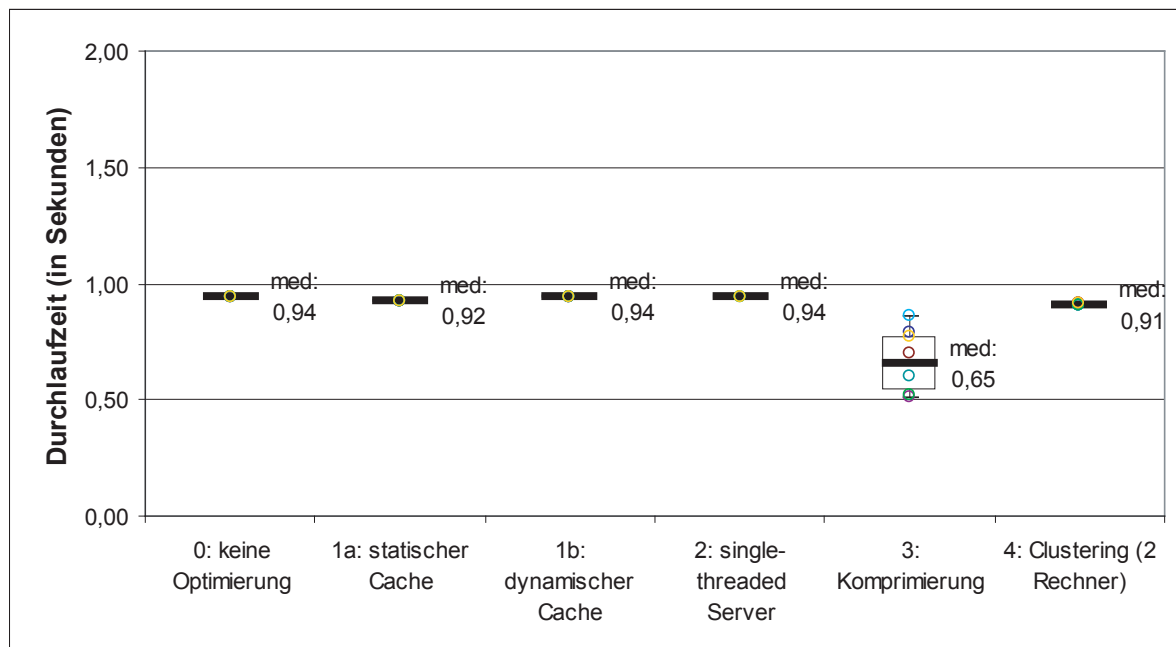


Abbildung H.3.: Boxplot der Durchlaufzeiten beim CP-Verfahren (statische Anfragen)

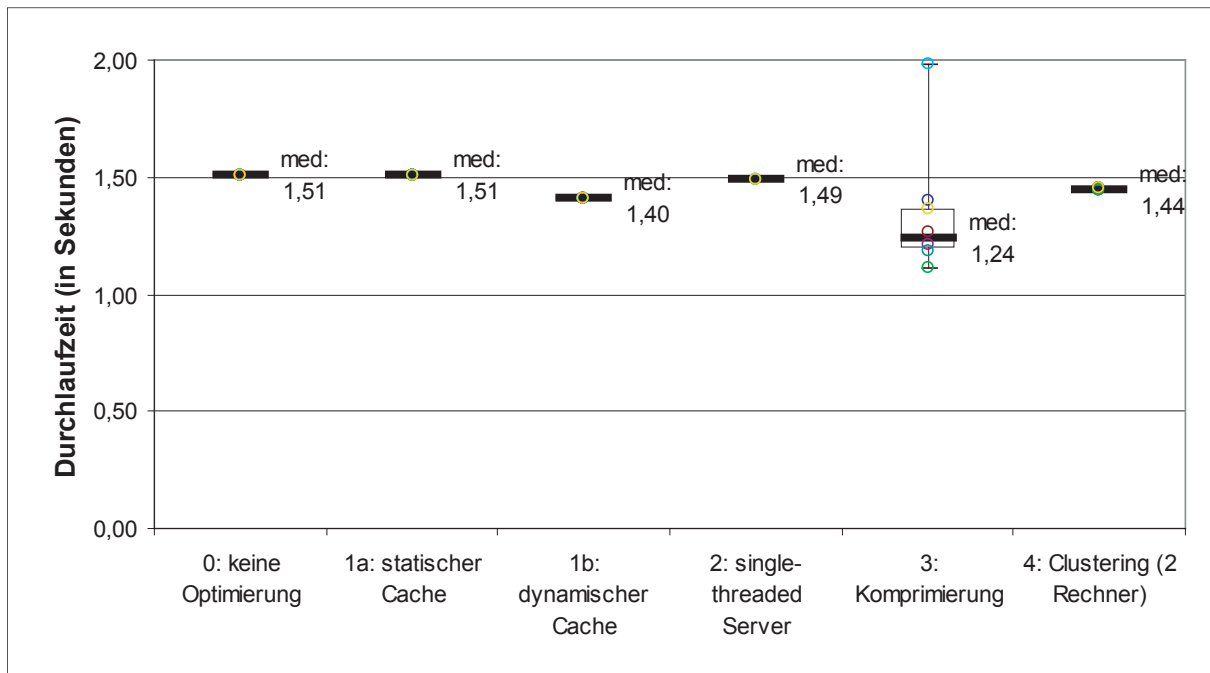


Abbildung H.4.: Boxplot der Durchlaufzeiten beim CP-Verfahren (dynamische Anfragen)

