Untersuchung des Lastverhaltens des Monday Webforms Backends

Entwicklung einer Anwendung zur Lasterzeugung

Bachelor-Thesis

zur Erlangung des akademischen Grades B.Sc.

Christian Lehr

2063585

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Design, Medien und Information
Department Medientechnik

Erstprüfer: Prof. Edmund Weitz

Zweitprüfer: Nils Meins

Hamburg, 23. 12. 2015

Inhaltsverzeichnis

Abstract

This thesis defines the term performance in the field of computer science and how it can be measured. The importance of performance in different areas will also be explained. The main subject of this work is the setup of a testing environment which will be used to test the performance of the software „Monday Webforms Backend“ by the company Monday Consulting.

Zusammenfassung

Im Rahmen dieser Bachelorarbeit wird erklärt, was Performance in der Informatik ist, wie man sie messen kann und welchen Stellenwert sie in verschiedenen Bereichen hat. Der Kern der Arbeit befasst sich mit dem Aufbau einer Testumgebung und der anschließenden Durchführung und Analyse von Last-/Performancetests an der Software „Monday Webforms Backend“ der Firma Monday Consulting.

Sperrvermerk

Die vorgelegte Bachelorarbeit mit dem Titel „Untersuchung des Lastverhaltens des Monday Webforms Backends “ beinhaltet vertrauliche Informationen und Daten des Unternehmens Monday Consulting GmbH.

Diese Bachelorarbeit darf nur vom Erst- und Zweitgutachter sowie berechtigten Mitgliedern des Prüfungsausschusses eingesehen werden. Eine Vervielfältigung und Veröffentlichung der Bachelorarbeit ist auch auszugsweise nicht erlaubt.

Dritten darf diese Arbeit nur mit der ausdrücklichen Genehmigung des Verfassers und Unternehmens zugänglich gemacht werden.

1 Einleitung

Last- und Performancetests sind ein wichtiges Instrument, um ein System im Hinblick auf das zu erwartende Benutzerverhalten zu prüfen. Ziel ist es, die Skalierbarkeit verschiedener Funktionen einer Applikation zu ermitteln [Sch12a].
Eine entscheidende Rolle spielen Tests dieser Art zum Beispiel vor Großveranstaltungen von internationalem Format wie den Olympischen Spielen. Sie sind fester Bestandteil der Projektplanung. Die Hard- und Softwareinfrastruktur für die Sommerspiele 2012 in London sollte bereits vor Beginn der Spiele rund 200.000 Stunden lang getestet werden. Bereits 2 Jahre zuvor liefen die Vorbereitungen auf Hochtouren, damit die mehr als 9.500 PCs und Laptops, etwa 900 Server und rund 1.000 Netzwerk- und Sicherheitskomponenten im Ernstfall reibungslos zusammenarbeiten [Sch12b].
Oft werden die Begriffe Last- und Performancetest synonym verwendet, nicht jeder Performancetest muss jedoch ein Lasttest sein. Performancetest ist lediglich der Oberbegriff für Tests, die ein korrektes Systemverhalten bei bestimmten Speicher- und CPU-Anforderungen sicherstellen sollen. Sind die Speicher und CPU-Anforderungen hoch, so wird in der Regel von Lasttests gesprochen. Wenn Beispielsweise die „Single Performance“ eines Systems ermittelt werden soll, ist es nicht nötig, das System unter Last zu setzen. Hier würde man nicht von einem Lasttest sprechen, sondern von einem Performancetest. Dies macht zum Beispiel dann Sinn, wenn man ermitteln möchte, ob die End-to-End Antwortzeit einer Applikation die größte Wartezeit auf Clientseite oder Serverseite hat. Hier kann es sein, dass nicht die Last für die schlechte Performance verantwortlich ist, sondern schon die „Single Performance“ nicht den Erwartungen entspricht. Ein Lasttest wäre in diesem Fall redundant, da die Performance unter Last nicht besser sein wird [Chea].
Da in dieser Bachelorarbeit das System unter verschiedenen Lastbedingungen getestet wird, wird im Folgenden der Oberbegriff Performancetest verwendet.

1.1 Motivation

Warum sollte man Performancetests durchführen? Um diese Frage zu beantworten soll zunächst ein Beispiel genannt werden, das zeigt, was passieren kann, wenn eine Applikation nicht ausreichend auf ihre Performance hin untersucht wird.
In den Medien wurde Anfang bis Mitte 2014 über die Show „Quizduell“ berichtet. Sie wurde über mehrere Folgen hinweg von Lastproblemen der gleichnamigen App gestört, die im Mittelpunkt der Sendung stand. Die App bot den Zuschauern die Mög-

lichkeit, am TV-Geschehen teilzunehmen. Erst im achten Anlauf lief eine Sendung fast störungsfrei [qui14].
Das Beispiel zeigt, dass gute Performance nicht optional, sondern geschäftskritisch ist. Regelmäßige Last- und Performancetests im Software-Entwicklungsprozess sind hierfür eine wichtige Voraussetzung.
Die Testergebnisse der Last- und Performancetests weisen frühzeitig auf Risiken hin und geben Aufschluss über vorhandene Kapazitäten sowie systemkritische Situationen. Weiterhin helfen sie, unnötige Investitionen in die Infrastruktur und Hardware zu vermeiden und schützen weitestgehend vor „Totalausfällen“ des Systems [Cheb].
Werden Verzögerungen oder Ausfälle, die durch entsprechende Tests vermeidbar gewesen wären, hingenommen, schadet eine Firma ihrer Bilanz sowie ihrem eigenen Ansehen. Letztlich kann sogar der Verlust von Kunden drohen [Sch12a].
Um Probleme dieser Art für die Software Monday Webforms Backend der Firma Monday Consulting zu vermeiden, wird in dieser Arbeit eine Testumgebung aufgebaut, mit deren Hilfe das Lastverhalten der Software anhand verschiedener Szenarien untersucht wird.

1.2 Ziel

Das Ziel dieser Bachelorarbeit ist der Aufbau einer funktionsfähigen Testumgebung, deren Grundlage das Monday Webforms Backend darstellt. Anschließend werden basierend auf dieser Testumgebung Performancetests durchgeführt. Die Last für die Tests wird durch eine in Java entwickelte Lastanwendung erzeugt. Über eine Benutzeroberfläche können verschiedene Parameter eingestellt werden, welche die Testkonfiguration beeinflussen.
Nach der Definition einiger Testfälle für die Performancetests, werden diese in der Testumgebung durchgeführt und mit Hilfe des Lastwerkzeugs JMeter (siehe 3.6) überwacht, um Informationen über das Systemverhalten unter verschiedenen Testbedingungen zu sammeln. Abschließend findet eine statistische Analyse der Testdaten und eine graphische Aufbereitung der gemessenen Daten statt, sowie eine Beschreibung des Lastverhaltens der Testumgebung in Bezug auf die verschiedenen Testfälle.

1.3 Gliederung

Im Folgenden wird die Struktur dieser Arbeit vorgestellt.

In **Kapitel 2** wird eine Einführung in den Begriff Performance gegeben. Weiterhin werden verschiedene Arten von Performancetests vorgestellt und deren Bedeutung in der heutigen Zeit dargelegt. Außerdem werden wichtige Bereiche in Webapplikationen vorgestellt, die als Ansatzpunkt für eine Performanceoptimierung essentiell sind.

In **Kapitel 3** wird die Software Monday Webforms Analytics und dessen Hard- und Softwareinfrastruktur vorgestellt. Diese beinhaltet das zu untersuchende Monday Webforms Backend. Weiterhin werden die Gründe für die Entwicklung der Software genannt. Anschließend werden die selbst entwickelte Lastanwendung samt Benutzeroberfläche sowie wichtige Klassen und Methoden vorgestellt, die für das weitere Verständnis von Bedeutung sind. Abschließend wird auf die für die Performancetests entworfenen Testszenarien sowie auf deren Durchführung eingegangen.

In **Kapitel 4** werden die Testergebnisse statistisch ausgewertet und graphisch dargestellt. Das Lastverhalten wird anhand der Graphen beschrieben und es werden Hypothesen zur Erklärung des Verhaltens aufgestellt.

Kapitel 5 fasst die Arbeit und ihre Resultate zusammen. Weiterhin findet eine Aussicht statt, welche weiteren Szenarien getestet werden könnten, um noch mehr Informationen über die Performance des Systems zu gewinnen.

2 Performance - Grundlagen

Im folgenden Kapitel wird zunächst erklärt, was Performance ist und wie sie sich messen lässt. Weiterhin wird gezeigt, was für einen Stellenwert die Messung von Performance in der heutigen Zeit hat. Es werden verschiedene Testarten vorgestellt, mit denen die Performance eines Systems untersucht werden können. Abschließend wird erklärt wie sich die Performance einer Softwareanwendung optimieren lässt.

2.1 Definition Performance

Der Begriff Performance beschreibt in der Informatik im Allgemeinen einen Ausdruck für die Leistung eines Systems, welches Hardware, Software oder auch nur einen einzelnen Algorithmus umfasst [Sto08]. Es gibt dabei verschiedene Modelle zur Beschreibung von Performance. Zwei werden im Folgenden vorgestellt.

Nach dem ISO/IEC 14756 Standard zur Messung der Performance bei Computersystemen lässt sich Performance in 3 Klassen unterteilen:

Die **erste Klasse** beschreibt die Fähigkeiten eines Computers. Dabei sind die Menge der Aktivitäten und die Korrektheit sowohl der durchgeführten Operationen als auch deren Ergebnisse enthalten. Die Benutzerfreundlichkeit zählt ebenfalls in diese Klasse.

Zur **zweiten Klassen** gehören die Stabilität und Konsistenz eines Computersystems und seiner Operationen. Es geht also weitestgehend um die Zuverlässigkeit des Systems.

Bei der **dritten Klasse** geht es darum, wie schnell das System die Aufgaben ausführt. Dabei geht es einerseits um die Zeit, die für ein Ergebnis benötigt wird und andererseits um die Anzahl von Aufgaben und Operationen, die in einer bestimmten Zeit ausgeführt werden [Dir06].

Ein besonderer Fokus sollte in der Performanceanalyse auf den Faktoren **Verlässlichkeit/Ausfallsicherheit**, **Geschwindigkeit** und **Wirtschaftlichkeit** gelegt werden [HG97]. Weiterhin sollte bedacht werden, dass Performance im Kontext mit einer Zielgruppe gesehen werden sollte. Ein Benutzer einer Webanwendung interessiert sich beispielsweise vor allem für schnelle Antwortzeiten und die Funktionsfähigkeit eines

Systems. Ein Webmaster hingegen legt mehr Wert auf einen hohen Datendurchsatz und eine hohe Verfügbarkeit [Chm09, S.6].
Ein zweiter Ansatz bei der Definition von Performance wird von Whitworth et al. [WFM06] vorgestellt. Sie beschreiben ein Netz der Systemperformance (Web of System Performance), das aus 8 miteinander gekoppelten Zielsetzungen besteht. Jeder Bereich kann dabei unterschiedlich gewichtet sein, je nach Umgebung und Anforderung. Die 8 Zielsetzungen werden wie folgt definiert.

Erweiterbarkeit Erlaubt das Hinzufügen von Erweiterungen durch Dritte.

Sicherheit Beschreibt den Umstand, wie gut sich ein System gegen äußere, unbefugte Zugriffe schützen kann.

Flexibilität Fähigkeit, in anderen Umgebungen betrieben zu werden.

Zuverlässigkeit Robustheit gegenüber starker Belastung und sichere Abschaltung bei einem Fehlerfall.

Funktionalität Begriff für den Umfang der Fähigkeiten eines Systems.

Nutzbarkeit Beschreibt die Effizienz bzw. Nutzerfreundlichkeit eines Systems.

Konnektivität Fähigkeit des Systems, mit weiteren Systemen zu kommunizieren, äußere Dienste in Anspruch zu nehmen oder selber Dienste anzubieten.

Datenschutz Fähigkeit des Systems, den Umfang der Informationen über sich selbst zu kontrollieren, die nach außen hin zugänglich gemacht wurden.

Die netzartige Struktur der Performance bei diesem Ansatz wird anhand Grafik 2.1 veranschaulicht:

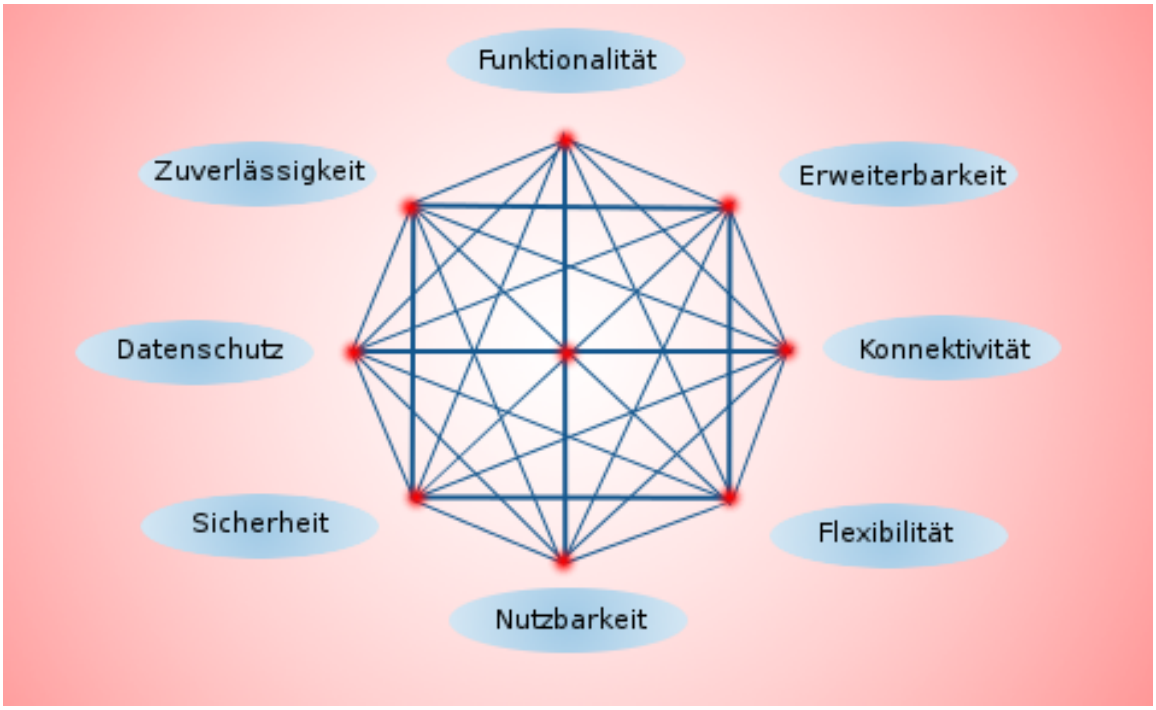

Abbildung 2.1: Web of Performance [WFM06]

Bei übermäßiger Gewichtung eines Teilaspekts und Vernachlässigung anderer Aspekte, kann das Ergebnis ein insgesamt nicht mehr performantes System sein. Weiterhin bedingen sich die einzelnen Teilaspekte. So kann es sein, dass die Verbesserung eines Teilziels die Verschlechterung eines anderen zur Folge hat [WFM06].

2.2 Das Ziel von Performancetests

Das übergeordnete Ziel von Performancetests ist, Informationen über ein System zu sammeln, um am Ende Aussagen über dessen Leistung treffen zu können. Es gibt unterschiedliche Parameter für die Bestimmung von Performance. Ein Beispiel hierfür ist der Datendurchsatz, welcher die Menge der Daten beschreibt, die in einer bestimmten Zeit verarbeitet werden. Je nach Anwendungstyp können unterschiedliche Parameter wichtig werden. Betrachtet man beispielsweise Webanwendungen, so spielt die Antwortzeit eine wesentliche Rolle. Die Antwortzeit ergibt sich dabei aus drei einzelnen Komponenten. Nach [Sto08] gilt folgende Gleichung:

Antwortzeit = Browserantwortzeit + Netzwerkantwortzeit + Serverantwortzeit

Die einzelnen Parameter dürfen dabei nicht isoliert betrachtet werden, sondern müssen im Kontext, beispielsweise zusammen mit der Nutzerzahl gesehen werden. Es gibt also kein einheitliches Maß für Performance. Im Endeffekt müssen für jedes System individuell passende Parameter und Vorgehensweisen gefunden werden. So hat die Komplexität einer Architektur natürlich auch einen Einfluss auf die Vorgehensweise. Bei komplexeren Systemlandschaften reicht die Betrachtung der Performance der einzelnen Komponenten nicht aus, um Rückschlüsse über die Gesamtperformance des Systems ziehen zu können. Das liegt daran, dass auch das Zusammenspiel der Komponenten einen Einfluss auf die Performance hat. Also muss auch bzw. vor allem das gesamte System getestet werden [Sto08].

2.2.1 Allgemeine Vorgehensweise zur Bestimmung von Performance

In den letzten Jahren wurden viele Methoden zur Performancemessung entwickelt, wobei jeder Test oft eigene Anforderungen hat. Auch wenn die Bestimmung von Performance ein individueller und manchmal komplexer Prozess ist, so gibt es dennoch einige grundlegende Schritte, die den Ablauf eines Tests beschreiben.
Diese sehen laut [JHK05] wie folgt aus:

1. **Analyse des Produktionssystems**
2. **Festlegung des Lastprofils**
3. **Implementierung des Testsystems**
4. **Durchführung der Testreihen und Sammlung von Informationen**
5. **Interpretation der Ergebnisse**
6. **Zusammenfassung**

In [Dir06] werden drei Arten der Performancemessung bei Computersystemen angegeben: Messung, Simulation und Modellierung. Bei der Messung findet ein reales Experiment auf einem echten System in Echtzeit statt. Es werden Eingaben von Benutzern getätigt, die anfallenden Daten werden gesammelt und gespeichert. Anhand dieser Daten können Leistungsparameter berechnet werden.
Bei der Simulation wird das System und dessen Benutzer vereinfacht nachgestellt.
Bei der Modellierung wird aus einem vereinfachten Modell des Systems einschließlich der Benutzer ein mathematisches Modell entwickelt. Die Berechnung der Leistungsparameter findet durch aufgestellte mathematische Formeln statt. Die Ergebnisse bei Messung, Simulation und Modellierung weisen Unterschiede auf. Dies liegt daran, dass nur bei der Messung ein echtes System getestet wird. Bei der Simulation und Modellierung wird lediglich ein vereinfachtes Modell geprüft. Auch wenn Simulation und Modellierung nicht die Präzision einer echten Messung aufweisen, liegt der Vorteil eines analytischen Modells darin, dass es bereits sehr früh in der Entwurfsphase eines Projekts angewandt werden kann und geringere Kosten verursacht [Ris05].

2.2.2 Metriken von Perfomance- und Lasttests

Um Aussagen über das Lastverhalten eines Systems machen zu können, werden üblicherweise folgende Metriken mit Hilfe von Lasttestwerkzeugen gemessen und ausgewertet:

Requests per Second (RPS) Hierbei handelt es sich um den Umfang der Interaktionen zwischen Browser und Webserver. Bleibt diese Zahl mit steigender Last konstant, bedeutet dies im Regelfall, dass der Webserver voll ausgelastet ist.

CPU-Auslastung Wenn sie dauerhaft deutlich über 50% liegt, ist die Leistung des Prozessors zu gering ausgelegt.

Queued Requests (QR) Dies ist die Anzahl der noch ausstehenden Abfragen in der Warteschlange des Servers: Wenn sie über eine längere Zeit größer Null ist, steht der Server unter Stress.

Umlaufzeit Die Umlaufzeit ist die Dauer zwischen Beginn und Abschluss einer Transaktion [1] und sollte unter Stress konstant bleiben. Wenn die Transaktionen langsamer werden oder misslingen, bedeutet dies, dass der Server keine weiteren Anfragen mehr bearbeiten kann.

Transaktionsdauer Verschiedene Testszenarien lösen auch unterschiedliche Transaktionen aus. Wenn bei mehreren Transaktionen zeitliche Ausreißer vorhanden sind, kann dies ein Hinweis auf einen Fehler in der Programmierung sein.

Verbindungen Wenn bei gleich bleibender Anzahl von Anfragen die Anzahl der gleichzeitigen Verbindungen zum Webserver steigt, bedeutet das, dass die Verbindungen länger als notwendig offen bleiben.

Speicherplatz Mit jeder neuen Verbindung sinkt der verfügbare Speicherplatz eines Webservers. Dieser wird nach Beendigung der Verbindung wieder freigegeben. Ein Speicherleck kann man daran erkennen, dass bei gleichbleibender Anzahl von wechselnden Verbindungen die Größe des verfügbaren Speichers abnimmt.

Datendurchsatz Misst die Menge an übertragenen Daten vom und zum Webserver in Kilobyte pro Sekunde. Das System hat seine Grenzen erreicht, wenn die Anzahl der Kilobyte nicht mehr proportional zur Anzahl der Nutzer ist [Fra07].

[1] Eine Transaktion ist in der Informatik eine Folge von Programmschritten, die als logische Einheit betrachtet werden, weil der Datenbestand nach fehlerfreier und vollständiger Ausführung in einem konsistenten Zustand hinterlassen wird [Tra15].

2.3 Stellenwert von Performancetests

Es gibt viele Bereiche, die von Performancetests profitieren können. In der Softwareentwicklung beispielsweise können Performancetests bei der Auswahl der Systemarchitekturen behilflich sein. Weiterhin können Implementierungen direkt getestet werden, wodurch früh Leistungsengpässe gefunden und eine frühe Optimierung der Software vorgenommen werden kann. Dadurch werden hohe Kosten für die spätere doch teils aufwändige Fehlerbehebung gespart [Sto08].
Besonders im World Wide Web ist eine gute Performance wichtig. Hier können bereits Millisekunden entscheidend sein. Nutzer warten im Schnitt nur 3 Sekunden beim Laden einer Webseite [YF10]. Wird diese Zeit nicht eingehalten, so wechseln viele User die Seite und 80% der Nutzer kommen nicht mehr zurück. Bereits eine Sekunde Verzögerung beim Laden einer Webseite hat 7% weniger Einnahmen, 11% weniger Seitenaufrufe und 16% weniger Kundenzufriedenheit zur Folge [rad]. Ebenfalls auf die Nutzung von Google hat die Geschwindigkeit Auswirkungen. Durch Tests wurde herausgefunden, dass bereits eine minimale Zeitverzögerung von wenigen 100 Millisekunden bei der Darstellung der Suchergebnisse einen negativen Einfluss auf die Anzahl der Suchanfragen hat. Eine halbe Sekunde Verzögerung verringert die Nutzung von Google bereits um 20 Prozent [May08].

In Unternehmen haben Performancetests ebenfalls einen hohen Stellenwert, da Systemverhalten und Zeit eine wichtige Rolle spielen und so auch Einfluss auf den Erfolg eines Unternehmens haben. Man möchte lange Wartezeiten von internen Applikationen vermeiden, da diese mit erheblichen Kosten für das Unternehmen verbunden sind. Als Beispiel sei hier der Online-Händler Amazon genannt, der das Verhältnis zwischen Ladezeit, Kundenzufriedenheit und Verkäufen näher untersucht hat. Dazu wurde die Darstellung der Amazon-Webseite künstlich um 100 ms verzögert. Das Ergebnis war, dass der Umsatz sich um 1% verringerte. Bei einem Jahresumsatz von 17,43 Milliarden US-Dollar sind dies 174 Millionen US-Dollar [Lin06].
Auch privat ist eine hohe Performance wünschenswert. Negativbeispiele sind hier, wenn beispielsweise die Navi-App nur mit Verzögerungen funktioniert oder länger zum Starten braucht, als die Fahrerin oder der Fahrer selbst zum Losfahren. Dieser möchte möglichst schnell in die richtige Richtung losfahren, ohne dabei erst zu warten, bis die App vollständig geladen ist [Tes15].
Die Wichtigkeit von Performance lässt sich psychologisch begründen. Der Wissenschaftler R. Miller hat sich vor 40 Jahren mit der Frage nach zumutbaren Antwortzeiten während einer Interaktion zwischen Mensch und Computer beschäftigt. Er hat diese Interaktion mit der Kommunikation zwischen 2 Menschen verglichen und herausgefunden, dass ein Mensch im Gespräch innerhalb von 2 Sekunden eine Antwort

erwartet. Verstreichen 4 Sekunden oder mehr ohne Antwort, so bricht die Kommunikation abrupt ab, vergleichbar mit einer plötzlich gekappten Telefonleitung. Diese Erwartungshaltung ist laut R. Miller ein psychologisches Bedürfnis [Chm09, S.16ff]. Grafik 2.2 veranschaulicht diesen Umstand am Beispiel der Ladezeit einer Webseite. Eine gewisse Wartezeit wird vom Nutzer toleriert und als Verzögerung oder Ladezeit wahrgenommen. Übersteigt die Wartezeit jedoch eine kritische Grenze so wird sie als Unterbrechung wahrgenommen. Ist die wahrgenommene Lade- und Darstellungszeit höher, als die Erwartungshaltung, sinkt die Toleranz beim Besucher und er beginnt abzuschweifen. Am Beispiel eines Online-Shops wäre der schlimmste Fall, dass der Besucher eine Kaufhandlung abbricht und die Anwendung verlässt.

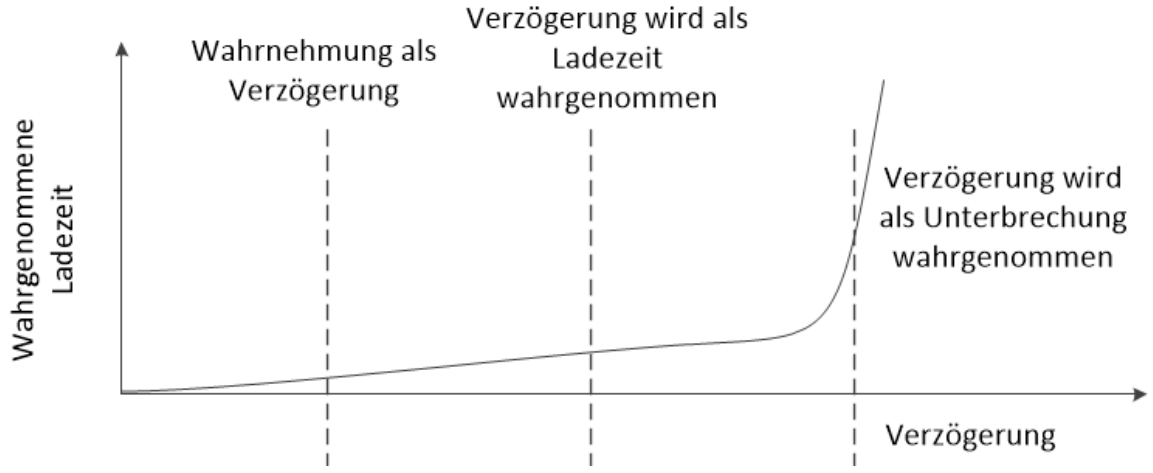

Abbildung 2.2: Wahrgenommene Ladezeit [KR13]

2.4 Testarten

Im Folgenden werden verschiedene Arten von Performancetests vorgestellt, die zur Analyse eines Systems eingesetzt werden können.

2.4.1 Performancetest

Performancetest ist der Oberbegriff für Tests, die ein korrektes Systemverhalten bei bestimmten Speicher- und CPU-Anforderungen sicherstellen sollen. Oft wird von einem Performancetest gesprochen, wenn das System unter „normalen “ Bedingungen getestet wird, es wird also nicht überlastet. Dabei wird das Zeitverhalten, die Mengenverarbeitung und der Ressourcenverbrauch überprüft [Fra07].

Lasttest

Bei einem Lasttest wird ein System unter Last gesetzt. Die Last wird dabei schrittweise erhöht. Lasttests dienen dazu, funktionale Fehler von Systemen/Software auf-

zudecken bei gleichzeitiger Betrachtung von Zeit- und Verbraucherverhalten.

Dauerlasttest

Von einem Dauerlasttest spricht man, wenn ein System über einen längeren Zeitraum (48-72 Stunden) unter Last gesetzt wird [Vog09].

Stresstest

Ein Stresstest testet das System unter „abnormen“ Bedingungen. Das heißt es wird übertriebene Last erzeugt, um herauszufinden, unter welchen Lastbedingungen funktionale Fehler auftreten bzw. die Antwortzeiten nicht mehr tragbar sind und das System überlastet ist. Dabei wird die Zahl der User kontinuierlich gesteigert, bis der kritische Punkt erreicht wird. Stresstests werden im Regelfall dazu verwendet, um Engpässe im System ausfindig zu machen [Vog09].

Zerfallskurve

Laut [KR13] haben Lasttests zum Ziel, eine Anwendung an ihre Grenzen zu bringen und durch kontinuierliches Messen eine Zerfallskurve zu erzeugen. Eine solche Kurve wird in Abbildung 2.3 gezeigt.

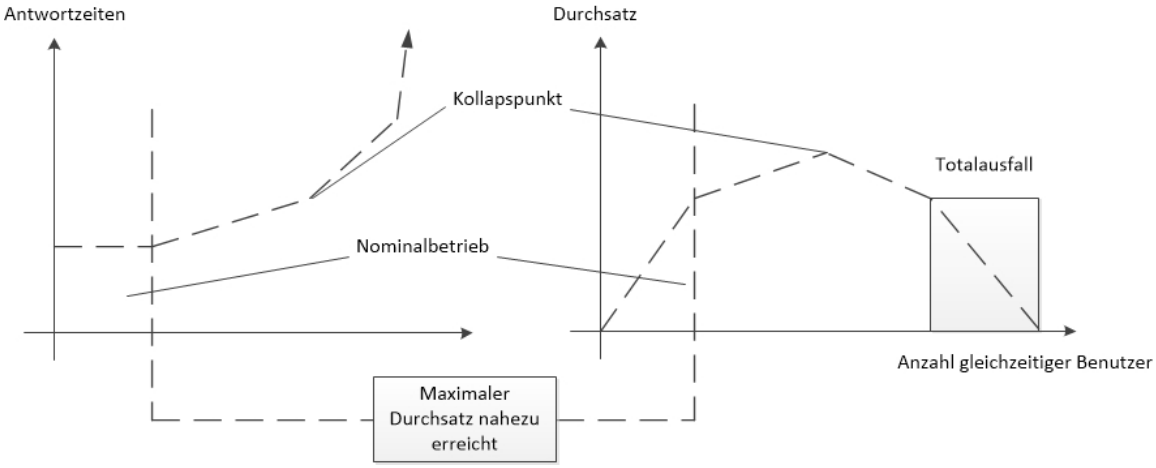

Abbildung 2.3: Zerfallskurve eines Lasttests [KR13]

Diese Kurve kann in 4 Bereiche aufgeteilt werden:

1. **Nominalbetrieb:** Solange die Last ein gewisses Maß nicht übersteigt, wird die Antwortzeit stabil bleiben und lediglich geringe Schwankungen aufweisen.

2. **Maximaler Durchsatzpunkt:** Ab einer bestimmten Last wird die Antwortzeit stetig ansteigen und die Schwankungen werden zunehmen. Dieser Punkt

wird als „maximaler Durchsatzpunkt “ bezeichnet und stellt die Grenze des Systems dar.

3. **Kollapspunkt:** Nach Erreichen des maximalen Durchsatzes wird die Performance der Anwendung durch eine weitere Erhöhung der Last zunehmend schlechter. Der Durchsatz sinkt und die Antwortzeiten beginnen stark zu schwanken. Dieser Punkt wird als Kollapspunkt bezeichnet.

4. **Totalausfall:** Wird die Last nun zunehmend erhöht, kann es zu einem Totalausfall der Anwendung kommen. Dies muss keinen Ausfall der Hardware bedeuten, sondern kann sich auch nur in extrem hohen Antwortzeiten äußern, die kein Benutzer tolerieren würde.

2.4.2 Weitere Testarten

Skalierbarkeitstest

Skalierbarkeit beschreibt die Fähigkeit eines Software-Produkts, technisch aufgerüstet werden zu können, um eine erhöhte Last zu verkraften. Bei einem Skalierbarkeitstest soll diese ermittelt werden. Ein System skaliert gut, wenn es beispielsweise bei doppelter Anzahl von Prozessoren die Hälfte der Rechenzeit benötigt oder bei Verdopplung der Belastung mit den doppelten Hardware-Ressourcen auskommt, ohne dass Eingriffe in die Software notwendig sind. Beim Skalierbarkeitstest wird wie beim Lasttest die Belastung des Systems schrittweise gesteigert [Fra07].

Speicherlecktest

Ein Speicherleck entsteht, indem zuvor benutzter Arbeitsspeicher von einer Software nicht wieder freigegeben wird. Das kann dazu führen, dass beispielsweise ein Webserver über die Zeit verfügbaren Speicher verliert, bis er arbeitsunfähig ist oder sogar abstürzt. Speicherlecks werden nach der Durchführung der Performancetests gesucht und zwar indem ein Test über vier bis fünf Tage durchgeführt wird. Speicherlecks werden daran erkannt, dass Antwortzeiten oder Übertragungsraten stetig nachlassen. Bereits kleine Speicherlecks können die Leistung eines Systems erheblich beeinflussen und sind somit ein wichtiger Ansatzpunkt für die Zuverlässigkeit eines Systems [Fra07].

Benchmark-Test

Bei einem Benchmark-Test findet die Messung der Performance anhand von wohldefinierten Workload-Modellen statt. Hier gibt es eine große Anzahl von Modellen. Vereinfacht gesagt, wird durch einen Benchmark-Test der Vergleich der Performance verschiedener Systeme möglich. Benchmark-Tests werden hauptsächlich für die Messung von Performance bei Hardware verwendet [Sto08].

Regressionstest

Werden Fehler einer Software korrigiert oder Wartungsarbeiten durchgeführt, so kommt es zwangsläufig dazu, dass Teile der Software verändert werden oder neue Softwarebausteine hinzukommen. Mit Hilfe von Fehlernachtests wird nachgewiesen, dass Fehler tatsächlich korrigiert wurden. Der Regressionstest testet ein bereits getestetes Programm, das verändert wurde. Das Ziel ist, nachzuweisen, dass durch die vorgenommenen Änderungen keine neuen Defekte eingebaut wurden.
Ein Regressionstest kann sehr aufwendig sein. Folgende Möglichkeiten kommen dabei in Betracht.

1. **Wiederholung aller Tests, die Fehler sichtbar gemacht haben, die nun korrigiert wurden (Fehlernachtest)**
2. **Test aller Programmstellen, die korrigiert oder verändert wurden. (Test geänderter Funktionalität)**
3. **Test aller Programmteile oder Bausteine, die neu eingefügt wurden (Test neuer Funktionalität)**
4. **Das komplette System (vollständiger Regressionstest)**

Im Prinzip sind die ersten 3 Tests zu wenig und im Idealfall müsste ein vollständiger Regressionstest durchgeführt werden. In der Praxis ist dieser aber fast immer zu zeit- und kostenintensiv. Deshalb wird oft überlegt, welche Testfälle ohne zu großen Informationsverlust weggelassen werden können [SL10].