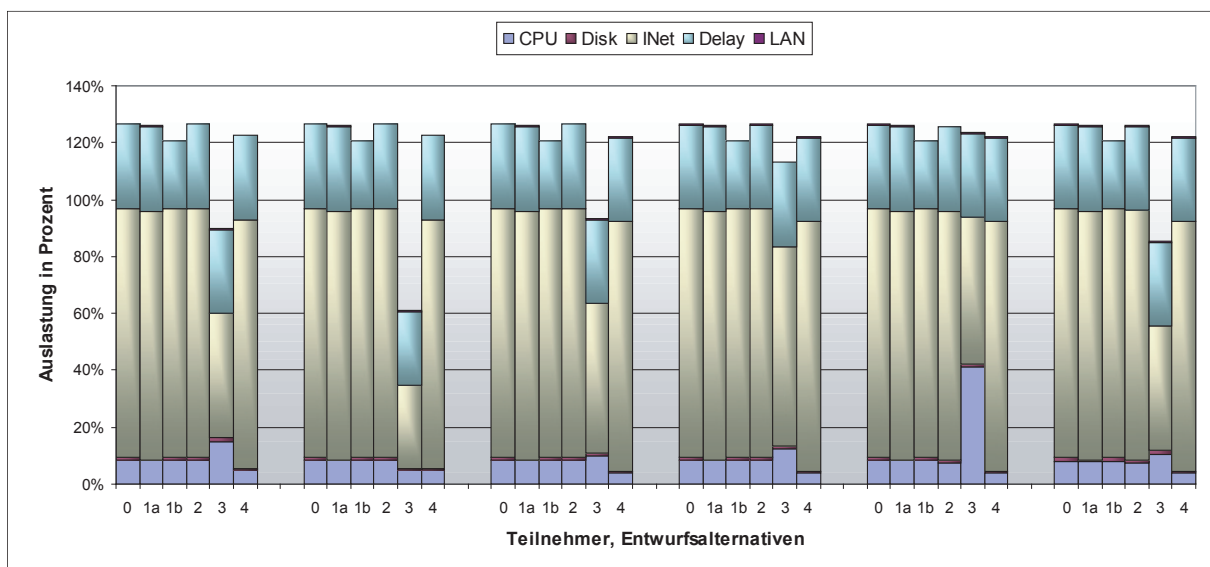


Abbildung H.5.: Vorhergesagte Auslastungen beim CP-Verfahren

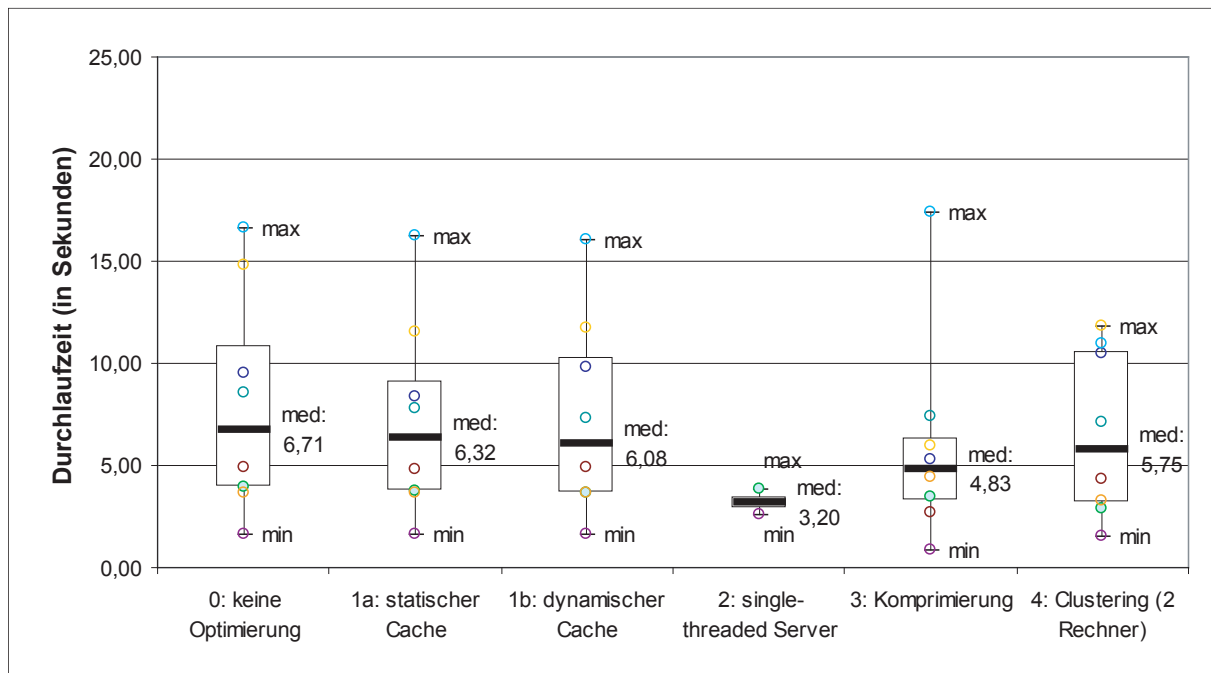


Abbildung H.6.: Boxplot der Durchlaufzeiten beim umlPSI-Verfahren (statische Anfragen)

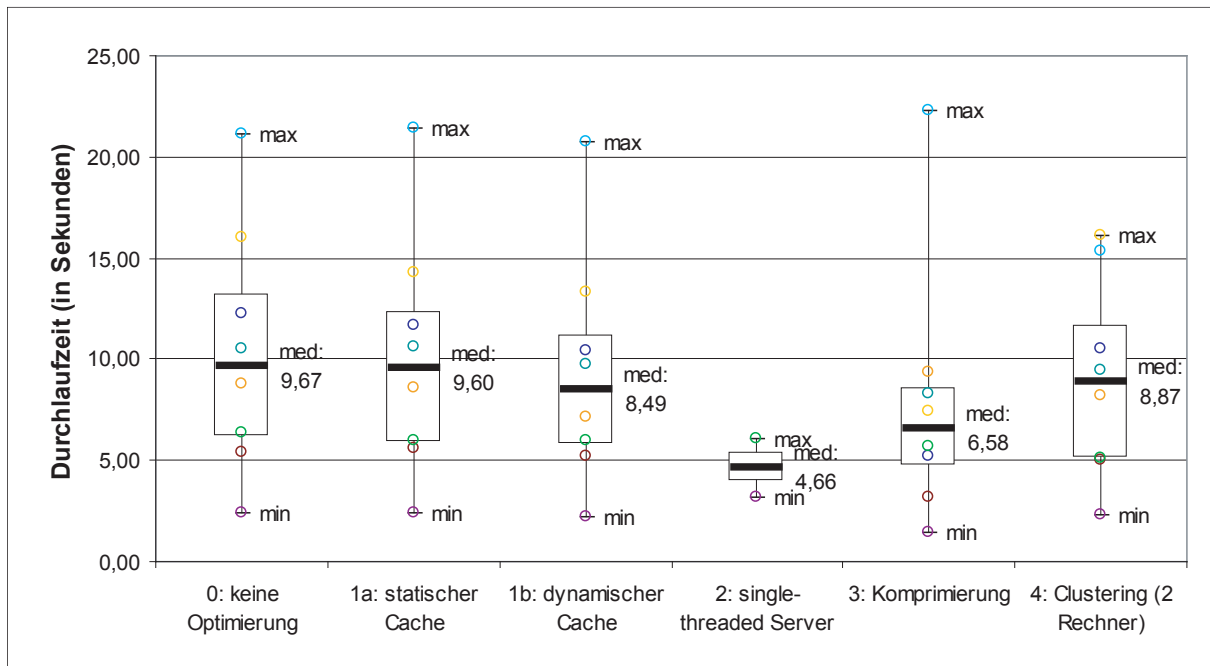


Abbildung H.7.: Boxplot der Durchlaufzeiten beim umlPSI-Verfahren (dynamische Anfragen)

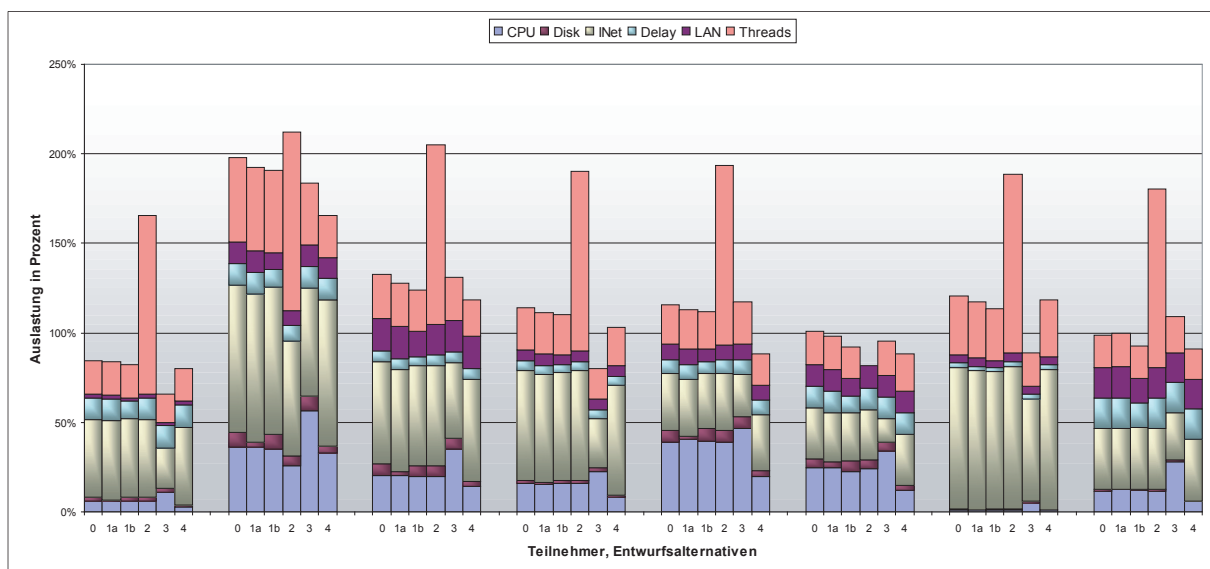


Abbildung H.8.: Vorhergesagte Auslastungen beim umlPSI-Verfahren

I. Messungen am Prototypen

Die Performance-Messung des im Experiment untersuchten Webservers wird in Abschnitt [5.6](#) näher erläutert. Es folgen die Rohdaten dieser Messungen. Die ausgewerteten Ergebnisse finden sich in Abschnitt [6.1.4](#).