2.5 Optimierung von Performance

Optimierung bedeutet im Allgemeinen die Verbesserung eines Merkmals, eines Materials oder eines Produkts. Optimierung im Kontext einer Webanwendung hat zum Ziel, seine Leistungsfähigkeit im Hinblick auf die Nutzung der Systemressourcen, der Geschwindigkeit sowie der „User Experience“ zu verbessern. Eine Optimierung kann allerdings erst dann stattfinden, wenn messbare Kriterien für die Software existieren. Es muss definiert werden, auf welchen Teil der Anwendung sich die Optimierung konzentriert und wie die Performance im Kontext der Anwendung definiert ist. Es muss definiert werden was als schnell bzw. langsam angesehen wird. Das Bestimmen, Messen und Ausführen der Kriterien erfolgt dabei über iterative Performancetests. Das bedeutet, dass mehrere Tests durchgeführt werden, wobei ein Test immer jeweils auf den Daten des vorherigen basiert. Der Workflow von Performanceoptimierung wird in Grafik 2.4 dargestellt [KR13].

Um Performance optimieren zu können, ist es wichtig, die Bereiche einer Anwendung zu kennen, die Einfluss auf die Performance haben. Daher werden in diesem Kapitel

die wichtigsten Bereiche und Begrifflichkeiten genannt und erläutert, die für eine Performanceoptimierung innerhalb einer Applikation von besonderem Interesse sind.

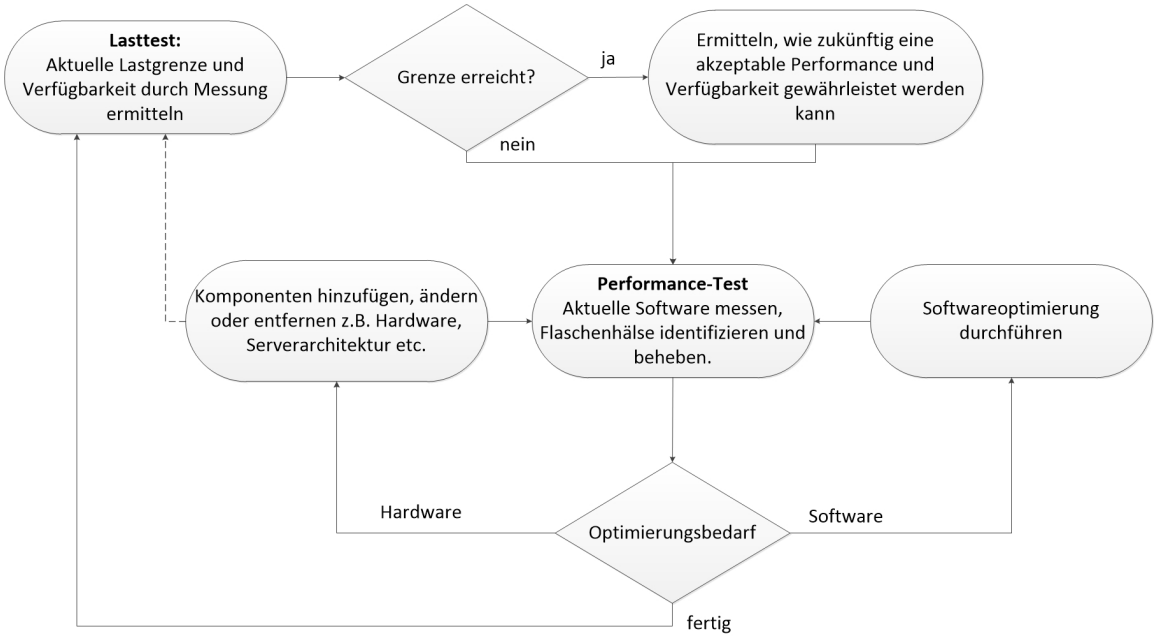

Abbildung 2.4: Allgemeine Vorgehensweise bei Optimierungen [KR13]

Flaschenhals

Ein Flaschenhals ist ein Engpass beim Transport von Daten. Dieser hat einen maßgeblichen Einfluss auf die Abarbeitungsgeschwindigkeit. Wenn das Gesamtsystem also optimiert werden soll, muss beim Flaschenhals angesetzt werden, da Optimierungen an anderer Stelle nur zu geringfügigen oder gar keinen messbaren Veränderungen führen würden [lex09].
Wird ein Flaschenhals beseitigt, so gibt es einen neuen Flaschenhals, der die Gesamtperformance einschränkt, natürlich nicht so stark wie der vorherige Flaschenhals. Dies wird anhand von Abbildung 2.5 veranschaulicht.

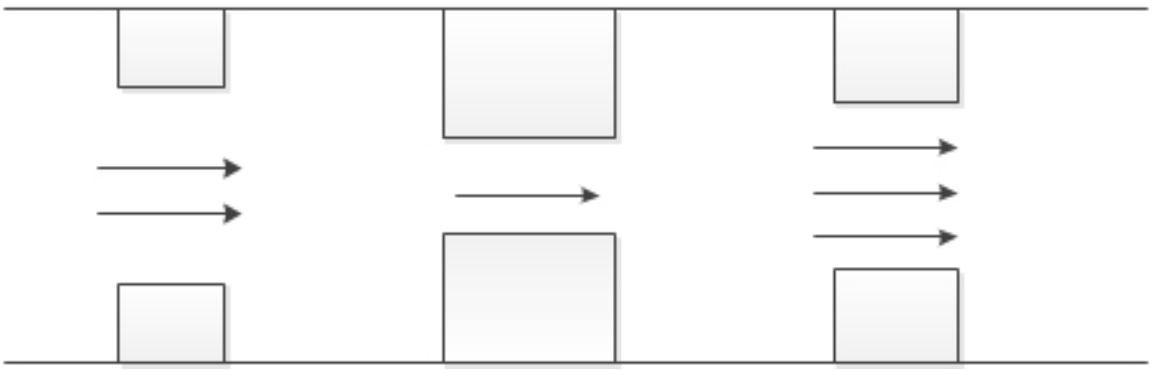

Abbildung 2.5: Wird ein Flaschenhals beseitigt, gibt es einen neuen weiteren [Bar07].

Betriebssystem

Ein besonders wichtiger Faktor ist das Betriebssystem, das Zugriffe auf die Systemressourcen verwaltet. Zwischen einzelnen Versionen von Betriebssystemen können große Performanceunterschiede bestehen. Beispielsweise konnte ein großes Unternehmen aus der Online Werbebranche 10 von 25 veralteten SuSe 11 Server durch Wechsel auf neuere Linux-Distribution abschalten. Die verbleibenden 15 Server waren leistungsfähiger als die vorigen 25 Server und das bei gleichbleibender Hardware.
Wichtiger Faktor sind neben der Wahl des Betriebssystems auch dessen Einstellungen. Über das Betriebssystem können die Hardwarekomponenten justiert werden. So können Änderungen an den Festplatten-Caches, Sektorgrößen, Swap-Größe, Scheduler und die Wahl des Dateisystems essentielle Wirkung haben [KR13].

Programmiersprache, Frameworks & Bibliotheken

Die Aktualität der Version der eingesetzten Programmiersprachen ist wichtig, da diese Bugfixes und neue Features enthalten, die die Performance positiv beeinflussen können. Bei der jungen Programmiersprache Ruby werden beispielsweise von Release zu Release viele Performance-Verbesserungen veröffentlicht.
In der Softwareentwicklung werden häufig Frameworks eingesetzt, die das Ziel haben, immer wieder auftretende Tätigkeiten zu abstrahieren und dem Entwickler eine einfache Möglichkeit geben, diese zu verwenden. Diese Abstraktion bedeutet oftmals jedoch, dass Einbußen der Performance zugunsten der Strukturierung und der einfachen Verwendung hingenommen werden müssen. Bei einer umfassenden Performance-Optimierung sollten Framework-Funktionen hinsichtlich ihrer Ausführungsgeschwindigkeit evaluiert werden. Weiterhin ist ebenfalls auf die Aktualität der Versionen von Frameworks, Bibliotheken und Treibern zu achten [KR13].
Die Wichtigkeit der Aktualität wird am folgenden Beispiel deutlich: Seit Firefox 3.1 wurde die native jQuery Methode document.querySelectorAll eingeführt. Durch diese können JavaScript-Bibliotheken DOM-Elemente per CSS-Selektor 2- bis 6- mal schneller auffinden als die bisherigen eigenen Implementierungen der Bibliotheken.

Um diese nutzen zu können, braucht man jedoch eine aktuelle jQuery-Version [Res08].

Hardware

Auch die Hardware hat großen Einfluss auf die Performance. Dabei sind vor allem wichtig: Prozessor, Arbeitsspeicher, Festplatte und das Netzwerkgerät. Auch wenn diese Teile heutzutage vergleichsweise schnell und preiswert sind, können sie durch eine zu hohe Anfragelast überlastet werden [Sch06].
T. Schlossnagle macht in seinem Werk deutlich, wie es dazu kommen kann: Ein Prozessorkern arbeitet eine Aufgabe nach der anderen ab. Die Zeitablaufsteuerung des Betriebssystems regelt dabei die Dauer einer Arbeitseinheit und die Umschaltung auf eine neue Aufgabe (sog. Context Switch). Der Context Switch findet selbst dann statt, wenn die gegenwärtige Aufgabe noch nicht vollständig ausgeführt wurde. Die Umschaltung kostet ihrerseits Zeit (einige Mikrosekunden). Da der Prozessorkern nur eine bestimmte Anzahl an Aufgaben pro Zeiteinheit abarbeiten kann, steigt mit der Last auch die Antwortzeit.
Prozesse benötigen neben der CPU-Zeit auch Speicher. Der Arbeitsspeicher wirkt sich daher auf die Antwortzeit und den Durchsatz aus: Mit steigender Zahl an Anfragen werden mehr Prozesse gestartet und damit immer mehr Speicher konsumiert. Ab einer gewissen Anzahl an Prozessen ist nicht mehr genügend freier Speicher vorhanden (Ressource Utilization) und das System beginnt mit dem Auslagern nicht aktiver Prozesse und deren im Speicher gehaltenen Zwischenberechnungen auf den viel langsameren Festplattenspeicher (sog. Swapping). Damit kommt die Verarbeitung fast zum Erliegen. Die Festplatte wirkt sich nach [Sch06], zusätzlich negativ aus, wenn viele datenintensive Operationen durchgeführt werden, wie es beispielsweise bei vielen Datenbankzugriffen der Fall ist. Dies hat zur Folge, dass sie die Lese- und Schreiboperationen nicht schnell genug abarbeiten kann und sich die Operationswarteschlange der Festplatte nach und nach füllt. Prozesse müssen nun länger bei Eingabe- und Ausgabeoperationen warten und die Performance sinkt.
I/O-Operationen werden auch beim Zugriff auf die Netzwerkhardware durchgeführt. Müssen zu viele Daten übertragen werden und wächst gleichzeitig die Dichte der Anfragen durch zu viele Clients, so kann die maximale Bandbreite erschöpft werden und damit nicht nur die Prozesse ausbremsen, sondern auch die Antwortzeit erhöhen.

Skalierung

Skalierung heißt, ein System mit Ressourcen zu erweitern, sodass mehr Last verarbeitet werden kann. Eine Software oder ein System ist dann skalierbar, wenn dieses durch Hinzufügen von mehr Hardware mehr Benutzer verarbeiten kann und die maximale Anzahl der Benutzer nur durch die aktuell eingesetzte Hardware begrenzt wird. Im Idealfall wird Software so entworfen, dass bei einer steigenden Anzahl an Benutzern ein linearer Anstieg des Ressourcenverbrauchs einhergeht. Dies ist in der Realität jedoch kaum zu erreichen, da ein großer Teil der eingesetzten Software und

Systeme mit einer steigenden Anzahl an Benutzern einen im Vergleich dazu größeren Ressourcenverbrauch verursachen. Oft ist die genutzte Software nur schwer skalierbar wie es beispielsweise bei Datenbanken der Fall ist. Dies bedeutet, dass ab einem gewissen Zeitpunkt durch den Einsatz von mehr Hardware keine weitere Skalierbarkeit und dadurch auch kein Performance-Gewinn zu erreichen ist. Im Rahmen dieser Bachelorarbeit findet beispielsweise eine Skalierung der CPU-Anzahl innerhalb verschiedener Testfälle statt (siehe Kapitel 3.7).

Bei der Skalierung wird zwischen der horizontalen und der vertikalen Skalierung unterschieden.

Vertikale Skalierung Bei der vertikalen Skalierung werden die eingesetzten Server, die noch nicht die maximale Hardware-Ausstattungsstufe im Sinne von freier Arbeitsspeicherslots, freier Festplattenplätze und Anzahl an CPUs erreicht haben, weiter ausgebaut. Schwache Server, die nicht mehr ausgebaut werden können, werden dabei durch stärkere Server ersetzt. Bei der vertikalen Skalierung wird also bereits bestehende Hardware optimiert.

Horizontale Skalierung Horizontale Skalierung bedeutet die Erweiterung der bestehenden Hardware um weitere Hardware. Die Last wird dabei auf die verfügbare Hardware verteilt, um einen parallelen Betrieb möglich zu machen. Um dies zu erreichen, müssen die Webanwendung und die beteiligten Softwaresysteme wie beispielsweise Datenbanken, auf mehrere Server verteilt werden. Dies ist jedoch nur mit Software möglich, die für ein solches Vorgehen entwickelt und optimiert wurde [KR13].

Architektur & Implementierung

Ob eine Webanwendung ausfallsicher ist und skalierbar ist, um auch beim Versagen einzelner Komponenten oder einem plötzlichen Anstieg der Arbeitsbelastung performant zu arbeiten, entscheidet die Architektur der Software. Eine Webanwendung wird in der Regel auf mehreren Servern ausgeführt. Dies dient zum einen der Arbeitsteilung (Programmcode verteilt ausführen). Zum anderen werden Dienste, wie (verteilte) Datenbank- und Proxyserver oder Lastverteiler auf dedizierten Hosts installiert [Sch06]. Mangelnde Redundanz gefährdet die Performance des Gesamtsystems, da der Ausfall nur einer einzigen Komponente zum Totalausfall führen kann. Das System kann von Clients nicht mehr erreicht werden, da ein zentraler Knotenpunkt ausgefallen ist (Router, Lastverteiler) oder nicht mehr korrekt seine Arbeit verrichtet, weil auf integrale Bestandteile nicht zugegriffen werden kann (Datenbankserver, Fileserver).

Die oben erwähnten software- und hardwareseitigen Ursachen für mangelnde Performance bei steigender Anfragedichte können zum Vorschein kommen, wenn bei

der Planung der Infrastruktur keine Serverlastverteilung, bzw. nicht genug parallele Server bereit gestellt wurden [Sch06]. Die Server laufen dann während einer Anfragespitze an ihren Kapazitätsgrenzen.

Zusammenfassend kann festgestellt werden, dass die Dichte der Anfragen einen kritischen Faktor bilden. Erst mit steigender Last durch zu viele Clients, wenn also das System an seine Grenzen gebracht wird, kommt es zu einer Sättigung und einem Abfall der Performance.

Datenhaltung

Die Datenhaltung in Webanwendungen ist vielseitig. Serverseitig werden Datenbanken oder Dateien verwendet, um die Daten der Anwendung zu speichern. Diese Daten werden von der Webanwendung gelesen und aufbereitet an den Client übertragen. Im Frontend-Bereich werden clientseitige Datenbanken oder Dateien verwendet, die sich auf der lokalen Festplatte des Benutzers befinden. Die meisten Webanwendungen benutzen heutzutage relationale Datenbanken. Die bei Webanwendungen am häufigsten eingesetzten relationalen Datenbanken sind MySQL und Postgres. Neben den relationalen Datenbanken gibt es noch die schemalosen Datenbanken. Bekannte Vertreter sind hier MongoDB, CouchBase, Riak und Cassandra. Schemalose Datenbanken haben im Vergleich zu relationalen Datenbanken noch nicht die volle Marktreife erreicht, da sie noch vergleichsweise jung sind und erst in den letzten Jahren an Bekanntheit gewonnen haben. In Sachen Performance und Skalierbarkeit sind schemalose Datenbanken aufgrund von flexiblen, schemalosen Strukturen den relationalen Datenbanken überlegen. Bei großen Anwendungen kann die Datenbank schnell zum Flaschenhals werden. Die Flaschenhälse resultieren häufig aus ineffizienten Datenbankzugriffen oder einer fehlerhaften Strukturierung des Datenbankschemas [KR13].

Netzwerk

Ein wichtiger Umstand beim Einfluss des Netzwerks auf die Performance ist das sogenannte Fat File Paradox [Lei08]. Eine große Datei benötigt aufgrund der Netzwerkprotokolle zur Überquerung einer weiten Strecke sehr viel Zeit, da in einem Netzwerk der Durchsatz eng an die Latenz [2] gekoppelt ist.
Das Transmission Control Protocol/Internet Protocol ist ein im Internet verwendetes Protokoll, das einen zuverlässigen Datentransport erlaubt. Die Funktionsweise sieht dabei wie folgt aus: Die Datenpakete werden solange wiederholt an den Empfänger gesendet, bis dieser den Empfang bestätigt. Die eigentlichen Daten werden segmentiert (höchstens 64 KB) und als IP-Datengramme übertragen [Hol02]. Wegen der obligatorischen Empfangsbestätigung sinkt der Datendurchsatz mit steigender Netzwerklatenz, da keine neuen Pakete übertragen werden, solange der fehlerfreie

[2] Bei der Latenz oder auch Verzögerungszeit handelt es sich um das Zeitintervall vom Ende eines Ereignisses bis zum Beginn der Reaktion auf dieses Ereignis [ITW].

Empfang nicht bestätigt wurde. Weiterhin wird bei Erkennung von Paketverlust die Menge der gesendeten Daten reduziert. Mit zunehmender Streckenlänge häufen sich die Paketverluste, was heißt, dass die Strecke zum Server auch Einfluss auf die Dauer der Übertragung hat [Lei08].

Ermittlung der Performance

Systemoptimierungen sollten erst durchgeführt werden, wenn Schwachstellen und Flaschenhälse einer Anwendung bekannt sind. Ebenfalls muss bedacht werden, dass die Performance der Software nur einen Teil der Gesamtperformance einer Webanwendung ausmacht. Netzwerk-, Server- oder Festplattenauslastung können vor allem bei Shared Web Hosting Angeboten unberechenbar sein und die Performance unterschiedlich stark beeinflussen. Um frühzeitig auf Schwachstellen reagieren und Gegenmaßnahmen einleiten zu können, empfiehlt sich die Liveüberwachung des Systems. Es gibt Überwachungstools die bei bestimmten Ereignissen wie Festplattenausfall, Netzwerküberlastung und hoher CPU-Auslastung automatisiert entsprechende Maßnahmen ausführen können.
Neben der automatisierten Liveüberwachung empfiehlt es sich nicht nur die Gesamtperformance des Systems zu untersuchen, sondern mit Hilfe von Performance- und Lasttests auch einzelne Komponenten der Anwendung auf ihre Leistungsfähigkeit hin zu untersuchen. Diese sollten per Monitoring oder Logging überwacht werden, wodurch man Auskunft über Auslastung, aufgetretene Fehler und mögliche Flaschenhälse erhält [KR13]. In dieser Bachelorarbeit findet ebenfalls ein Monitoring des Systems unter Last statt. Dieses wird mit Hilfe von JMeter bewerkstelligt. Näheres dazu findet sich in Kapitel 3.6.

Ausfallsicherheit

Ein zuverlässiges System ist gegen Ausfälle gesichert. Sollte das System trotz Vorsorgemaßnahmen ausfallen, muss gewährleistet sein, dass keine Daten verloren gehen. Ausfallsicherheit kann dadurch erhöht werden, dass Systemkomponenten mehrfach bereitgestellt werden, damit diese als Backup-Komponenten bei Ausfall automatisch die Funktion der ausgefallenen Komponente übernehmen können. Dies nennt sich Failover. Bei den betroffenen Systemkomponenten eines Failover kann es sich um die Anwendung, einen Teil der Anwendung, die Datenbank, die Hardware, die Firewall, das Netzwerk oder sogar das komplette Rechenzentrum handeln [Fra07]. Im Folgenden werden einige Strategien zur Erhöhung der Ausfallsicherheit erörtert.

Verwendung von redundanten Webservern Ausfallsicherheit kann durch einen redundanten Web-Server erhöht werden. Dabei überwachen sich zwei Server gegenseitig über eine Netzwerkverbindung, indem sie Signale austauschen. Wenn ein Server nicht mehr antwortet, weil er ausgefallen ist, startet der andere Server automatisch die Übernahme der Dienste.

Verwendung eines Load Balancers Eine weitere Möglichkeit zur Optimierung der Ausfallsicherheit ist die Verwendung eines Load Balancers. Dieser verteilt die anfallende Last auf mehrere Webserver, die zu einem virtuellen Server zusammengeschlossen sind. Wenn einer der Server ausfällt, wird die Performance der Anwendung nicht bzw. wenig davon beeinträchtigt. Der Load Balancer selbst wird paarweise betrieben, damit das System auch dann stabil bleibt, wenn einer der Load Balancer ausfällt [Fra07]. Abbildung 2.6 veranschaulicht diesen Umstand.

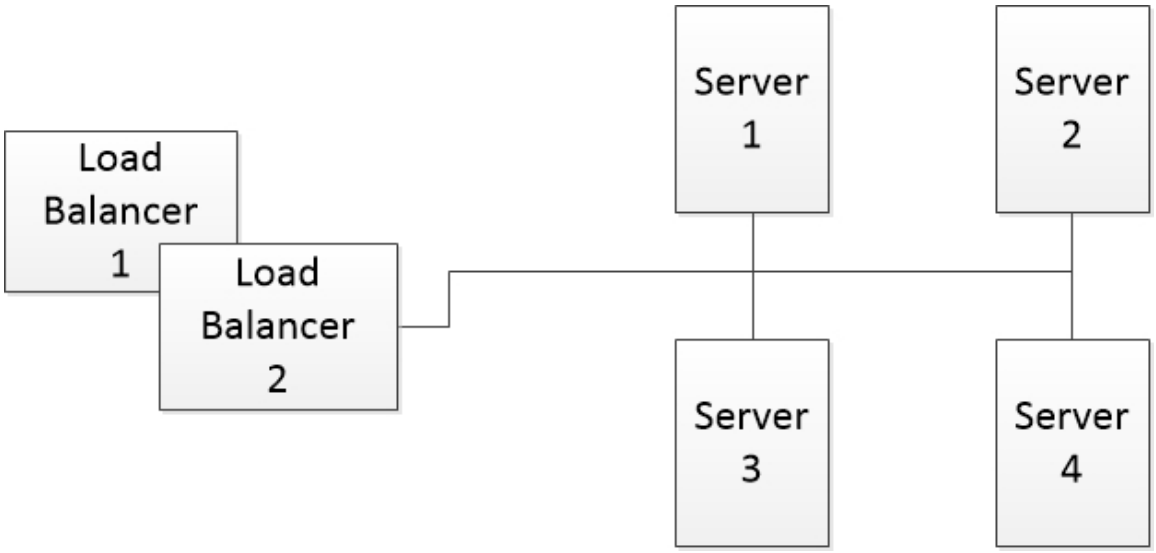

Abbildung 2.6: Die Last wird auf mehrere Server verteilt [Fra07].

Graceful Degradation Ist die gezielte Abschaltung von Diensten und kann zum Einsatz kommen, um auf Lastspitzen einer Anwendung zu reagieren. Es wird unterschieden zwischen der Abschaltung einzelner Features der Anwendung, der Umleitung von Systemressourcen von anderen Anwendungen oder der Limitierung der Zugriffe auf die Anwendung. Bei einer Anwendung, die in mehrere voneinander unabhängige Features aufgeteilt werden kann, könnten bei Auftritt von Lastsptitzen beispielsweise einzelne Features abgeschaltet werden. Die Reihenfolge der einzelnen Features wird im Idealfall durch die Priorität der einzelnen Funktionen bestimmt.
Ein Beispiel für Graceful Degradation ist Facebook, welches eine Monitoring-Anwendung besitzt, Lastspitzen erkennt und als Gegenmaßnahme unkritische Features abschaltet [Haa]. Das Hauptgeschäftsmodell von Facebook ist der Umsatz durch eingeblendete Werbung. Kommt es zu Lastspitzen könnte beispielsweise die niedrigste Priorität auf der Chat-Funktion liegen. Diese könnte als Gegenmaßnahme vorübergehend ausgeschaltet werden, um die Ressourcen für Basisfunktionalitäten wie Statusmeldungen zu verwenden [KR13].

Usability

Die Performance einer Anwendung kann durch die Verbesserung der Nutzerführung optimiert werden. Eine Verbesserung der Nutzerführung verkürzt die Zeit, die ein Besucher einer Webanwendung benötigt, um sein Ziel zu erreichen. Jede Bewegung eines Nutzers innerhalb der Anwendung kann Last erzeugen, dies kann sich nachteilig auf die Performance auswirken, da die Webserver zusätzlich belastet werden. Wird beispielsweise festgestellt, dass Nutzer von der Startseite über Unterseite A nach Unterseite B gelangen und erst dort die gewünschten Inhalte findet, kann es sinnvoll sein, die Unterseite B direkt auf der Startseite zu verlinken. Dadurch würde der Server entlastet und der Datentransfer reduziert. Weiterhin wird dem Nutzer überflüssiges Klicken sowie Wartezeit erspart.
Eine weitere Möglichkeit die Benutzerführung zu optimieren, könnte darin liegen, Arbeitsabläufe versteckt im Hintergrund auszuführen. So wäre es bei einer Upload-Funktion für Bilder denkbar, Dateien, direkt nach dem Auswählen, an den Server hochzuladen. Der Hochladevorgang kann über ein für den Nutzer unsichtbares IFrame oder per JavaScript durchgeführt werden. Dadurch hätte der Nutzer die Möglichkeit, direkt weitere Bilder auszuwählen oder bereits hochgeladene Bilder mit Metainformationen zu versehen, ohne sich selbst um den Hochladenvorgang kümmern zu müssen. Ein weiteres Beispiel dieser Art sind Online-Texteditoren. Diese können automatisch in regelmäßigen Abständen Inhalte speichern, wodurch der Nutzer nach einem unvorhersehbaren Absturz des Browsers mit einem aktuellen Stand des Dokuments weiterarbeiten kann [KR13].

Clientseitige Optimierung

Der User interagiert mit der Webanwendung über eine im Browser dargestellte Webseite. Daher definiert sich die Performance aus Sicht des Benutzers als die Ladezeit dieser Seite. Sie ist diejenige Zeit, die verstreicht, während die vom Benutzer ausgelöste Anfrage durch den Webserver abgearbeitet und das Resultat an den Client übertragen wird. S. Souders hat während seiner Arbeit bei Yahoo! herausgefunden, dass der überwiegende Teil der gesamten Ladezeit einer Webseite im Webbrowser verbraucht wird, während HTML-Anweisungen interpretiert und ausgeführt werden. Für seine Messungen hat er die auf Alexa.com 10 bestbewertetsten Webseiten in den USA auf die Zeit hin untersucht, die serverseitig verbraucht wird. Bei jeder Messung wurde sowohl ein voller Browsercache als auch ein leerer Browsercache betrachtet. Bis auf 2 Ausnahmen lag die Zeit, die vom Backend/Server in Anspruch genommen wurde unter 10%. Das heißt, dass 80-90% der Zeit vom Front-End gebraucht wurde [Sou08].

Soll folglich die Ladezeit einer Seite optimiert werden, so geschieht dies vor allem auf Frontend-Seite. Jedes Element einer Seite wie beispielsweise eine Grafik oder JavaScript-, CSS-Datei muss erst bei einem Webserver angefordert werden, um an-

Die 10 meist besuchten Webseiten in den USA im Jahr 2008		
Webseite	Leerer Cache	voller Cache
www.aol.com	3%	3%
www.ebay.com	5%	19%
www.facebook.com	5%	19%
www.google.com/search?q=flowers	53%	100%
search.live.com/results.aspx?q=flowers	33%	100%
www.msn.com	2%	6%
www.myspace.com	2%	2%
en.wikipedia.org/wiki/Flowers	6%	9%
www.yahoo.com	3%	4%
www.youtube.com	2%	3%

Tabelle 2.1: Anteil der Ladezeit in Prozent, der durch das Backend verursacht wird [Sou08].

schließend an den Webbrowser ausgeliefert werden zu können. Dadurch entsteht Wartezeit, die bei mehr als vier angeforderten Elementen mehr als 50% der Ladezeit ausmachen kann [Kin08]. Der Grund ist, dass der Browser nur zwei gleichzeitige Verbindungen zum selben Webserver haben kann. Werden viele Elemente vom gleichen Webserver geladen, kommt es zu entsprechenden Verzögerungen. Dazu kommt, dass sich der Overhead von TCP/IP[3] negativ auf die Ladeseite auswirkt, da bei vielen Elemente ebenfalls viele Verbindungen zum Server geöffnet werden müssen. Zusätzliche Verzögerungen kommen dadurch zu stande, dass TCP/IP erst mehrere Datenpakete hin und her überträgt, ehe eine Verbindung aufgebaut ist. Die Protokolle führen für den Verbindungsaufbau einen sogenannten Dreiwege-Handshake durch [Chm09].
Weiterhin wurde laut Nahum et al. TCP nicht dafür konzipiert, viele Daten zwischen Client und Server zu übertragen, da dabei mehr Pakete ausgetauscht werden als semantisch nötig [NBK02].
Durch eine Reduzierung der zu übertragenen Daten wird die Übertragungsdauer verringert. Weiterhin kann die Datenkompression des Browsers und des Webservers verwendet werden, um die übertragenen Daten zu minimieren. Dies kann durch eine Einstellung am Webserver erreicht werden und ist ohne Änderung der Anwendung möglich. Die gewonnene Geschwindigkeit geht auf Kosten der CPU, da die Komprimierung der Daten Rechenzeit des Webservers beansprucht. Dies ist jedoch zu vernachlässigen, da CPUs heutzutage Hardwareerweiterungen für die Komprimierung

[3]Protokolle zur Übertragung von Daten im Internet.

von Daten beinhalten. Weitere Zeit kann dadurch eingespart werden, dass ähnliche Inhalte auf dem Übertragungsweg zusammengefasst werden. Darunter fällt beispielsweise das Zusammenfassen von JavaScript-, Stylesheet- oder Grafikdaten, was zur Folge hat, dass weniger Verbindungen zum Server aufgebaut werden müssen. Solche Funktionalitäten werden zum Teil schon als Erweiterungen für Webserver wie beispielsweise Apache angeboten [KR13].

2.6 Fazit

In diesem Kapitel wurde der Begriff Performance anhand von 2 Modellen vorgestellt. Es wurde gezeigt, dass Performance ein weitreichender Begriff ist, der mehr umfasst als schnelle Antwortzeiten. Darunter fallen beispielsweise Datenschutz, Ausfallsicherheit und die Nutzerfreundlichkeit. Weiterhin wurde gezeigt, was das Ziel von Performancetests ist. Dieses besteht in der Sammlung von Informationen eines Systems, um anschließend Aussagen über dessen Leistung machen zu können. Dazu wurden einige Metriken vorgestellt, die zum Messen von Performance genutzt werden können. Darunter fallen beispielsweise Antwortzeit, Datendurchsatz und CPU-Auslastung. Es wurde ebenfalls beschrieben, dass Performance immer im Kontext gesehen werden muss und es kein einheitliches Maß für Performancemessung gibt.
Anschließend wurde Performance unter ökonomischen und psychologischen Aspekten betrachtet, wo gezeigt wurde, dass bereits kleine Performanceeinbußen finanzielle Schäden zur Folge haben können.
Neben den Testarten Performance- und Lasttest wurden weitere wie beispielsweise der Skalierbarkeitstest und Speicherlecktest vorgestellt.
Abschließend wurde auf den Bereich Performanceoptimierung eingegangen. Hier wurde zunächst erklärt, dass ein Flaschenhals ein Engpass beim Transport von Daten ist. Weiterhin wurde beschrieben, wie sich Netzwerk und Hardware auf Performance auswirken können. Anschließend wurden einige Möglichkeiten genannt, die die Ausfallsicherheit eines Systems gewährleisten. Darunter fallen die Verwendung von redundanten Webservern und Load Balancern. Zudem wurde auf das Thema Usability eingegangen. Es wurde gezeigt, dass gute Usability sich positiv auf die Performance einer Anwendung auswirken kann. Abschließend wurde gezeigt, dass Performanceoptimierung vor allem auch auf Frontendseite geschehen muss, da der Großteil der Ladezeit einer Anwendung durch die Interpretation von HTML-Anweisungen bestimmt wird.

3 Aufbau und Einrichtung der Testumgebung

In diesem Kapitel wird zunächst die Software- und Hardwarearchitektur des „Monday Webforms Analytics“ beschrieben, welches das Monday Webforms Backend beinhaltet. Weiterhin werden die Beweggründe für die Entwicklung der Anwendung Monday Webforms Analytics erläutert. Anschließend wird die selbst entwickelte Anwendung vorgestellt, deren Hauptfunktion in der Lasterzeugung für die später durchgeführten Performancetests besteht. Letztlich wird auf die Entwicklung der Testszenarien für die Performancetests eingegangen und worauf bei der Durchführung der Tests geachtet wurde.

3.1 Monday Webforms Analytics

Webforms ist ein Plugin für die beiden Content Management Systeme[1] CoreMedia und FirstSpirit [2] und dient der Erstellung von HTML-Formularen im Web.
Das Monday Webforms Analytics ist eine Analyse-Erweiterung für Webforms. Es basiert auf 2 Webanwendungen, dem Monday Webforms Backend und der Monday Webforms Reporting-WebApp, welche ebenfalls in Grafik 3.1 zu sehen ist. Letztere enthält eine auf AngularJS[3] basierende Webanwendung, die über eine Spring Boot[4] Anwendung Zugriff auf die Daten des Backends hat und diese im Browser darstellt. Im Rahmen dieser Bachelorarbeit wird lediglich das Monday Webforms Backend getestet, somit ist die Reporting-WebApp für diese Arbeit nicht relevant und soll lediglich der Vollständigkeit halber genannt werden.

[1] Redaktionssystem, mit dessen Hilfe der Inhalt z.B. von Websites verwaltet wird. Dabei erfolgt eine Trennung von eigentlichem redaktionellem Inhalt und dem Layout. Folglich können Inhalt als auch Layout getrennt voneinander verändert werden, ohne in den jeweils anderen Bereich eingreifen zu müssen [Wir].

[2] Monday Consulting GmbH ist Implementierungspartner für die Hersteller der beiden genannten Content Management Systeme CoreMedia AG und e-Spirit AG [Cor][Fir].

[3] AngularJS ist ein quelloffenes Javascript-Framework für die Entwicklung clientseitiger Webapplikationen [Bri].

[4] Spring Boot ist ein Projekt auf Basis des Spring-Frameworks und ermöglicht die einfache Entwicklung eigenständiger lauffähiger Spring-Anwendungen [Spr15].

3.1.1 Hard- und Softwarearchitektur

Die Architektur eines Benutzers, der die Software Monday Webforms Analytics erwirbt, wird im Normalfall aus einem Webserver, einem Applikationsserver und einem Datenbankserver bestehen. Der Aufbau wird in Grafik 3.1 gezeigt.