Performance-Messung am Webserver

Alternative 0: keine Optimierung

Dynamische Abfrage:

http://localhost:8000/suche.htm?title=&author=mann&select=nur+Autor&Submit=suchen

	Average Duration	Minimum Duration	Maximum Duration	Average Speed	Users	Time (HH:MM:SS)	Average Size (kBytes)
Total Repeats(seconds)	(seconds)	(seconds)	(seconds)	(kBytes/sec)			
3	1.3070	1.4940	1.5340	3,7620	1	00:00:10	5,6690
6	1.5310	1.4950	1.6050	3,7030	1	00:00:20	5,6690
11	1.4900	1.4550	1.5050	3,8050	1	00:00:30	5,6690
15	1.5220	1.4950	1.5650	3,7250	1	00:00:40	5,6690
18	1.4970	1.4940	1.5050	3,7870	1	00:00:50	5,6690
23	1.5160	1.4940	1.5850	3,7390	1	00:01:00	5,6690
27	1.5050	1.4950	1.5150	3,7670	1	00:01:10	5,6690
30	1.5080	1.5050	1.5140	3,7590	1	00:01:20	5,6690
35	1.5130	1.4950	1.5270	3,7470	1	00:01:30	5,6690
39	1.5130	1.5040	1.5390	3,7470	1	00:01:40	5,6690
42	1.5090	1.4950	1.5190	3,7570	1	00:01:50	5,6690
47	1.5040	1.4940	1.5250	3,7690	1	00:02:00	5,6690
51	1.5030	1.4950	1.5150	3,7720	1	00:02:10	5,6690
54	1.5040	1.4940	1.5250	3,7690	1	00:02:20	5,6690
59	1.5140	1.4940	1.5400	3,7440	1	00:02:30	5,6690
63	1.5020	1.4940	1.5250	3,7740	2	00:02:40	5,6690
69	1.5160	1.5000	1.5260	3,7390	2	00:02:50	5,6690
78	1.5320	1.4990	1.6830	3,6140	2	00:03:00	5,5350
87	1.5300	1.4980	1.7200	3,7050	2	00:03:10	5,6690
93	1.5120	1.5050	1.5200	3,7490	2	00:03:20	5,6690
101	1.5160	1.5020	1.5210	3,7390	2	00:03:30	5,6690
111	1.5080	1.4950	1.5240	3,7590	2	00:03:40	5,6690
117	1.5140	1.5090	1.5160	3,7440	2	00:03:50	5,6690
125	1.5120	1.4990	1.5190	3,7490	2	00:04:00	5,6690
135	1.5080	1.4950	1.5200	3,7590	2	00:04:10	5,6690
141	1.5130	1.5080	1.5150	3,7470	2	00:04:20	5,6690
149	1.5120	1.5040	1.5180	3,7490	2	00:04:30	5,6690
159	1.5090	1.4860	1.5170	3,7570	2	00:04:40	5,6690
165	1.4980	1.4890	1.5080	3,7840	2	00:04:50	5,6690
173	1.5010	1.4920	1.5120	3,7770	2	00:05:00	5,6690
Mittelwerte:	1,5106	1,4958	1,5387	3,7499			5,6645

Statische Abfrage:

http://localhost:8000/1.html

	Average Duration	Minimum Duration	Maximum Duration	Average Speed	Users	Time (HH:MM:SS)	Average Size (kBytes)
Total Repeats(seconds)	(seconds)	(seconds)	(seconds)	(kBytes/sec)			
1	0,9790	0,9790	0,9790	5,7490	1	00:00:10	5,6280
3	0,9660	0,9660	0,9660	5,8260	1	00:00:20	5,6280
4	0,9370	0,9370	0,9370	6,0060	1	00:00:30	5,6280
5	0,9760	0,9760	0,9760	5,7660	1	00:00:40	5,6280
7	0,9600	0,9550	0,9660	5,8630	1	00:00:50	5,6280
8	0,9980	0,9980	0,9980	5,6390	1	00:01:00	5,6280
9	0,9650	0,9650	0,9650	5,8320	1	00:01:10	5,6280
11	0,9750	0,9660	0,9850	5,7720	1	00:01:20	5,6280
12	0,9550	0,9550	0,9550	5,8930	1	00:01:30	5,6280
13	0,9660	0,9660	0,9660	5,8260	1	00:01:40	5,6280
15	0,9650	0,9640	0,9660	5,8320	1	00:01:50	5,6280
16	0,9650	0,9650	0,9650	5,8320	1	00:02:00	5,6280
17	0,9670	0,9670	0,9670	5,8200	1	00:02:10	5,6280
19	0,9760	0,9660	0,9870	5,7660	1	00:02:20	5,6280
20	0,9860	0,9860	0,9860	5,7080	1	00:02:30	5,6280
21	0,9660	0,9660	0,9660	5,8260	2	00:02:40	5,6280
25	0,9840	0,9560	1,0390	5,7200	2	00:02:50	5,6280
27	0,9600	0,9550	0,9650	5,8630	2	00:03:00	5,6280
29	0,9850	0,9850	0,9860	5,7140	2	00:03:10	5,6280
33	0,9810	0,9660	1,0180	5,7370	2	00:03:20	5,6280
35	0,9740	0,9660	0,9820	5,7780	2	00:03:30	5,6280
37	0,9660	0,9560	0,9760	5,8260	2	00:03:40	5,6280
41	0,9730	0,9660	0,9760	5,7840	2	00:03:50	5,6280
43	1,0170	0,9660	1,0680	5,5340	2	00:04:00	5,6280
45	0,9710	0,9650	0,9760	5,7860	2	00:04:10	5,6280
49	0,9920	0,9560	1,0690	5,6730	2	00:04:20	5,6280
51	0,9700	0,9660	0,9750	5,8020	2	00:04:30	5,6280
53	0,9920	0,9650	1,0190	5,6730	2	00:04:40	5,6280
57	0,9690	0,9560	0,9790	5,8080	2	00:04:50	5,6280
59	0,9630	0,9450	0,9810	5,8440	2	00:05:00	5,6280
Mittelwerte:	0,9733	0,9649	0,9846	5,7836			5,6280