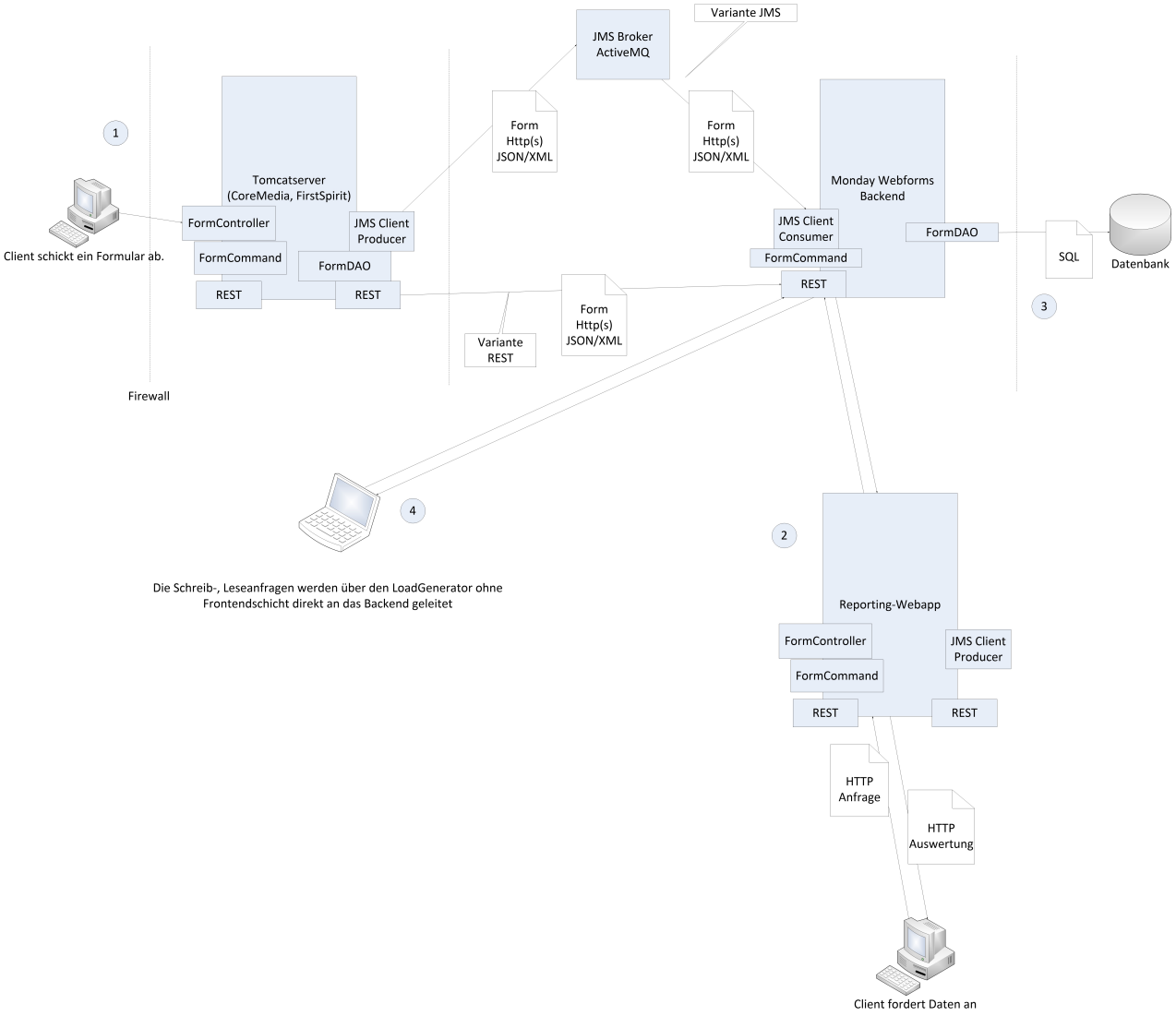

Abbildung 3.1: Architektur der Testumgebung

Ein Webserver, auf dem sich eine CoreMedia oder FirstSpirit-Instanz befindet, nimmt die Anfrage des Clients entgegen und leitet diese per REST[5] oder mit Hilfe von JMS[6]

[5]Representational State Transfer bezeichnet ein Programmierparadigma für verteilte Systeme, wie zum Beispiel Webservices [RES15].

[6]JMS ist ein in Java realisiertes Nachrichtensystem, mit dem ein Sender Nachrichten verschicken und ein Empfänger diese erhalten kann [JMS].

an den Applikationsserver weiter. Auf diesem Server befindet sich das Monday Webforms Backend. Die Anfrage eines Clients kann zwei Formen annehmen: Entweder es wird ein Formular ausgefüllt und abgeschickt (siehe Nummer 1 auf Abbildung 3.1) oder über die Reporting-WebApp sollen Informationen über bereits abgeschickte Formulare dargestellt werden (siehe Nummer 2 auf Abbildung 3.1). Das Monday Webforms Backend kommuniziert anschließend mit der Datenbank (siehe Nummer 3 auf Grafik 3.1) und hat je nach Clientanfrage 2 Hauptaufgaben. Die erste Aufgabe besteht darin, die vom Nutzer verschickten Formulardaten in die Datenbank zu schreiben. Die zweite Aufgabe besteht im Auslesen von Daten aus der Datenbank, die in der Reporting-WebApp dargestellt werden sollen.
In dieser Bachelorarbeit soll das Backend isoliert getestet werden ohne Frontendschicht. Das bedeutet, dass die Reporting-WebApp ebenfalls nicht getestet wird. Aus diesem Grund werden die beiden Webserver, die die WebApp bzw. die CoreMedia-/FirstSpirit-Instanz beinhalten, gegen eine eigens entwickelte Anwendung zur Lasterzeugung ausgetauscht. Diese kommuniziert direkt mit dem Backend auf dem Tomcatserver (siehe Nummer 4 auf Abbildung 3.1).

Das Betriebssystem des Rechners für die Testumgebung ist Ubuntu 14.04. Er besitzt 6 Kerne, die in 12 Threads à 3,30 GHz umgerechnet werden und 16 GB RAM. Die Testumgebung wurde mit Hilfe von virtuellen Maschinen eingerichtet. Dafür wurde die kostenfreie Software VirtualBox verwendet. Es wurden 2 virtuelle Maschinen aufgesetzt. Die virtuellen Maschinen benutzen Ubuntu als Betriebssystem. Der Rechner, auf dem die Lastanwendung läuft, verwendet Windows 10 als Betriebssystem. Im Folgenden werden die beiden virtuellen Maschinen und der Rechner, auf dem die Lastanwendung läuft, vorgestellt.

- **Lasterzeuger:** Hier befindet sich die Lastanwendung, die für die Erzeugung der Last zuständig ist. Über eine Benutzeroberfläche können verschiedene Parameter eingestellt werden. Es gibt die Möglichkeit, über das Monday Webforms Backend Daten in die Datenbank zu schreiben, Daten aus der Datenbank auszulesen oder beide Vorgänge parallel stattfinden zu lassen.

- **Tomcat:** Auf dieser Maschine wurde Tomcat 7.0.52 installiert. Hier befindet sich das deployte Monday Webforms Backend, dass im Rahmen dieser Bachelorarbeit getestet werden soll. Dieses nimmt Anfragen der Lastanwendung entgegen und schreibt Daten in die Datenbank bzw. liest sie aus.

- **Datenbank:** Hier befindet sich die Postgres-Datenbank[7], durch welche die Daten der Formulare gespeichert und bereitgestellt werden. Die installierte Version der Postgres-Datenbank ist 9.3.8.

[7]Postgres oder auch PostgreSQL ist ein freies Datenbankmanagementsystem [pos11]

3.1.2 Motivation für das Projekt Monday Webforms Analytics

Der Hauptgrund für die Entwicklung von Monday Webforms Analytics war, dass das Produkt Webforms als unvollständig erachtet wurde und es bis dahin keinerlei Möglichkeit gab, Informationen über bereits abgeschickte Formulare darzustellen und zu speichern. Vor dem Hintergrund, dass Konkurrenzprodukte diese Funktionalität bereits anboten, sollte sich dies mit Webforms Analytics ändern. Hinzu kommt, dass Konkurrenzprodukte diese Funktionalität bereits anboten und man nachziehen wollte.

3.2 Entwicklung der Lastanwendung

Im Rahmen dieser Arbeit wurde eine funktionsfähige Lastanwendung entwickelt, die es dem Nutzer erlaubt, über eine Oberfläche Formulardaten in eine Datenbank zu schreiben und auszulesen, indem mit dem Monday Webforms Backend kommuniziert wird. Im Folgenden werden die Klassen vorgestellt, die für die Lastanwendung von Bedeutung sind. Die Lastanwendung wurde mit Hilfe des Springframeworks[8] der Programmiersprache Java entwickelt.

LoadController Dies ist der Controller, mit dessen Hilfe der FormSaver bzw. der QueryStressExecutor gesteuert wird.

NeatDataGenerator Diese Klasse generiert Zufallsformulardaten, die vom FormSaver in die Datenbank gespeichert werden.

StressPerformer Dieses Interface bietet Methoden, um Daten aus einer Datenbank auszulesen oder in sie hineinzuschreiben. Es gibt 2 Klassen, die den StressPerformer implementieren, den QueryGetFormSummaryPerformer und QueryGetFormDataPerformer.

FormSaver Diese Klasse bestand bereits unter dem Namen SimpleIntegrationAndPerformanceTest und beinhaltet die Methode saveForm, mit dessen Hilfe die Formulardaten in eine Datenbank geschrieben werden. Diese wurde für eigene Zwecke modifiziert, so dass sie kontinuierlich Last erzeugen kann. Diese Klasse kann mit Hilfe eines Controllers konfiguriert, gestartet und gestoppt werden.

QueryStressExecutor Benutzt den QueryGetFormSummaryPerformer und QueryGetFormDataPerformer, um kontinuierlich Last zu erzeugen. Im Gegensatz zum Schreibvorgang gibt es beim Lesevorgang 2 Methoden, die für diese Arbeit von Interesse sind:

[8] Spring ist ein quelloffenes Framework der für die Java-Plattform. Ziel von Spring ist es, dass Entwickeln mit Java zu vereinfachen und gute Programmierpraktiken zu fördern [Spr15].

getFormData und getFormSummary. Beide Methoden sind zum Auslesen von Formulardaten bestimmt. Der Unterschied besteht darin, dass getFormSummary eine Übersicht aller verfügbaren Formulare darstellt, wohingegen getFormData die jeweiligen Daten eines bestimmten Formulars zurückgibt. Über die Benutzeroberfläche wird die Wahrscheinlichkeit für einen getFormSummary-Aufruf eingestellt. Die Wahrscheinlichkeit für einen getFormSummary und einen getFormData-Aufruf bedingen sich gegenseitig. Würde man beispielsweise für die getFormSummary-Wahrscheinlichkeit einen Wert von 20% angeben, so würde dies bedeuten, dass ein getFormData-Aufruf mit 80 prozentiger Wahrscheinlichkeit stattfindet.

DatabaseClearer Ist dafür zuständig, die Daten einer Datenbank am Ende eines Testdurchlaufs zu löschen.

CSVExportConfiguration Bereitet die nötigen Schritte vor, damit eine CSVExport-Simulation stattfinden kann.

QSETestConfiguration, SIPTestConfiguration Über diese Klassen werden der QueryStressExecutor und der FormSaver konfiguriert.

3.3 Funktionsweise

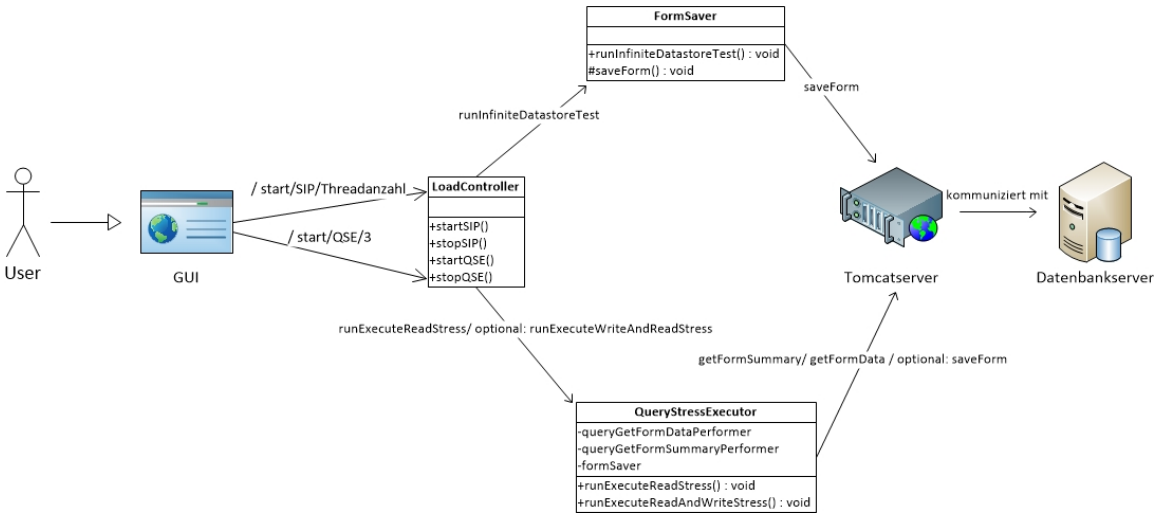

Abbildung 3.2: Funktionsweise der Lastanwendung

Die Funktionsweise der Lastanwendung wird in 3.2 grafisch dargestellt. Diese wird über den LoadController gesteuert. Dieser ist ein REST-Controller und basiert auf einer SpringBoot-Applikation. Er hat die Aufgabe, die 2 Klassen FormSaver und

QueryStressExecutor zu steuern. Dies geschieht über URLs, die von der Benutzeroberfläche an den LoadController gesendet werden. Anhand der URL wird entweder die runInfiniteDatastoreTest-Methode aufgerufen, die die saveForm-Methode beinhaltet oder die runExecuteReadStress-Methode, welche die getFormData- und getFormSummary-Methoden enthalten. Optional kann, wie in 3.4.2 beschrieben wurde, neben dem Auslesevorgang auch ein Schreibvorgang parallel stattfinden. Hierzu würde der LoadController statt der runExecuteReadStress- die runExecuteWriteAndReadStress-Methode aufrufen. Die Klassen FormSaver und QueryStressExecutor kommunizieren mit Hilfe der genannten Methoden mit dem Tomcatserver. Dieser wiederum kommuniziert mit der Datenbank.

3.4 Benutzeroberfläche der Lastanwendung

Abbildung 3.3 zeigt die Oberfläche für die entwickelte Lastanwendung. Die Lasterzeugung kann auf 3 Wegen geschehen:

1. Formulardaten in eine Datenbank zu schreiben
2. Formulardaten aus einer Datenbank auszulesen
3. 1. und 2. parallel laufen zu lassen

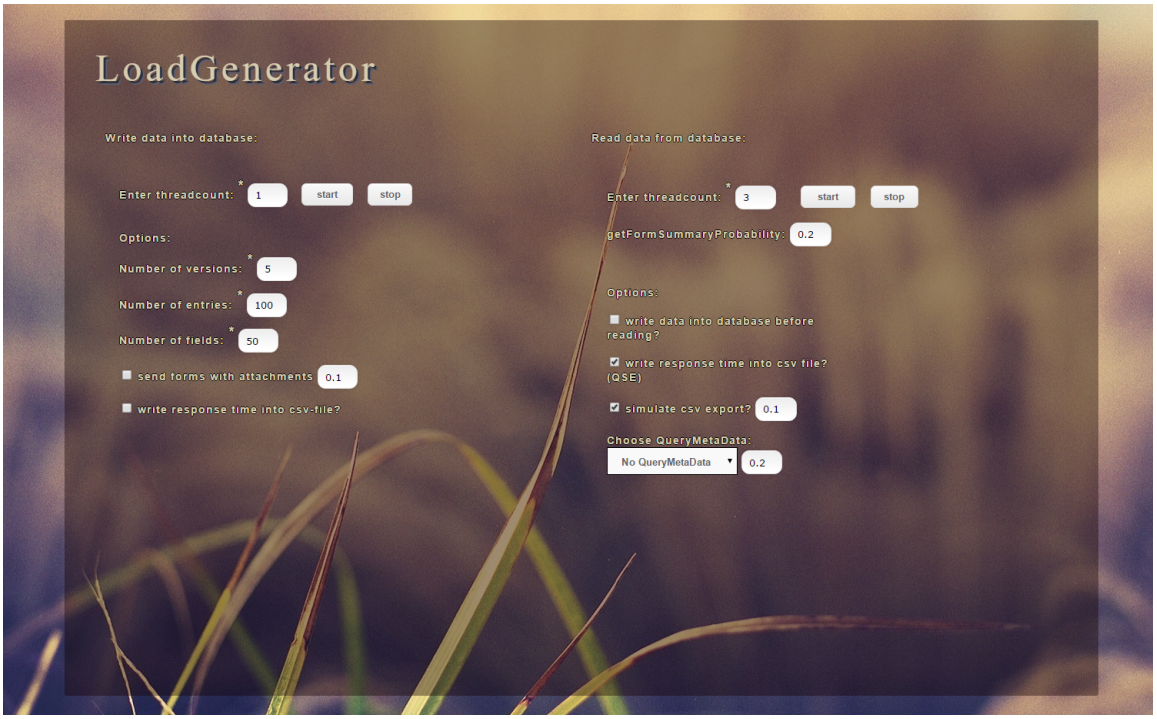

Abbildung 3.3: Benutzeroberfläche für die Lastanwendung

Das Design ist einfach gehalten. Elemente, die zum Schreibvorgang gehören, wurden links angeordnet, die des Lesevorgangs wurden auf der rechten Seite zusammengefasst. Verschiedene Konfigurationen lassen sich über Inputfelder, Checkboxen und ein Dropdownmenü vornehmen, auf die nun näher eingegangen wird.