Statische Abfrage:

http://localhost:8000/tux.jpg

	Average Duration	Minimum Duration	Maximum Duration	Average Speed	Users	Time (HH:MM:SS)	Average Size (kBytes)
Total Repeats(seconds)	(seconds)	(seconds)	(seconds)	(kBytes/sec)			
1	0,9890	0,9890	0,9890	5,6750	1	00:00:10	5,6130
3	0,9890	0,9890	0,9890	5,6750	1	00:00:20	5,6130
4	0,9890	0,9890	0,9890	5,6750	1	00:00:30	5,6130
5	0,9800	0,9800	0,9800	5,7280	1	00:00:40	5,6130
7	0,9890	0,9890	0,9900	5,6750	1	00:00:50	5,6130
8	0,9880	0,9880	0,9880	5,6810	1	00:01:00	5,6130
9	0,9890	0,9890	0,9890	5,6750	1	00:01:10	5,6130
11	1,0130	0,9890	1,0380	5,5410	1	00:01:20	5,6130
12	0,9950	0,9950	0,9950	5,6410	1	00:01:30	5,6130
13	0,9890	0,9890	0,9890	5,6750	1	00:01:40	5,6130
15	1,0070	0,9790	1,0360	5,5740	1	00:01:50	5,6130
16	1,0700	1,0700	1,0700	5,2460	1	00:02:00	5,6130
17	0,9790	0,9790	0,9790	5,7330	1	00:02:10	5,6130
19	1,0040	0,9890	1,0190	5,5910	1	00:02:20	5,6130
20	1,0090	1,0090	1,0090	5,5630	1	00:02:30	5,6130
21	0,9790	0,9790	0,9790	5,7330	2	00:02:40	5,6130
25	1,0070	0,9980	1,0270	5,5740	2	00:02:50	5,6130
27	0,9980	0,9970	0,9990	5,6240	2	00:03:00	5,6130
29	1,1680	1,0950	1,2420	4,8060	2	00:03:10	5,6130
32	1,0180	0,9890	1,0640	5,5140	2	00:03:20	5,6130
35	1,0000	0,9980	1,0060	5,6130	2	00:03:30	5,6130
37	1,0050	0,9980	1,0130	5,5850	2	00:03:40	5,6130
39	1,0020	0,9990	1,0060	5,6020	2	00:03:50	5,6130
43	1,0040	0,9980	1,0210	5,5910	2	00:04:00	5,6130
45	1,0060	1,0060	1,0070	5,5800	2	00:04:10	5,6130
47	1,0050	0,9970	1,0140	5,5850	2	00:04:20	5,6130
51	0,9990	0,9980	1,0010	5,6190	2	00:04:30	5,6130
53	1,0160	0,9790	1,0530	5,5250	2	00:04:40	5,6130
55	0,9720	0,9720	0,9730	5,7750	2	00:04:50	5,6130
59	0,9950	0,9860	0,9990	5,6410	2	00:05:00	5,6130
Mittelwerte:	1,0051	0,9967	1,0151	5,5905			5,6130

Auslastung:

	Minimum	Durchschnitt	Maximum
CPU	0,00%	7,36%	54,00%
Disk	0,00%	0,28%	13,84%

Alternative 1a: statischer Cache

Dynamische Abfrage:

http://localhost:8000/suche.htm?title=&author=mann&select=nur+Autor&Submit=suchen

	Average Duration	Minimum Duration	Maximum Duration	Average Speed	Users	Time (HH:MM:SS)	Average Size (kBytes)
Total Repeats(seconds)	(seconds)	(seconds)	(seconds)	(kBytes/sec)			
2	2,3010	1,5050	3,0970	2,4640	1	00:00:10	5,6690
6	1,5150	1,4950	1,5350	3,7420	1	00:00:20	5,6690
9	1,5250	1,5050	1,5450	3,7170	1	00:00:30	5,6690
14	1,5000	1,4940	1,5080	3,7790	1	00:00:40	5,6690
18	1,5160	1,4950	1,5310	3,7390	1	00:00:50	5,6690
21	1,5020	1,4990	1,5050	3,7740	1	00:01:00	5,6690
26	1,4930	1,4820	1,5050	3,7970	1	00:01:10	5,6690
30	1,5100	1,4950	1,5250	3,7540	1	00:01:20	5,6690
33	1,5010	1,4950	1,5050	3,7770	1	00:01:30	5,6690
38	1,5020	1,4840	1,5240	3,7740	1	00:01:40	5,6690
42	1,4990	1,4850	1,5150	3,7820	1	00:01:50	5,6690
45	1,5020	1,4940	1,5090	3,7740	1	00:02:00	5,6690
50	1,4840	1,4450	1,5040	3,8200	1	00:02:10	5,6690
54	1,4840	1,4840	1,4850	3,8200	1	00:02:20	5,6690
57	1,4880	1,4850	1,4950	3,8100	1	00:02:30	5,6690
62	1,4940	1,4840	1,5250	3,7950	1	00:02:40	5,6690
66	1,4890	1,4850	1,4950	3,8070	1	00:02:50	5,6690
71	1,5030	1,4780	1,5260	3,7720	2	00:03:00	5,6690
80	1,5040	1,4660	1,5540	3,7690	2	00:03:10	5,6690
88	1,5170	1,4910	1,5300	3,7370	2	00:03:20	5,6690
95	1,5070	1,4940	1,5490	3,7620	2	00:03:30	5,6690
104	1,5030	1,4850	1,5250	3,7720	2	00:03:40	5,6690
112	1,5070	1,4860	1,5560	3,7620	2	00:03:50	5,6690
119	1,5010	1,4880	1,5120	3,7770	2	00:04:00	5,6690
128	1,5390	1,4350	1,8550	3,6840	2	00:04:10	5,6690
135	1,5140	1,4840	1,5710	3,7440	2	00:04:20	5,6690
143	1,5080	1,4850	1,5470	3,7590	2	00:04:30	5,6690
152	1,5060	1,4850	1,5460	3,7640	2	00:04:40	5,6690
159	1,5150	1,4930	1,5480	3,7420	2	00:04:50	5,6690
167	1,5070	1,4850	1,5340	3,7620	0	00:05:00	5,6690
Mittelwerte:	1,5312	1,4855	1,5887	3,7244			5,6690

Statische Abfrage:

http://localhost:8000/1.html

	Average Duration	Minimum Duration	Maximum Duration	Average Speed	Users	Time (HH:MM:SS)	Average Size (kBytes)
Total Repeats(seconds)	(seconds)	(seconds)	(seconds)	(kBytes/sec)			
1	2,0490	2,0490	2,0490	2,7470	1	00:00:10	5,6280
2	0,9560	0,9560	0,9560	5,8870	1	00:00:20	5,6280
4	0,9760	0,9660	0,9860	5,7660	1	00:00:30	5,6280
5	0,9560	0,9560	0,9560	5,8870	1	00:00:40	5,6280
6	1,0160	1,0160	1,0160	5,5390	1	00:00:50	5,6280
8	0,9660	0,9560	0,9770	5,8260	1	00:01:00	5,6280
9	0,9960	0,9960	0,9960	5,6510	1	00:01:10	5,6280
10	0,9660	0,9660	0,9660	5,8260	1	00:01:20	5,6280
12	0,9560	0,9760	0,9960	5,7660	1	00:01:30	5,6280
13	0,9560	0,9650	0,9650	5,8320	1	00:01:40	5,6280
14	0,9760	0,9760	0,9760	5,7660	1	00:01:50	5,6280
16	0,9660	0,9360	0,9960	5,8260	1	00:02:00	5,6280
17	0,9760	0,9760	0,9760	5,7660	1	00:02:10	5,6280
18	0,9760	0,9760	0,9760	5,7660	1	00:02:20	5,6280
20	0,9730	0,9660	0,9810	5,7840	1	00:02:30	5,6280
21	0,9760	0,9760	0,9760	5,7660	1	00:02:40	5,6280
22	0,9760	0,9760	0,9760	5,7660	1	00:02:50	5,6280
25	1,0060	0,9760	1,0670	5,5940	2	00:03:00	5,6280
28	0,9590	0,9460	0,9760	5,8690	2	00:03:10	5,6280
30	0,9780	0,9720	0,9850	5,7550	2	00:03:20	5,6280
33	0,9800	0,9550	0,9990	5,7430	2	00:03:30	5,6280
36	0,9870	0,9850	0,9890	5,7020	2	00:03:40	5,6280
38	0,9530	0,9870	0,9990	5,6680	2	00:03:50	5,6280
41	0,9880	0,9760	1,0020	5,6960	2	00:04:00	5,6280
43	0,9850	0,9850	0,9860	5,7140	2	00:04:10	5,6280
46	0,9880	0,9860	0,9900	5,6960	2	00:04:20	5,6280
49	0,9860	0,9850	0,9880	5,7080	2	00:04:30	5,6280
51	0,9860	0,9850	0,9870	5,7080	2	00:04:40	5,6280
54	0,9810	0,9760	0,9860	5,7370	2	00:04:50	5,6280
57	0,9880	0,9850	0,9950	5,6960	0	00:05:00	5,6280
Mittelwerte	1,0152	1,0093	1,0223	5,6465			5,6280