3.4.1 Konfigurationsmöglichkeiten für den Schreibvorgang

Threadcount Bestimmt die Anzahl der Threads, die gestartet werden sollen, um Last zu erzeugen. Je höher diese Zahl ist, desto mehr Last wird erzeugt.

Number of versions Bestimmt die Anzahl der Versionen, die ein vom NeatData-Generator generiertes Formular annehmen kann.

Number of entries Bestimmt die Anzahl der Einträge für eine Formularvorlage.

Number of fields Bestimmt die Anzahl der Felder der generierten Formulare.

send forms with attachments Wird diese Checkbox gesetzt, so werden die generierten Formulare mit einem Dateinanhang mit einer Größe von ca. 900 Kilobyte verschickt. In das Textfeld daneben wird die Wahrscheinlichkeit für das Vorkommen eines Anhangs eingetragen.

write response time into csv-file? Wird diese Checkbox gesetzt, werden die Antwortzeiten der saveForm-Methode des FormSavers in eine csv-Datei geschrieben.

3.4.2 Konfigurationsmöglichkeiten für den Lesevorgang

Threadcount Wie beim Schreibvorgang kann auch beim Lesevorgang die Threadanzahl eingestellt werden.

getFormSummaryProbability Diese Zahl gibt an, wie wahrscheinlich es sein wird, dass statt einem getFormData-Aufruf ein getFormSummary-Aufruf auftritt.

write data into database before reading? Wird diese Checkbox gesetzt, wird neben dem Auslesevorgang ein weiterer Thread gestartet, der für den Schreibvorgang zuständig ist. Auslese- und Schreibvorgang würden also parallel ausgeführt werden. Es erscheint ebenfalls ein neuer Block, in dem die Konfigurationen für den Schreibvorgang vorgenommen werden können.

write response time into csv file?(QSE) Wird diese Checkbox gesetzt, so werden die Antwortzeiten der beiden Methoden getFormData und getFormSummary in eine csv-Datei geschrieben.

simulate csv export? Bei Setzen dieser Checkbox wird ein csv-Export simuliert.

Choose QueryMetaData Hier kann eine von 3 QueryMetaData[9]-Varianten ausgewählt werden. Zur Verfügung stehen die Optionen: No QueryMetaData, QueryMetaData1, QueryMetaData2.

3.5 Anforderungen der Performancetests

Im Idealfall wird bei einem Performance-/Lasttest ein Soll-Ist-Vergleich ausgeführt. Im Rahmen dieser Bachelorarbeit wird stattdessen ein experimenteller Ansatz gewählt. Es wird also nicht geguckt, ob das System vorher definierten Parametern entspricht, sondern beobachtet, wie sich das System unter der erzeugten Last verhält. Ebenfalls soll die Anzahl der CPUs der virtuellen Maschinen variiert werden, damit ein Vergleich zwischen Last, Antwortzeiten und Anzahl der CPUs stattfinden kann. Der Grund für die experimentelle Vorgehensweise ist, dass vom Arbeitgeber keine konkreten Vorgaben gemacht wurden, wie die Performance des Backends auszusehen hat.

3.6 Monitoring

Monitoring ist das Überwachen von Systemressourcen und stellt eine wichtige Komponente in einem Performancetest dar, da es viele Informationen über die Performance eines Systems liefert. In JMeter wird Monitoring über das Plugin PerfMon realisiert. Dafür muss auf jedem Server, auf dem Systemressourcen überwacht werden sollen, das Plugin heruntergeladen werden. Anschließend wird die PerfMon.bat-Datei gestartet. In der JMeter-Oberfläche des Clients kann nun die IP des jeweiligen Servers angegeben werden. Weiterhin müssen die zu überwachenden Komponenten ausgewählt werden. Für die Performancetests wurden Arbeitsspeicher und CPU-Auslastung überwacht. Um diesen Vorgang zu vereinfachen, wurde ein Batchskript geschrieben, dass das eben genannte Prozedere automatisiert ausführt.

3.7 Entwicklung der Testszenarien

Insgesamt wurden 9 Testszenarien entwickelt und getestet. Diese werden in Grafik 3.4 gezeigt.
Wie in der Abbildung zu erkennen ist, lassen sich alle Testfälle in 3 Kategorien einteilen:

[9] QueryMetaData ist ein Objekt im Webforms Analytics Projekt, das Filteroptionen für bereits bestehende Formulardaten bietet.

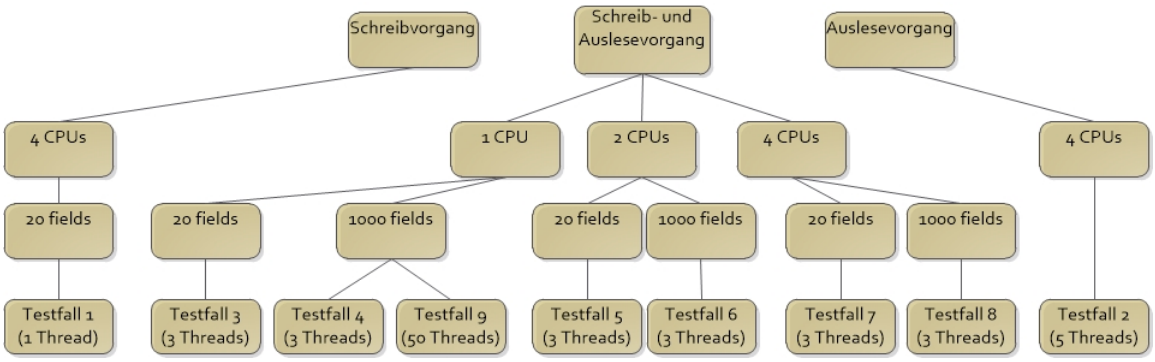

Abbildung 3.4: Szenarien für die durchgeführten Performancetests

1. Reiner Auslesevorgang: Es werden Formulardaten aus einer Datenbank ausgelesen. Es werden keine Daten in die Datenbank geschrieben.

2. Reiner Schreibvorgang: Es werden Formulardaten in eine Datenbank geschrieben, ohne dass welche ausgelesen werden.

3. Auslese- und Schreibvorgang finden parallel statt.

Die meisten Testfälle lassen sich 3. zuordnen, da vor allem dieses Szenario in der Praxis stattfinden wird. Deshalb wurde für den reinen Schreib- und den reinen Lesevorgang lediglich ein Testfall ausgewählt, mit dem Ziel, schreib- bzw. lesespezifisches Lastverhalten der Testumgebung zu beobachten.
Bis auf die Anzahl der CPUs, der Anzahl der Formularfelder und der Threadanzahl wurden über die verschiedenen Testszenarien hinweg fixe Parameter gewählt. Dies dient dem Zweck, eine gewisse Vergleichbarkeit der Testfälle zu ermöglichen. Das Lastverhalten des Backends wird somit vor allem in Bezug auf die Anzahl der CPUs, die Anzahl der Felder eines verschickten Formulars und die Anzahl der verwendeten Threads hin untersucht.

Folgende Einstellungen wurden für die Performancetests gewählt:

Anzahl der CPUs Es wurden Testfälle mit einer CPU, zwei CPUs und vier CPUs gewählt, da hierdurch die Testumgebung auf ihre Skalierbarkeit hin untersucht werden kann (Siehe 2.5).

Threadcount Je nach Testfall wurden 1 Thread, 3 Threads oder 50 Threads gewählt.

getFormSummaryProbability Diese lag bei allen Testfällen bei 20%. Daraus ergibt sich eine 80 prozentige Wahrscheinlichkeit, dass statt einem getFormSummary-Aufruf ein getFormData-Aufruf stattfindet. Die getFormSummary-Wahrscheinlichkeit wurde bewusst kleiner als die getFormData-Wahrscheinlichkeit ausgewählt, da ein Endnutzer diese Funktionalität ebenfalls öfter verwenden wird.

Number of versions Hier wurde ein Wert von 5 gewählt.

Number of entries Hier wurde ein Wert von 100 eingestellt.

Number of fields Je nach Testfall wurde hier ein Wert von 20 oder 1000 gewählt. In der Praxis werden die wenigsten Formulare 1000 Felder haben, dieser Wert wurde bewusst einmal klein und einmal groß gewählt, um zu beobachten, wie sich dies auf die Last auswirkt.

send forms with attachments Diese Checkbox wurde bei jedem Testfall gesetzt und es wurde eine fixe Wahrscheinlichkeit von 20% eingestellt, dass zu einem Formular zusätzlich ein Dateianhang mitgesendet wird.

simulate csv export? Diese Option wurde auch stets gesetzt und mit einer Wahrscheinlichkeit von 10% konfiguriert.

Choose QueryMetaData Es wurde für alle Testfälle QueryMetaData2 ausgewählt. Diese führt ein Paging[10] von 100 Einträgen durch und filtert die Einträge der letzten 3 Tage in zeitlich absteigender Abfolge. Weiterhin wurde eine Wahrscheinlichkeit von 20% eingestellt, dass genau diese QueryMetaData ausgeführt wird. Andernfalls wird eine andere Default-QueryMetaData ausgeführt, die lediglich ein Paging von 10 durchführt, ohne die Einträge in irgendeiner Hinsicht zu filtern und zu sortieren.

3.8 Ablauf der Performancetests

Um etwas über die Zufälligkeit der gewonnenen Daten aussagen zu können (siehe 4.1), wurde jeder der 9 Testfälle 5 mal durchgeführt. Auf einem Server laufen viele weitere Prozesse, die ebenfalls Last erzeugen, jedoch nichts mit der von der Lastanwendung erzeugten Last zu tun haben. Dies könnte die Messergebnisse beeinflussen. Die Testdauer betrug 5 Minuten. Nach jedem Testdurchlauf wurden die virtuellen Server neugestartet und die Datenbank geleert, so dass jeder Testdurchlauf unter

[10] Paging bedeutet in diesem Zusammenhang die Anzahl der Einträge, die aufgelistet werden. Bei der bereits erwähnten Reporting-WebApp würde ein Paging von 100 bedeuten, dass 100 Einträge pro Seite angezeigt werden.

gleichen Startbedingungen stattfinden konnte. Weiterhin wurde die Last erzeugende Maschine auf ihre CPU-Auslastung hin überwacht. Diese sollte nicht über einen längeren Zeitraum über 50% sein. Gemessenen wurden über die Performancetests folgende Parameter:

1. Anzahl der saveForm-Aufrufe pro Sekunde, also wie viele Einträge pro Sekunde in die Datenbank gespeichert werden.
2. Anzahl der readQuery-Methode pro Sekunde, sprich wie oft pro Sekunde, Daten aus der Datenbank ausgelesen werden.
3. CPU-Auslastung der 2 virtuellen Maschinen, auf denen sich der Tomcat und die Datenbank befinden.
4. Arbeitsspeicherauslastung des Tomcatservers und des Datenbankservers.
5. Antwortzeiten der getFormSummary-Methode
6. Antwortzeiten der getFormData-Methode
7. Antwortzeiten der saveForm-Methode

3.9 Fazit

In diesem Kapitel wurde die Architektur einer Umgebung vorgestellt, die das Monday Webforms Backend verwendet. Statt einer typischen, praxisnahen Umgebung wird in dieser Arbeit eine modifizierte Umgebung genutzt, die sich zur praxisnahen darin unterscheidet, dass es keine Frontendschicht gibt. Stattdessen wird über eine entwickelte Lastanwendung direkt mit dem Webforms Backend kommuniziert. Dies wurde damit begründet, dass in dieser Arbeit lediglich das Backend untersucht werden soll. Errichtet wurde die Testumgebung mit Hilfe der Software VirtualBox. Die Testumgebung, die in dieser Arbeit verwendet wird, umfasst 2 virtuelle Maschinen. Auf der einen Maschine wurde eine Tomcat-Instanz installiert, auf der anderen eine Postgres-Datenbank.
Anschließend wurde auf das Projekt Monday Webforms Analytics eingegangen und erklärt was die Motivation für die Entwicklung dieser Software war. Diese bestand im Wesentlichen darin, das Produkt Webforms zu vervollständigen und mit der Funktionalität ähnlicher Produkte konkurrieren zu können. Anschließend wurden relevante Klassen der Lastanwendung und ihre Funktion vorgestellt. Die für das weitere Verständnis wichtigsten Methoden sind die saveForm-Methode zum Speichern von Formulardaten in eine Datenbank sowie die beiden Methoden zum Lesen aus einer Datenbank (getFormSummary und getFormData). Weiterhin wurden die 9 Testszenarien für die Performancetests und ihre Konfigurationsparameter vorgestellt. Die verschiedenen Testszenarien unterscheiden sich in der verwendeten Threadanzahl, der Anzahl der Formularfelder und der Anzahl der verwendeten CPUs. Hierdurch wird eine

Vergleichbarkeit der Testfälle ermöglicht. Überwacht wird das System während eines Testdurchlaufs mit Hilfe der Anwendung JMeter. Gemessen wurden CPU-Auslastung und Arbeitsspeicherauslastung der beiden virtuellen Maschinen, Antwortzeiten und Anzahl der Aufrufe einer saveForm, getFormSummary, getFormData-Methode pro Sekunde.

4 Auswertung der Messergebnisse

In diesem Kapitel werden die Ergebnisse der Performancetests vorgestellt. Zunächst findet eine statistische Analyse der CPU-Auslastung des Tomcatservers von Testfall 4 (siehe 3.7) statt, um einen Anhaltspunkt für die Aussagekraft der Daten zu haben. Anschließend werden prägnante Graphen gezeigt, anhand derer eine Beschreibung des Lastverhaltens stattfindet. Abschließend werden anhand des beobachteten Lastverhaltens der unterschiedlichen Testfälle Hypothesen zum Systemverhalten aufgestellt.

4.1 Statistische Analyse

Zunächst einmal soll die Zufälligkeit der Daten ermittelt werden. Dies wird im Rahmen dieser Bachelorarbeit für ein Anwendungsbeispiel erfolgen, müsste aber strenggenommen für jeden Testfall durchgeführt werden. Für die statistischen Berechnungen wurde die Programmiersprache R verwendet. R ist eine freie Programmiersprache, die sich auf statistisches Rechnen und statistische Grafiken spezialisiert hat.
Die Berechnungen werden am Beispiel der CPU-Auslastung des Tomcatservers von Testfall 4 erfolgen.
Die Überprüfung erfolgt in 2 Schritten. Im ersten Schritt soll überprüft werden, ob die Daten normalverteilt sind. Dies ist wichtig, um im zweiten Schritt zu ermitteln, welches statistische Verfahren angewandt werden soll. Somit ergeben sich zwei Nullhypothesen:

- **Nullhypothese 1:** Die Daten sind normalverteilt.
- **Nullhypothese 2:** Es gibt keine signifikanten Unterschiede innerhalb der Messwiederholungen des Testfalls 4.

Die Nullhypothese 2 ist von besonderem Interesse, da sie die Stärke der späteren Aussagen bestimmt.

Im besten Fall würde die Nullhypothese 2 für jeden Testfall positiv ausfallen. Das würde bedeuten, dass das Lastverhalten eines bestimmten Testfalls mit hoher Wahrscheinlichkeit immer wieder ein spezifisches gleiches Muster abbildet. Würde man also nun beispielsweise das Lastverhalten von Testfall 1 mit dem Verhalten von Testfall 2 vergleichen, so hätten die Aussagen eine hohe Aussagekraft.

Im schlechtesten Fall würde die Nullhypothese 2 für jeden Testfall verworfen. Dies würde bedeuten, dass das Lastverhalten bei jedem Testfall zufälliger Natur wäre. Somit hätten Vergleichsaussagen des Lastverhaltens unterschiedlicher Testfälle eine geringe Aussagekraft.

Im ersten Schritt wird nun also auf Normalverteilung überprüft.
Laut [Bre] besteht die Überprüfung auf Normalverteilung aus drei Schritten:

1. Grafische Überprüfung mit Wahrscheinlichkeitsnetz (QQ-Plot), Histogramm und Boxplot
2. Vergleich der Kennzahlen
3. Tests auf Normalverteilung mit hoher Güte, beispielsweise der Shapiro-Wilks-, Anderson-Darling- und Cramér-von-Mises-Test

Für die Überprüfung auf Normalverteilung werden Punkt 1 und 3 ausgeführt. Wie später zu sehen sein wird, sind die Ergebnisse dieser beiden Schritte sehr eindeutig, was einen Vergleich der Kennzahlen nicht mehr zwingend erforderlich macht.
Für die grafische Überprüfung werden sowohl die Boxplots als auch die Gaußkurven der einzelnen Messreihen miteinander verglichen. Diese werden in 4.1 und 4.2 gezeigt.