Performance-Messung am Webserver

Alternative 1b: dynamischer Cache

Dynamische Abfrage:

http://localhost:8000/suche.htm?title=&author=mann&select=nur+Autor&Submit=suchen

	Average Duration (seconds)	Minimum Duration (seconds)	Maximum Duration (seconds)	Average Speed (kBytes/sec)	Users	Time (HH:MM:SS)	Average Size (kBytes)
Total Repeats (seconds)	3	1.6090	1.5150	1.7970	3,5230	1 00:00:10	5,6690
3	1.5080	1.5040	1.5180	3,7590	5,6690	1 00:00:20	5,6690
6	1.5110	1.5050	1.5270	3,7520	5,6690	1 00:00:30	5,6690
11	1.5120	1.5040	1.5250	3,7490	5,6690	1 00:00:40	5,6690
15	1.5140	1.5050	1.5320	3,7440	5,6690	1 00:00:50	5,6690
18	1.5270	1.5050	1.5850	3,7130	5,6690	1 00:01:00	5,6690
22	1.5050	1.4950	1.5150	3,7670	5,6690	1 00:01:10	5,6690
27	1.4980	1.4950	1.5050	3,7840	5,6690	1 00:01:20	5,6690
30	1.4940	1.4850	1.5040	3,7950	5,6690	1 00:01:30	5,6690
34	1.4990	1.4940	1.5060	3,7820	5,6690	1 00:01:40	5,6690
39	1.5180	1.5050	1.5460	3,7350	5,6690	1 00:01:50	5,6690
42	1.5370	1.5150	1.5750	3,6880	5,6690	1 00:02:00	5,6690
46	1.5060	1.4740	1.5440	3,7640	5,6690	1 00:02:10	5,6690
51	1.5290	1.5110	1.5640	3,7080	5,6690	2 00:02:20	5,6690
58	1.5150	1.4800	1.5790	3,7420	5,6690	2 00:02:30	5,6690
66	1.5220	1.5050	1.5550	3,7250	5,6690	2 00:02:40	5,6690
75	1.5150	1.4950	1.5290	3,7420	5,6690	2 00:02:50	5,6690
82	1.5180	1.4990	1.5540	3,7350	5,6690	2 00:03:00	5,6690
90	1.5440	1.4700	1.8180	3,6720	5,6690	2 00:03:10	5,6690
106	1.5150	1.5050	1.5250	3,7420	5,6690	2 00:03:20	5,6690
114	1.5150	1.5040	1.5300	3,7420	5,6690	2 00:03:30	5,6690
123	1.5170	1.5050	1.5390	3,7370	5,6690	2 00:03:40	5,6690
129	1.5220	1.5050	1.5570	3,7250	5,6690	2 00:03:50	5,6690
138	1.5130	1.4580	1.5950	3,7470	5,6690	2 00:04:00	5,6690
147	1.1040	0.5890	1.6030	4,6300	5,1110	2 00:04:10	5,1110
158	1.0390	0.9660	1.2660	5,4560	5,6690	2 00:04:20	5,6690
166	1.0560	0.9700	1.1200	5,3680	5,6690	2 00:04:30	5,6690
176	1.2650	0.9800	2.3790	4,4810	5,6690	2 00:04:40	5,6690
184	1.0810	0.9940	1.1410	5,2440	5,6690	2 00:04:50	5,6690
195	1.0650	0.9820	1.1210	5,3230	5,6690	0 00:05:00	5,6690
Mittelwerte:	1.4358	1.3806	1.5385	4,0025	5,6504		

Statische Abfrage:

http://localhost:8000/L.html

	Average Duration (seconds)	Minimum Duration (seconds)	Maximum Duration (seconds)	Average Speed (kBytes/sec)	Users	Time (HH:MM:SS)	Average Size (kBytes)
Total Repeats (seconds)	1	1.0490	1.0490	5,3650	5,6280	1 00:00:10	5,6280
3	0.9660	0.9560	0.9760	5,8260	5,6280	1 00:00:20	5,6280
4	0.9650	0.9650	0.9650	5,8320	5,6280	1 00:00:30	5,6280
5	0.9650	0.9650	0.9650	5,8320	5,6280	1 00:00:40	5,6280
7	0.9610	0.9560	0.9660	5,8560	5,6280	1 00:00:50	5,6280
8	0.9750	0.9750	0.9750	5,7720	5,6280	1 00:01:00	5,6280
9	1.0310	1.0310	1.0310	5,4590	5,6280	1 00:01:10	5,6280
11	0.9650	0.9650	0.9660	5,8320	5,6280	1 00:01:20	5,6280
12	0.9770	0.9770	0.9770	5,7600	5,6280	1 00:01:30	5,6280
13	0.9650	0.9650	0.9650	5,8320	5,6280	1 00:01:40	5,6280
15	0.9700	0.9550	0.9850	5,8020	5,6280	1 00:01:50	5,6280
16	0.9710	0.9710	0.9710	5,7960	5,6280	1 00:02:00	5,6280
18	0.9670	0.9560	0.9780	5,8200	5,6280	2 00:02:10	5,6280
21	0.9840	0.9800	1.0020	5,6620	5,6280	2 00:02:20	5,6280
23	0.9740	0.9680	0.9810	5,7780	5,6280	2 00:02:30	5,6280
26	0.9740	0.9690	0.9770	5,7780	5,6280	2 00:02:40	5,6280
29	0.9790	0.9780	0.9800	5,7490	5,6280	2 00:02:50	5,6280
31	0.9920	0.9810	1.0030	5,6730	5,6280	2 00:03:00	5,6280
34	0.9790	0.9790	0.9810	5,7490	5,6280	2 00:03:10	5,6280
37	0.9810	0.9800	0.9820	5,7370	5,6280	2 00:03:20	5,6280
39	0.9740	0.9690	0.9800	5,7780	5,6280	2 00:03:30	5,6280
41	0.9800	0.9790	0.9810	5,7430	5,6280	2 00:03:40	5,6280
45	0.9810	0.9780	0.9860	5,7370	5,6280	2 00:03:50	5,6280
47	1.0020	0.9950	1.0100	5,6170	5,6280	2 00:04:00	5,6280
51	1.0380	1.0640	1.2170	5,4220	5,6280	2 00:04:10	5,6280
53	0.9950	0.9650	1.0250	5,6560	5,6280	2 00:04:20	5,6280
57	1.0210	0.9650	1.1360	5,5120	5,6280	2 00:04:30	5,6280
59	0.9740	0.9530	0.9960	5,7780	5,6280	2 00:04:40	5,6280
63	1.0270	0.9950	1.0990	5,4800	5,6280	2 00:04:50	5,6280
66	1.0210	1.0160	1.0300	5,5120	5,6280	0 00:05:00	5,6280
Mittelwerte:	0.9871	0.9767	1.0045	5,7048	5,6280		

Statische Abfrage:

http://localhost:8000/tux.jpg

	Average Duration (seconds)	Minimum Duration (seconds)	Maximum Duration (seconds)	Average Speed (kBytes/sec)	Users	Time (HH:MM:SS)	Average Size (kBytes)
Total Repeats (seconds)	1	0.9780	0.9780	5,7390	5,6130	1 00:00:10	5,6130
3	1.0060	0.9890	1.0230	5,5800	5,6130	1 00:00:20	5,6130
4	0.9990	0.9990	0.9990	5,6190	5,6130	1 00:00:30	5,6130
5	0.9900	0.9900	0.9900	5,6700	5,6130	1 00:00:40	5,6130
7	1.0060	0.9890	1.0240	5,5800	5,6130	1 00:00:50	5,6130
8	0.9590	0.9590	0.9590	5,8530	5,6130	1 00:01:00	5,6130
9	1.0190	1.0190	1.0190	5,5080	5,6130	1 00:01:10	5,6130
11	0.9890	0.9890	0.9890	5,6750	5,6130	1 00:01:20	5,6130
12	1.0100	1.0100	1.0100	5,5570	5,6130	1 00:01:30	5,6130
13	0.9890	0.9890	0.9890	5,6750	5,6130	1 00:01:40	5,6130
15	0.9890	0.9680	1.0100	5,6750	5,6130	1 00:01:50	5,6130
16	0.9880	0.9880	0.9880	5,6810	5,6130	1 00:02:00	5,6130
17	0.9890	0.9890	0.9890	5,6750	5,6130	2 00:02:10	5,6130
21	0.9940	0.9890	1.0000	5,6470	5,6130	2 00:02:20	5,6130
23	1.0250	0.9890	1.0620	5,4760	5,6130	2 00:02:30	5,6130
25	1.0340	1.0040	1.0650	5,4280	5,6130	2 00:02:40	5,6130
29	0.9940	0.9890	1.0010	5,6470	5,6130	2 00:02:50	5,6130
31	1.0300	0.9980	1.0620	5,4500	5,6130	2 00:03:00	5,6130
33	1.0050	1.0030	1.0070	5,5850	5,6130	2 00:03:10	5,6130
36	0.9930	0.9890	1.0010	5,6530	5,6130	2 00:03:20	5,6130
39	1.0060	0.9890	1.0290	5,5800	5,6130	2 00:03:30	5,6130
41	0.9960	0.9890	1.0030	5,6360	5,6130	2 00:03:40	5,6130
44	0.9930	0.9890	1.0000	5,6530	5,6130	2 00:03:50	5,6130
47	1.0100	0.9590	1.0590	5,5570	5,6130	2 00:04:00	5,6130
49	1.0050	1.0040	1.0060	5,5850	5,6130	2 00:04:10	5,6130
53	1.0120	0.9700	1.0730	5,5460	5,6130	2 00:04:20	5,6130
57	1.0040	0.9700	1.0480	5,5910	5,6130	2 00:04:30	5,6130
59	1.0200	0.9990	1.0420	5,5030	5,6130	2 00:04:40	5,6130
63	1.0450	1.0090	1.1220	5,3710	5,6130	2 00:04:50	5,6130
65	1.0600	0.9990	1.1220	5,2950	5,6130	0 00:05:00	5,6130
Mittelwerte:	1.0046	0.9898	1.0223	5,5897	5,6130		

Auslastung:

	Minimum	Durchschnitt	Maximum
CPU	1.00%	8.01%	37.37%
Disk	0.00%	0.11%	0.85%

Alternative 2: single-threaded Server

Dynamische Abfrage:

http://localhost:8000/suche.htm?title=&author=mann&select=nur+Autor&Submit=suchen

	Average Duration (seconds)	Minimum Duration (seconds)	Maximum Duration (seconds)	Average Speed (kbytes/sec)	Users	Time (HH:MM:SS)	Average Size (kbytes)
Total Repeats (seconds)	3	1.5870	1.4850	1.7740	3,5720	1 00:00:10	5,6690
6	1.4950	1.4950	1.4950	3,7920	5,6690	1 00:00:20	5,6690
11	1.4960	1.4940	1.5050	3,7890	5,6690	1 00:00:30	5,6690
15	1.4970	1.4940	1.5050	3,7870	5,6690	1 00:00:40	5,6690
18	1.4990	1.4950	1.5100	3,7820	5,6690	1 00:00:50	5,6690
23	1.4920	1.4840	1.4950	3,8000	5,6690	1 00:01:00	5,6690
27	1.4940	1.4850	1.5050	3,7950	5,6690	1 00:01:10	5,6690
30	1.4940	1.4950	1.5050	3,7950	5,6690	1 00:01:20	5,6690
35	1.4960	1.4940	1.5040	3,7890	5,6690	1 00:01:30	5,6690
39	1.4940	1.4940	1.4950	3,7950	5,6690	1 00:01:40	5,6690
42	1.5140	1.4950	1.5450	3,7440	5,6690	1 00:01:50	5,6690
47	1.4960	1.4950	1.5050	3,7890	5,6690	1 00:02:00	5,6690
51	1.5020	1.4950	1.5150	3,7740	5,6690	1 00:02:10	5,6690
54	1.5070	1.4950	1.5150	3,7620	5,6690	2 00:02:20	5,6690
63	1.6210	1.5680	1.9490	3,4970	5,6690	2 00:02:30	5,6690
72	1.5830	1.4880	1.6910	3,5810	5,6690	2 00:02:40	5,6690
76	1.5940	1.5410	1.6070	3,5560	5,6690	2 00:02:50	5,6690
86	1.5710	1.5000	1.6070	3,6090	5,6690	2 00:03:00	5,6690
95	1.5410	1.4640	1.7520	3,6790	5,6690	2 00:03:10	5,6690
102	1.5560	1.5140	1.5790	3,6430	5,6690	2 00:03:20	5,6690
108	1.7020	1.5530	2.0210	3,3310	5,6690	2 00:03:30	5,6690
116	1.6120	1.4840	2.0190	3,5170	5,6690	2 00:03:40	5,6690
123	1.6240	1.5110	2.0180	3,4910	5,6690	2 00:03:50	5,6690
130	1.6320	1.5150	2.0590	3,4740	5,6690	2 00:04:00	5,6690
138	1.6440	1.5130	2.1150	3,4480	5,6690	2 00:04:10	5,6690
145	1.6270	1.4950	2.0280	3,4840	5,6690	2 00:04:20	5,6690
152	1.6060	1.4940	2.0110	3,5300	5,6690	2 00:04:30	5,6690
160	1.6490	1.4910	2.0390	3,4380	5,6690	2 00:04:40	5,6690
167	1.6370	1.5050	2.0980	3,4630	5,6690	2 00:04:50	5,6690
173	1.5350	1.4540	1.6150	3,6930	5,6690	2 00:05:00	5,6690
Mittelwerte:	1.5599	1.4995	1.7190	3.6400			5,6690

Performance-Messung am Webserver

Alternative 3: Komprimierung

Dynamische Abfrage:
http://localhost:8000/suche.htm?title=&author=mann&select=nur+Autor&Submit=suchen