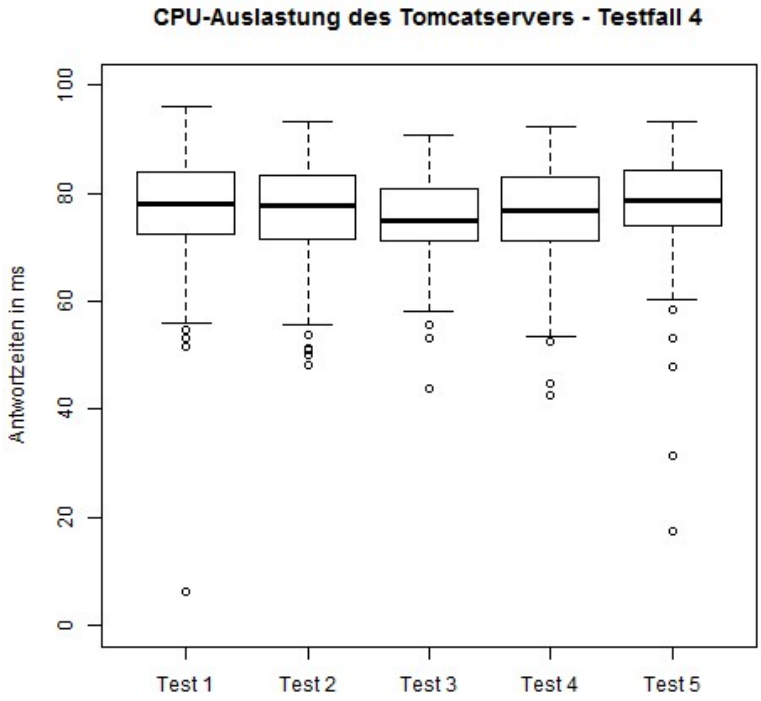

Abbildung 4.1: Boxplots für die 5 Messreihen von Testfall 4

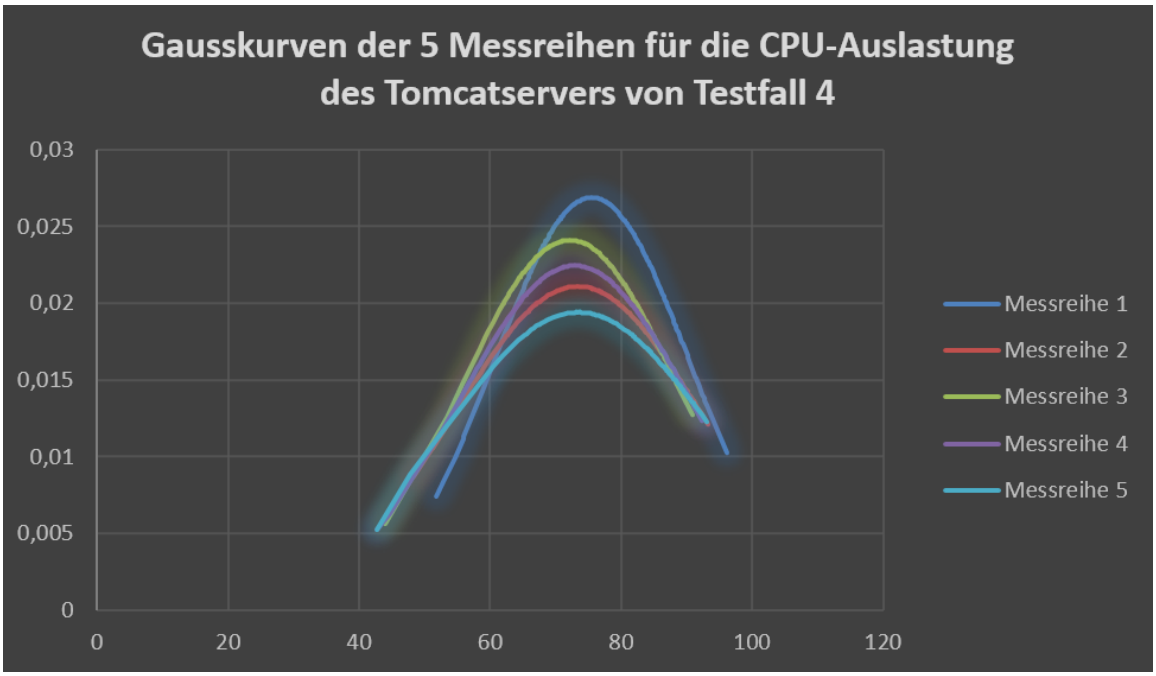

Abbildung 4.2: Gausskurven für die 5 Messreihen von Testfall 4

Die Asymmetrie und die Ausreißer der Boxplots sowie die Asymmetrie der Gausskurven sind erste Indizien, die gegen eine Normalverteilung sprechen.
Anschließend soll laut [Bre] im dritten Schritt ein Test auf Normalverteilung mit hoher Güte stattfinden. Hier gibt es verschiedene zur Auswahl. In diesem Fall wird der Shapiro-Wilk-Test angewandt. Eine hohe Güte sei wichtig, um ein geringes Risiko für die Testenscheidung „Messreihe ist normalverteilt“ zu haben, obwohl tatsächlich die Messreihe nicht normalverteilt ist. Die Ergebnisse des Tests werden in Tabelle 4.1 dargestellt.

Ergebnisse des Shapiro-Wilk-Tests				
Messreihe 1	Messreihe 2	Messreihe 3	Messreihe 4	Messreihe 5
$4.215 * 10^{-13}$	$1.306 * 10^{-05}$	0.0001265	$2.743 * 10^{-05}$	$1.32 * 10^{-14}$

Tabelle 4.1: p-Werte des Shapiro-Wilk-Tests

Da der p-Wert für jede Messreihe weit unter 0,05 liegt, wird die Nullhypothese 1: „Messreihe ist normalverteilt “ für jede Messreihe von Testfall 4 verworfen.
Da die 5 Messreihen von Testfall 4 nicht normalverteilt sind und auch nicht voneinander abhängig sind, wird für die Überprüfung von Nullhypothese 2 der Kruskal-Wallis-Test benutzt. Die Entscheidung für den Test wurde mit Hilfe d auf folgender Quelle [Kel13] getroffen. Der Test lieferte für die 5 Messreihen einen p-Wert von $1.764 * 10^{-06}$. Somit wird die Nullhypothese 2 „Zwischen den Messreihen von Testfall

4 besteht kein signifikanter Unterschied“ verworfen. Auch wenn die einzelnen Messreihen sich optisch zu gleichen scheinen, sind sie doch unterschiedlich genug, so dass ein signifikanter Unterschied zwischen ihnen besteht. Dies kann unter anderem daran liegen, dass auf einem Server weitere Prozesse laufen, die die Last beeinflussen können. Dieser Faktor wurde jedoch in diesem Fall klein gehalten, wie man zum einen an dem Graphen des Idle-Verhaltens (siehe 4.3) erkennen kann. Zum anderen liefen die überwachten Server mit Betriebssystemen ohne Benutzeroberfläche, wodurch Last eingespart werden sollte. Ein weiterer Grund, der zur Verwerfung der zweiten Nullhypothese geführt hat, könnte sein, dass Funktionalitäten zur Lasterzeugung wie beispielsweise das Anfügen eines Anhangs oder die Simulierung eines CSV-Exports wahrscheinlichkeitsbasiert stattfanden. Auch wenn für alle Konfigurationen von Testfall 4 entsprechend die gleichen Wahrscheinlichkeiten angegeben wurden, ist dies keine Garantie dafür, dass beispielsweise die absoluten Vorkommnisse eines CSV-Exports der fünf durchgeführten Messreihen gleich sind. Dies könnte sich auch im Lastverhalten einzelner Messreihen bemerkbar machen und dazu geführt haben, dass sich die Messreihen von Testfall 4 nicht mehr gleichen.

4.2 Beschreibung des Lastverhaltens

Im Folgenden werden einige Aussagen über das Lastverhalten des Backends unter Betrachtung der verschiedenen Testfälle gemacht. Die Ergebnisse der eben verwendeten statistischen Methoden sind ein Indiz, dass die Aussagen vermutlich eine geringe Aussagekraft haben, auch wenn man dies nicht von Testfall 4 auf die anderen Testfälle verallgemeinern kann. Es können jedoch gewisse Tendenzen im Verhalten des Lastverhaltens beobachtet und beschrieben werden, die sich rein graphisch bemerkbar machen.

Idleverhalten der Testumgebung

Abbildung 4.3 zeigt das Idle-Verhalten der 2 überwachten Server.

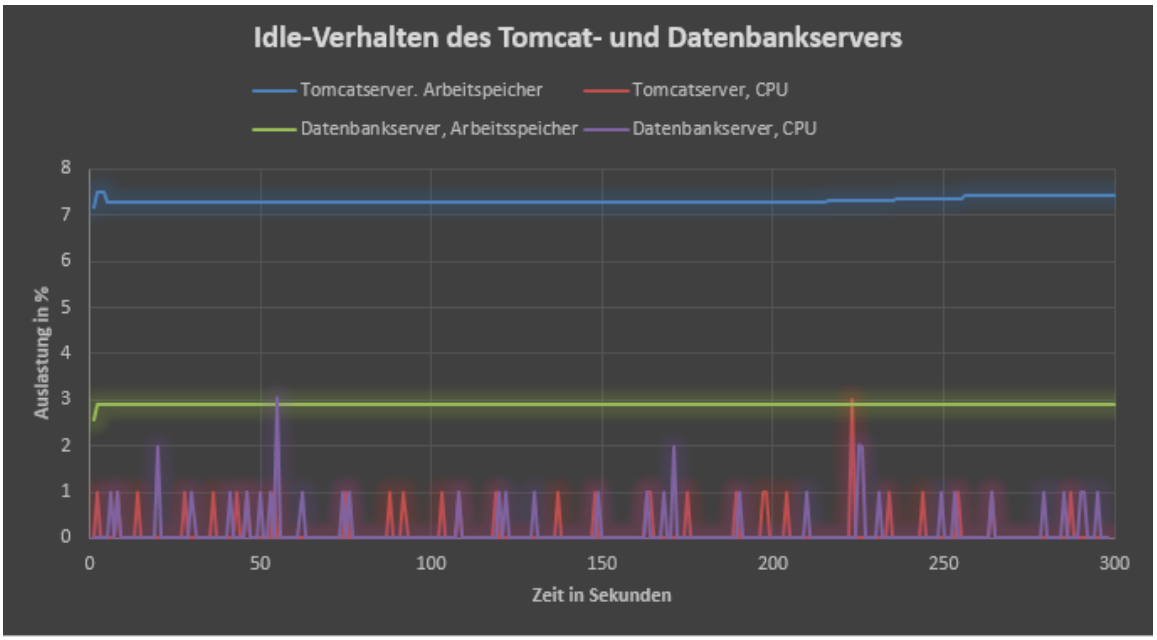

Abbildung 4.3: Auslastung der Server, wenn keine Last von außern erzeugt wird.

Die CPU-Auslastung (lila und rote Linie) ist sehr gering ausgeprägt und pendelt für den Tomcat- als auch den Datenbankserver die meiste Zeit zwischen 0 und 1% mit gelegentlichen Peaks, welche die 3%-Grenze jedoch kaum überschreiten. Die Auslastung des Arbeitspeichers (blaue Linie) des Tomcatservers bleibt recht konstant. Es gibt keine auffälligen Peaks und gegen Ende steigt die Auslastung leicht an. Sie liegt im Durchschnitt bei 7,32 %. Der Arbeitsspeicher des Datenbankservers weist ebenfalls eine gleichmäßige Auslastung auf. Diese liegt bei 2,91%.

Reiner Schreib-/Lesevorgang (Testfall 1 und 2)

Beim reinen Schreibvorgang (Testfall 1, Abbildung 4.4) ist zu beobachten, dass die CPU-Auslastung des Tomcatservers (avg: 19,91%) höher ausfällt als die des Datenbankservers (avg: 1,05%).

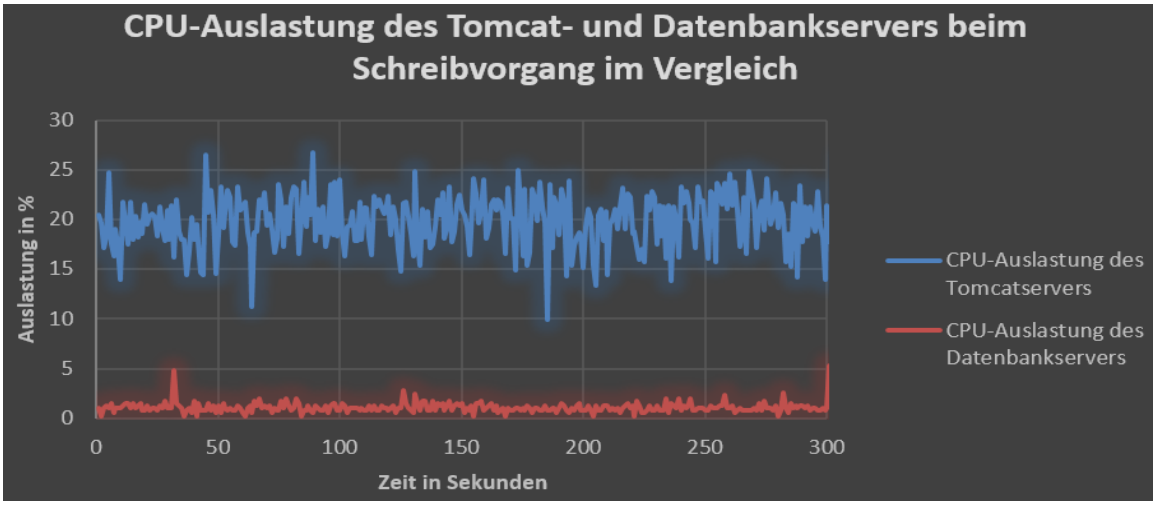

Abbildung 4.4: Last wurde durch einen Thread erzeugt.

Beim reinen Lesevorgang (Testfall 2, Abbildung 4.5) ist das Gegenteil zu sehen: hier liegt die CPU-Auslastung der Datenbank (avg: 82,61) deutlich höher als die des Tomcatservers (avg: 10,17). Die Last ist insgesamt höher ausgefallen, da hier statt einem Thread wie bei Testfall 1 fünf Threads zur Lasterzeugung genutzt wurden.

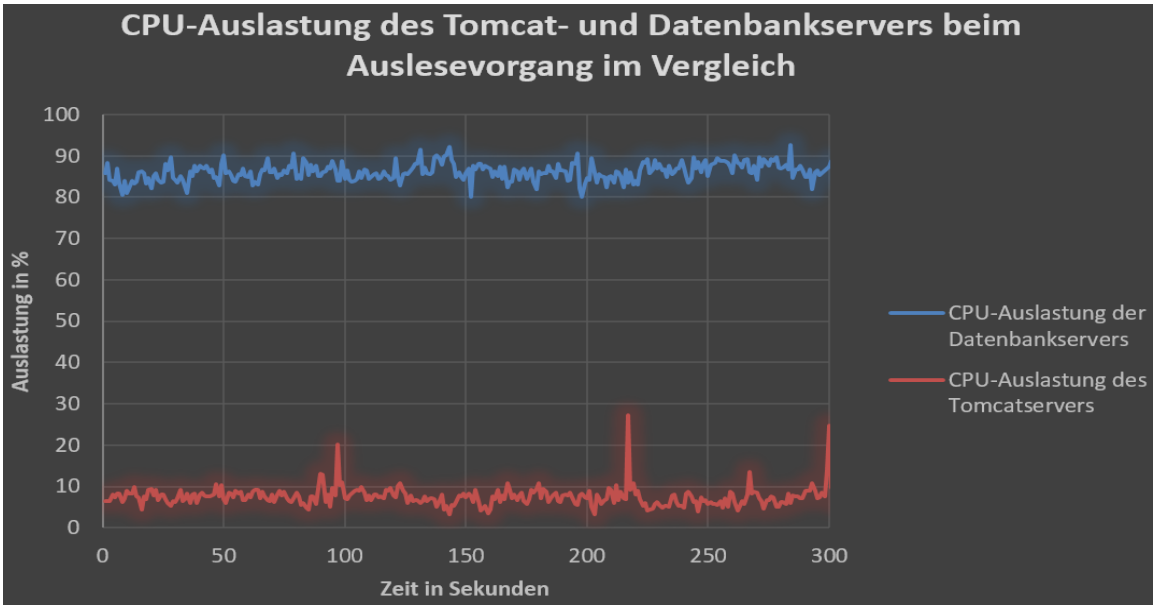

Abbildung 4.5: Last wurde durch 5 Threads erzeugt

Auslastung des Arbeitsspeichers

Der Arbeitsspeicher war sowohl für den Tomcatserver als auch den Datenbankserver über alle Testszenarien hinweg wenig ausgelastet. Der durchschnittliche Maximalwert für die Auslastung des Arbeitsspeichers des Tomcatservers betrug 9.954% für den Datenbankserver 4.221%.

Auf Grafik 4.6 ist zu sehen, dass der Arbeitsspeicher des Tomcatservers mehr ausgelastet wird, als der des Datenbankservers. Exemplarisch wurde dies für Testfall 4 und 7 gezeigt, dies war jedoch für jedes Testszenario der Fall.