	Average Duration (seconds)	Minimum Duration (seconds)	Maximum Duration (seconds)	Average Speed (kBytes/sec)	Users	Time (HH:MM:SS)	Average Size (kBytes)
Total Repeats (seconds)							
4	1.0210	0.7890	1.6790	1.5900	1	00:00:10	1.6230
11	0.8110	0.7880	0.8790	2.0010	1	00:00:20	1.6230
18	0.7900	0.7890	0.7990	2.0540	1	00:00:30	1.6230
24	0.8280	0.7900	0.9370	1.9600	1	00:00:40	1.6230
30	0.8470	0.7890	0.9780	1.9160	1	00:00:50	1.6230
37	0.7960	0.7890	0.8010	2.0390	1	00:01:00	1.6230
44	0.8730	0.7390	1.4220	1.8590	1	00:01:10	1.6230
51	0.7960	0.7880	0.8090	2.0390	1	00:01:20	1.6230
57	0.7920	0.7880	0.8000	2.0490	1	00:01:30	1.6230
64	0.7970	0.7880	0.8180	2.0360	1	00:01:40	1.6230
72	0.8020	0.7890	0.8390	2.0240	1	00:01:50	1.6230
78	0.7900	0.7880	0.8000	2.0540	1	00:02:00	1.6230
84	0.7870	0.7590	0.8080	2.0620	1	00:02:10	1.6230
91	0.7970	0.7880	0.8190	2.0360	1	00:02:20	1.6230
99	0.7920	0.7890	0.8100	2.0490	1	00:02:30	1.6230
105	0.7980	0.7880	0.8200	2.0340	1	00:02:40	1.6230
111	0.8000	0.7880	0.8400	2.0290	1	00:02:50	1.6230
119	0.7910	0.7870	0.8000	2.0520	1	00:03:00	1.6230
126	0.8270	0.7890	1.0190	1.9630	1	00:03:10	1.6230
132	0.7970	0.7890	0.8190	2.0360	1	00:03:20	1.6230
138	0.7990	0.7880	0.8200	2.0470	1	00:03:30	1.6230
146	0.7920	0.7870	0.8080	2.0490	1	00:03:40	1.6230
153	0.7930	0.7890	0.7990	2.0470	1	00:03:50	1.6230
159	0.8030	0.7900	0.8200	2.0210	1	00:04:00	1.6230
165	0.8040	0.7700	0.8480	2.0190	1	00:04:10	1.6230
172	0.7990	0.7880	0.8100	2.0470	1	00:04:20	1.6230
180	0.7900	0.7890	0.8000	2.0540	1	00:04:30	1.6230
186	0.7950	0.7890	0.8090	2.0420	1	00:04:40	1.6230
192	0.7970	0.7880	0.8290	2.0360	1	00:04:50	1.6230
200	0.8010	0.7880	0.8360	2.0260	1	00:05:00	1.6230
Mittelwerte:	0.8098	0.7852	0.8825	2.0090			1.6230

Statische Abfrage:
http://localhost:8000/1.html

	Average Duration (seconds)	Minimum Duration (seconds)	Maximum Duration (seconds)	Average Speed (kBytes/sec)	Users	Time (HH:MM:SS)	Average Size (kBytes)
Total Repeats (seconds)							
2	0.3210	0.0790	0.5640	1.9000	1	00:00:10	0.6100
4	0.0900	0.0900	0.0900	6.7780	1	00:00:20	0.6100
6	0.0880	0.0860	0.0900	6.9320	1	00:00:30	0.6100
9	0.0900	0.0900	0.0910	6.7780	1	00:00:40	0.6100
11	0.0910	0.0900	0.0930	6.7030	1	00:00:50	0.6100
13	0.0900	0.0900	0.0900	6.7780	1	00:01:00	0.6100
15	0.1100	0.0900	0.1300	5.5450	1	00:01:10	0.6100
17	0.0850	0.0800	0.0900	7.1760	1	00:01:20	0.6100
20	0.0860	0.0800	0.0900	7.0930	1	00:01:30	0.6100
22	0.0900	0.0900	0.0900	6.7780	1	00:01:40	0.6100
24	0.0950	0.0900	0.1000	6.4210	1	00:01:50	0.6100
27	0.0930	0.0900	0.1000	6.5590	1	00:02:00	0.6100
29	0.0900	0.0900	0.0900	6.7780	1	00:02:10	0.6100
31	0.0900	0.0900	0.0900	6.7780	1	00:02:20	0.6100
33	0.0850	0.0800	0.0910	7.1760	1	00:02:30	0.6100
36	0.0890	0.0870	0.0900	6.8540	1	00:02:40	0.6100
38	0.0900	0.0900	0.0910	6.7780	1	00:02:50	0.6100
40	0.0850	0.0800	0.0900	7.1760	1	00:03:00	0.6100
42	0.1010	0.0800	0.1230	6.0400	1	00:03:10	0.6100
45	0.0870	0.0810	0.0910	7.0110	1	00:03:20	0.6100
47	0.0850	0.0810	0.0900	7.1760	1	00:03:30	0.6100
49	0.0950	0.0800	0.1100	6.4210	1	00:03:40	0.6100
51	0.0900	0.0900	0.0900	6.7780	1	00:03:50	0.6100
54	0.1030	0.0800	0.1600	5.9220	1	00:04:00	0.6100
56	0.1200	0.0900	0.1500	5.0630	1	00:04:10	0.6100
58	0.0880	0.0870	0.0900	6.9320	1	00:04:20	0.6100
60	0.0830	0.0800	0.0870	7.3490	1	00:04:30	0.6100
63	0.0870	0.0810	0.0910	7.0110	1	00:04:40	0.6100
65	0.0850	0.0800	0.0900	7.1760	1	00:04:50	0.6100
67	0.0900	0.0900	0.0900	6.7780	1	00:05:00	0.6100
Mittelwerte:	0.0991	0.0847	0.1141	6.5553			0.6100

Statische Abfrage:
http://localhost:8000/tux.jpg

	Average Duration (seconds)	Minimum Duration (seconds)	Maximum Duration (seconds)	Average Speed (kBytes/sec)	Users	Time (HH:MM:SS)	Average Size (kBytes)
Total Repeats (seconds)							
2	0.9300	0.9320	0.9320	5.6280	1	00:00:10	5.2340
4	0.9330	0.9330	0.9330	5.6100	1	00:00:20	5.2340
6	0.9320	0.9320	0.9320	5.6160	1	00:00:30	5.2340
8	0.9570	0.9320	0.9830	5.4690	1	00:00:40	5.2340
11	0.9580	0.9460	0.9720	5.4630	1	00:00:50	5.2340
13	0.9320	0.9320	0.9320	5.6160	1	00:01:00	5.2340
15	0.9320	0.9320	0.9320	5.6160	1	00:01:10	5.2340
17	0.9320	0.9320	0.9330	5.6160	1	00:01:20	5.2340
20	0.9330	0.9330	0.9330	5.6100	1	00:01:30	5.2340
22	0.9570	0.9330	0.9820	5.4690	1	00:01:40	5.2340
24	0.9320	0.9320	0.9330	5.6160	1	00:01:50	5.2340
26	0.9330	0.9330	0.9330	5.6100	1	00:02:00	5.2340
29	0.9390	0.9320	0.9430	5.5740	1	00:02:10	5.2340
31	0.9320	0.9320	0.9320	5.6160	1	00:02:20	5.2340
33	0.9340	0.9320	0.9360	5.6040	1	00:02:30	5.2340
35	0.9330	0.9330	0.9330	5.6100	1	00:02:40	5.2340
38	0.9320	0.9320	0.9330	5.6160	1	00:02:50	5.2340
40	0.9330	0.9320	0.9340	5.6100	1	00:03:00	5.2340
42	0.9400	0.9320	0.9490	5.5680	1	00:03:10	5.2340
44	0.9480	0.9320	0.9640	5.5210	1	00:03:20	5.2340
47	0.9320	0.9320	0.9330	5.6160	1	00:03:30	5.2340
49	0.9480	0.9330	0.9640	5.5210	1	00:03:40	5.2340
51	0.9320	0.9320	0.9320	5.6160	1	00:03:50	5.2340
53	0.9170	0.9020	0.9320	5.7080	1	00:04:00	5.2340
56	0.9560	0.9320	0.9800	5.4750	1	00:04:10	5.2340
58	0.9320	0.9320	0.9320	5.6160	1	00:04:20	5.2340
60	0.9320	0.9320	0.9320	5.6160	1	00:04:30	5.2340
62	0.9470	0.9320	0.9620	5.5270	1	00:04:40	5.2340
65	0.9330	0.9330	0.9330	5.6100	1	00:04:50	5.2340
67	0.9320	0.9320	0.9320	5.6160	1	00:05:00	5.2340
Mittelwerte:	0.9371	0.9316	0.9429	5.5860			5.2340

Auslastung:	Minimum	Durchschnitt	Maximum
CPU	1.00%	8.20%	34.00%
Disk	0.00%	0.15%	2.13%

Alternative 4: Clustering

Dynamische Abfrage:
http://localhost:8000/suche.htm?title=&author=mann&select=nur+Autor&Submit=suchen

	Average Duration (seconds)	Minimum Duration (seconds)	Maximum Duration (seconds)	Average Speed (kBytes/sec)	Users	Time (HH:MM:SS)	Average Size (kBytes)
Total Repeats (seconds)							
3	1.5580	1.5250	1.6250	3.6390	1	00:00:10	5.6690
6	1.5910	1.5240	1.6260	3.5630	1	00:00:20	5.6690
9	1.5250	1.5250	1.5250	3.7170	1	00:00:30	5.6690
14	1.6220	1.5240	1.7050	3.4950	1	00:00:40	5.6690
19	1.6300	1.6240	1.6380	3.4780	1	00:00:50	5.6690
21	1.6320	1.6250	1.6380	3.4740	1	00:01:00	5.6690
24	1.6490	1.6350	1.6570	3.4380	1	00:01:10	5.6690
28	1.6820	1.6250	1.7460	3.3700	1	00:01:20	5.6690
32	1.6840	1.6250	1.7750	3.3660	1	00:01:30	5.6690
36	1.6570	1.6370	1.6850	3.4210	1	00:01:40	5.6690
39	1.6410	1.6250	1.6650	3.4550	1	00:01:50	5.6690
42	1.6280	1.6240	1.6350	3.4820	1	00:02:00	5.6690
48	1.6660	1.5260	1.9150	3.4030	2	00:02:10	5.6690
56	1.6320	1.6290	1.6410	3.4740	2	00:02:20	5.6690
63	1.6330	1.6280	1.6400	3.4720	2	00:02:30	5.6690
69	1.6290	1.6280	1.6330	3.4890	2	00:02:40	5.6690
76	1.6450	1.6330	1.6750	3.4460	2	00:02:50	5.6690
84	1.6820	1.6330	1.7600	3.3700	2	00:03:00	5.6690
92	1.8160	1.6670	2.1030	3.1220	2	00:03:10	5.6690
99	1.6860	1.6290	1.7420	3.3620	2	00:03:20	5.6690
105	1.7000	1.6020	1.7740	3.3350	2	00:03:30	5.6690
111	1.9450	1.6310	3.4610	2.9150	2	00:03:40	5.6690
118	1.6450	1.6310	1.6580	3.4460	2	00:03:50	5.6690
126	1.6790	1.6330	1.7520	3.3760	2	00:04:00	5.6690
134	1.6490	1.5380	1.7380	3.4380	2	00:04:10	5.6690
139	1.5940	1.5360	1.6560	3.5560	2	00:04:20	5.6690
148	1.6170	1.5430	1.6800	3.5060	2	00:04:30	5.6690
156	1.4770	0.2640	1.7440	3.4130	2	00:04:40	5.0410
164	1.6380	1.6330	1.6530	3.4610	2	00:04:50	5.6690
171	1.7100	1.6450	1.9290	3.3150	0	00:05:00	5.6690
Mittelwerte:	1.6514	1.5582	1.7685	3.4263			5.6481

Statische Abfrage:
http://localhost:8000/1.html

	Average Duration	Minimum Duration (seconds)	Maximum Duration (seconds)	Average Speed (kbytes/sec)	Users	Time (HH:MM:SS)	Average Size (kBytes)
Total Repeats	1	1.2300	1.2300	1.2300	4.5760	1 00:00:10	5.6280
3	1.0700	1.0250	1.1150	5.2600	1	00:00:20	5.6280
4	1.0250	1.0250	1.0250	5.4910	1	00:00:30	5.6280
5	1.1580	1.1580	1.1580	4.8600	1	00:00:40	5.6280
6	1.1120	1.1120	1.1120	5.0610	1	00:00:50	5.6280
7	1.1150	1.1150	1.1150	5.0480	1	00:01:00	5.6280
9	1.1300	1.1150	1.1450	4.9810	1	00:01:10	5.6280
10	1.0260	1.0260	1.0260	5.4850	1	00:01:20	5.6280
11	1.0260	1.0260	1.0260	5.4850	1	00:01:30	5.6280
12	1.0370	1.0370	1.0370	5.4270	1	00:01:40	5.6280
13	1.1160	1.1160	1.1160	5.0430	1	00:01:50	5.6280
15	1.1150	1.1150	1.1160	5.0480	1	00:02:00	5.6280
17	1.1400	1.1320	1.1480	4.9370	2	00:02:10	5.6280
19	1.1250	1.1180	1.1330	5.0030	2	00:02:20	5.6280
22	1.0950	1.0330	1.1330	5.1400	2	00:02:30	5.6280
25	1.0720	1.0340	1.1400	5.2500	2	00:02:40	5.6280
27	1.1370	1.1340	1.1410	4.9500	2	00:02:50	5.6280
29	1.0650	1.0340	1.0970	5.2850	2	00:03:00	5.6280
31	1.0820	1.0310	1.1330	5.2010	2	00:03:10	5.6280
34	1.1440	1.1260	1.1710	4.9200	2	00:03:20	5.6280
37	1.1380	1.0680	1.2000	4.9460	2	00:03:30	5.6280
39	1.0830	1.0340	1.1330	5.1970	2	00:03:40	5.6280
41	1.0860	1.0400	1.1330	5.1820	2	00:03:50	5.6280
43	1.1330	1.1330	1.1340	4.9670	2	00:04:00	5.6280
45	1.1390	1.0340	1.2450	4.9410	2	00:04:10	5.6280
48	1.1270	1.0470	1.1310	4.9940	2	00:04:20	5.6280
51	1.1020	1.0400	1.1330	5.1070	2	00:04:30	5.6280
53	1.1110	1.0910	1.1320	5.0660	2	00:04:40	5.6280
55	1.0650	1.0400	1.0900	5.2850	2	00:04:50	5.6280
58	1.1030	1.0370	1.1400	5.1020	00 05:00	5.6280	
Mittelwerte:	1.1036	1.0769	1.1283	5.1079			5.6280

Danksagung

Nach nunmehr 5 Jahren Studium und 6 Monaten Diplomarbeit ist es an der Zeit verschiedenen Menschen zu danken.

Als erster Stelle sind meine Eltern zu nennen, die mir nicht nur das Studium ermöglicht haben, sondern mich auch während dieser Zeit in jeder erdenklichen Weise unterstützt haben. Ihnen widme ich diese Arbeit.

Herrn Jun.-Prof. Dr. Ralf Reussner gilt mein Dank für die hervorragende Betreuung während der Diplomarbeit und die viele Zeit, die er für mich aufwendete. Steffen Becker half mir bei der Durchführung des Experiments und Viktoria Firus las meine Ausarbeitung zur Korrektur und gab mir viele wichtige Hinweise. Die freundliche Arbeitsatmosphäre in der Palladio-Gruppe und der Abteilung Software-Engineering motivierte mich immer wieder während der letzten Monate.

Yvette Teiken danke ich für die Implementierung des Webservers, den ich in meiner Fallstudie als Untersuchungsobjekt nutzen konnte. Den 24 Teilnehmern der Übungen und des Experiments danke ich für ihren Einsatz, insbesondere auch für die nachträgliche Korrektur der Ergebnisse, durch die ich meine Auswertung maßgeblich verbessern konnte. Ebenso danke den sieben freiwilligen Teilnehmern der Kontrollgruppe, die die Zeit aufwendeten und intuitiv die Performance des Webservers analysierten.

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich diese Diplomarbeit selbstständig angefertigt habe und alle Teile, die wörtlich oder inhaltlich anderen Quellen entstammen, als solche kenntlich gemacht und in das Literaturverzeichnis aufgenommen habe. Diese Arbeit wurde weder in dieser noch einer ähnlichen Form einer anderen Prüfungsbehörde vorgelegt.

Oldenburg, den 28. Oktober 2004

(Heiko Koziolk)