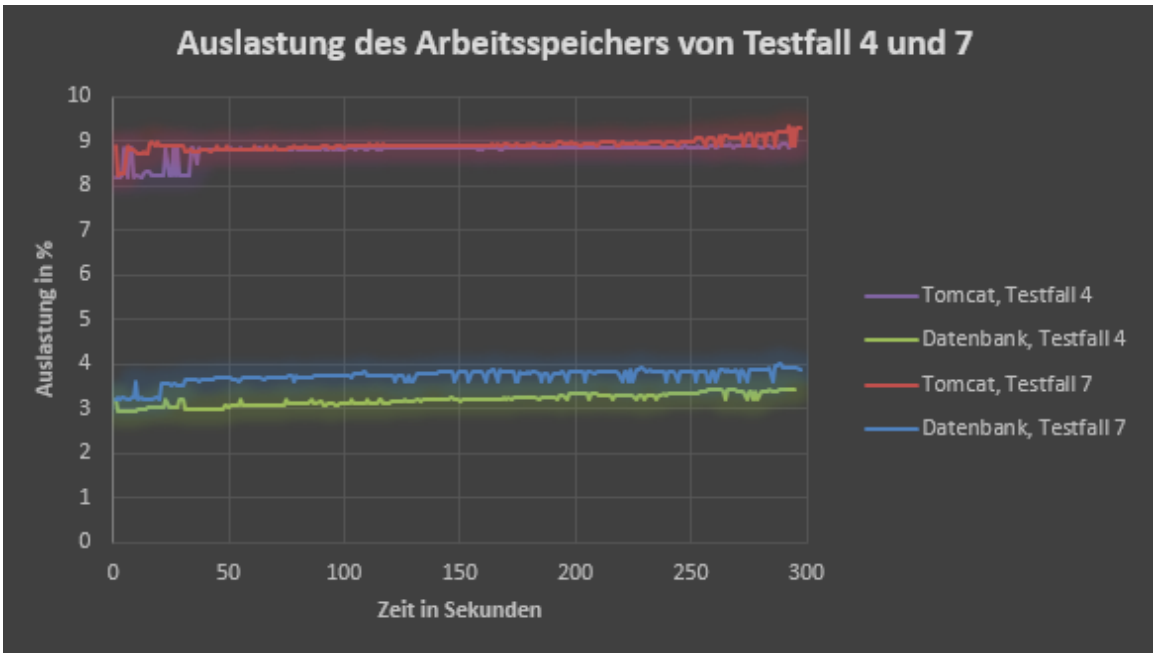

Abbildung 4.6: Auslastung des Arbeitsspeichers

In Tabelle 4.2 wurden die Durchschnittswerte für die Arbeitspeicherauslastung des Idle-Szenarios mit der des worst-case-Szenarios (Testfall 9) und best-case-Szenarios (Testfall 7) verglichen, um zu zeigen, dass die Unterschiede gering sind. Die durchschnittliche Auslastung sowohl des Tomcatservers als auch des Datenbankservers der beiden Testfälle unterscheiden sich zueinander und zum Idle-Szenario nicht mehr als 5%. Ein Grund für die geringe Auslastung des Arbeitsspeichers könnte sein, dass sich wenig Einträge in der Datenbank befanden. Es wäre somit zu überprüfen, ob die Menge der Daten in einer Datenbank sich auf die Auslastung des Arbeitsspeichers auswirken würde.

Durchschnittswerte der Auslastung des Arbeitsspeichers			
Server	Idle	Testfall 7	Testfall 9
Tomcatserver	7,32%	8,93%	10,85%
Datenbankserver	2,91%	3,74%	4,43%

Tabelle 4.2: geringe Unterschiede in der Auslastung zwischen Idle, Testfall 7 und 9

Alle gemessenen Parameter im Überblick (Testfall 3-8)

Auf Abbildung 4.7 werden die Durchschnittswerte aller gemessenen Parameter der Testfälle 3 - 8 dargestellt. Die Auslastung des Arbeitsspeichers wurde an dieser Stelle ausgenommen, da diese bereits betrachtet wurde.

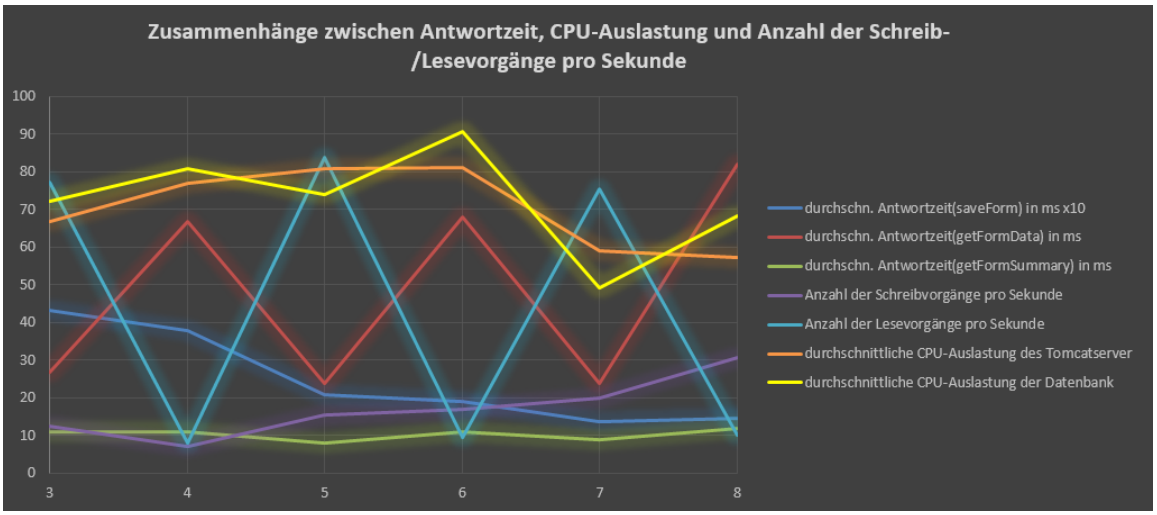

Abbildung 4.7: Durchschnittswerte der gemessenen Parameter im Vergleich

Anhand der Grafik können folgende Beobachtungen gemacht werden:

1. Die Auslastung des Datenbank- und Tomcatservers (gelb und orange) ist bei Testfall 7 und 8 am geringsten. Dies ergibt Sinn, da in Testfall 7 und 8 vier CPUs verwendet wurden, statt zwei oder einer CPU. Dies führt dazu, dass die Last besser verarbeitet werden kann. Dazu passt, dass die Anzahl der Schreibvorgänge pro Sekunde bei 7 und 8 ihr Maximum erreicht. Dies ist auch logisch, wenn die Last die Software nicht behindert, kann sie ungestört arbeiten, was sich auch an den niedrigeren Antwortzeiten der saveForm-Methode (dunkelblau) bemerkbar macht. Ebenfalls zu beobachten ist, dass die Auslastung beider Server bei den geraden Testfällen höher ist als bei den ungeraden. Dies hängt mit der Anzahl der Felder der verschickten Formulare zusammen und wird in Punkt 4 näher erläutert.

2. Weiterhin ist zu erkennen, dass die Antwortzeiten der getFormSummary-Methode (grün) über alle Testfälle hinweg niedriger sind als die der getFormData-Methode (rot). In Testfall 7 und 8 scheinen sie sich anzugleichen, weil die Antwortzeiten der getFormData-Methode hier am niedrigsten sind. Die Antwortzeiten der Methode getFormSummary bleiben über die Testfälle hinweg recht konstant. Es kann festgehalten werden: Die Anzahl der CPUs wirkt sich auf die Antwortzeiten der getFormData-Methode aus, aber nicht bzw. kaum auf die Antwortzeiten der getFormSummary-Methode. Die Antwortzeiten der saveForm-Methode (dunkelblau) sind größer als die der Auslese-Methoden. Dies ist in Grafik 4.7 nicht zu sehen, da die Antwortzeiten der saveForm-Methode durch 10 geteilt werden mussten, um grafisch mit den anderen Werten zusammen dargestellt werden zu können.

3. Die Auslastung des Tomcat- und Datenbankservers haben ihr Maximum bei Testfall 5 und 6. Hier wurde mit zwei CPUs getestet bei Testfall 3 und 4 hingegen mit nur einer CPU. Hier wäre zu erwarten, dass dementsprechend die Last bei Testfall 3 und 4 höher sein müsste, da weniger Ressourcen für die Verarbeitung der Last zur Verfügung stehen.

4. Ebenfalls zu beobachten ist, dass in den geraden Testfällen (2, 4, 6, 8) die Antwortzeiten der getFormData-Methode höher sind als die der ungeraden Testfälle. Dies liegt daran, dass in den geraden Testfällen die Anzahl der Felder der verschickten Formulare größer sind als die der ungeraden Testfällen. Es beansprucht mehr Ressourcen, Formulare mit 1000 Feldern zu verarbeiten als Formulare, die 20 Felder groß sind. Dazu passend ist der Verlauf der hellblauen Linie, die die Anzahl der Lesevorgänge pro Sekunde repräsentiert. Sie hat gegensätzlich zur roten Linie ihre Maxima immer da, wo die rote Linie ihre Minima hat. Dies ist verständlich, da bei niedrigeren Antwortzeiten, der Lesevorgang öfter stattfinden kann.

Überlastung (Testfall 9)

Testfall 9 (Grafik 4.8) wird einzeln betrachtet, da vor allem die Auslastung der Datenbank ein anderes Muster zeigt, als das der restlichen Testfälle.

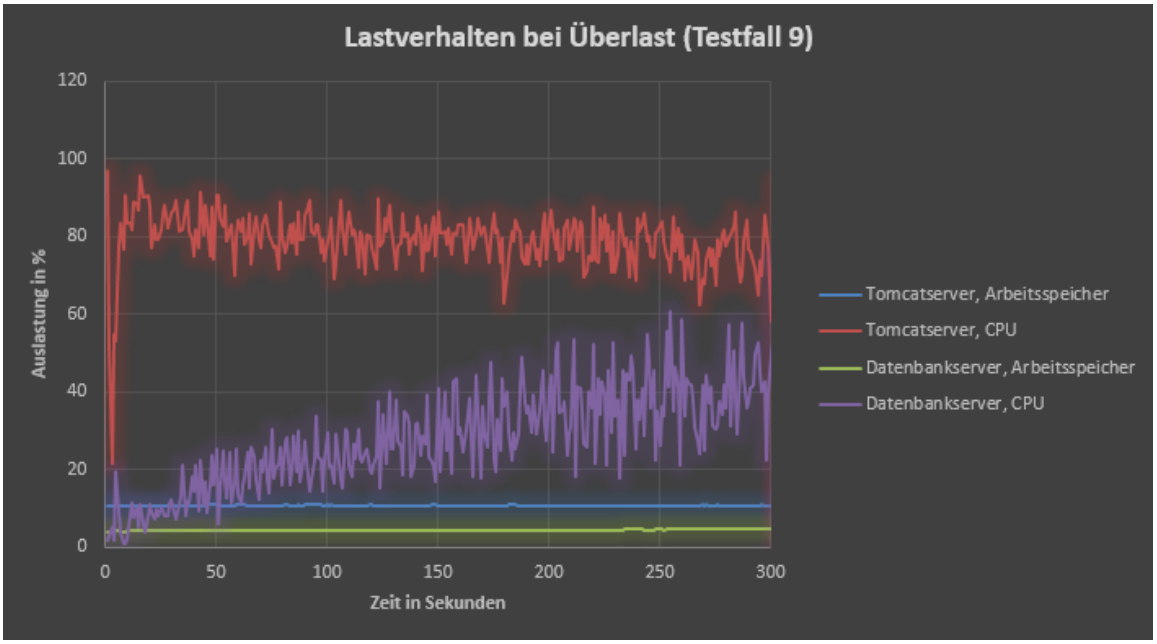

Abbildung 4.8: Testfall 9 wurde mit übertriebener Last gemessen, um zu sehen, wie das System sich bei Überlastung verhält

In den Testfällen 3-8 zeigt das Lastverhalten der Datenbank bis auf einige Peaks eine gewisse Gleichmäßigkeit (siehe 4.5 und 4.4). Diese ist in Testfall 9 nicht zu erkennen (lila Linie). Hier ist die Auslastung am Anfang gering und steigt mit der Zeit stetig an. Beim Tomcatserver ist das Gegenteil der Fall (rote Linie). Eine mögliche Begründung für dieses Verhalten ist, dass der Tomcatserver zunächst überlastet ist, wodurch keine Anfragen an die Datenbank weitergeleitet werden können. Nach einer gewissen Zeit wirken Caching-Mechanismen des Tomcats, die eine Verringerung der Antwortzeiten der saveForm-Methode zur Folge haben. Dadurch nimmt die Überlastung des Tomcatservers ab, wodurch mehr Anfragen an die Datenbank geleitet werden. Dies resultiert in einer Erhöhung der Auslastung des Datenbankservers.
Betrachtet man die Antwortzeiten (Grafik 4.9), so sieht man, dass die Antwortzeiten der saveForm-Methode mit der Zeit stetig abnehmen. Dies passt zum Anstieg der CPU-Auslastung der Datenbank.

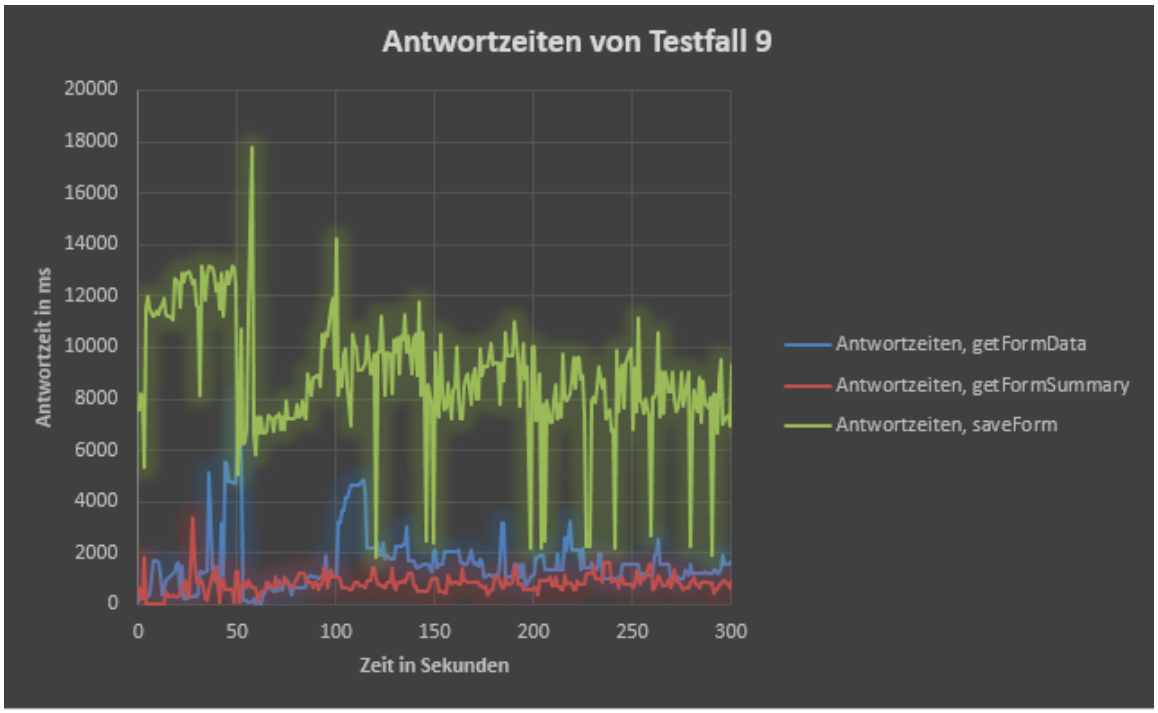

Abbildung 4.9: Die Antwortzeiten der saveForm-Methode werden stetig kleiner

Eine klassische Zerfallskurve wie in 2.4.1 beschrieben wurde, konnte nicht nachgestellt werden. Ab einer gewissen Threadzahl (>50) kam es dazu, dass der Tomcat der Testumgebung SocketExceptions geworfen hat. Dies führte dazu, dass Last erzeugende Threads geschlossen wurden. Es wird vermutet, dass die Antwortzeiten zu hoch waren und zu viele Verbindungen zum Tomcat bestanden, so dass die Last nicht mehr verarbeitet werden konnte. Dies führte dazu, dass auf dem System kein Totalausfall herbeigeführt werden konnte.

Wie viel Last kann das System verarbeiten?

Wie viel Last das Monday Webforms Backend verarbeiten kann, hängt von mehreren Parametern ab. Darunter fallen unter anderem die verwendeten Hardwareressourcen und die Größe der verschickten Formulare. Tabellen 4.3 und 4.4 zeigen die Durchschnittswerte der Anzahl der Schreib- und Lesevorgänge pro Sekunde für die Messreihen der Testfälle 3 - 8 und die dazugehörigen Antwortzeiten. Hier ist zu sehen, dass die Größe der Formulare sich vor allem auf die Anzahl der Lesevorgänge pro Sekunde äußert. Testfall 4, 6 und 8, die Formulare mit 1000 Feldern verwendet haben, weisen viel kleinere Werte auf als die ungeraden Testfälle 3, 5 und 7, in denen Formulare mit 20 Feldern verwendet wurden. Ein derartiges Muster ist bei der Anzahl der Schreibvorgänge unter diesen Bedingungen nicht zu erkennen. Der Tabelle ist zu entnehmen, dass das Backend maximal 30,6 Einträge pro Sekunde in die Datenbank schreiben konnte. Was die Auslesevorgänge betrifft, lag das Maximum bei 83,8 Lesevorgängen

pro Sekunde. An dieser Stelle wären Referenzwerte aus der Praxis interessant, um zu vergleichen, inwiefern die hier ermittelten Werte den Anforderungen des Alltags gerecht werden.

Anzahl der Schreib- und Lesevorgänge pro Sekunde		
Testfall	Schreibvorgänge pro Sekunde	Lesevorgänge pro Sekunde
Testfall 3	12,6	77,4
Testfall 4	7,2	8
Testfall 5	15,4	83,8
Testfall 6	17	9,6
Testfall 7	20	75,6
Testfall 8	30,6	10,2

Tabelle 4.3

Antwortzeiten der Testfälle 3-8			
Testfall	saveForm in ms	getFormData in ms	getFormSummary in ms
Testfall 3	43,3	27	11
Testfall 4	37,8	67	11
Testfall 5	20,8	24	8
Testfall 6	19,1	68	11
Testfall 7	13,6	24	9
Testfall 8	14,5	82	12

Tabelle 4.4: Antwortzeiten für die Schreib- und Lesevorgänge

5 Fazit

Im Rahmen dieser Bachelorarbeit wurde eine funktionsfähige Testumgebung aufgebaut, die verwendet wurde, um das Monday Webforms Backend auf seine Performance hin zu testen. Die Last wurde durch eine in Java entwickelte Anwendung erzeugt. Für die Überwachung des Systems während der Performancetests wurde das Lastwerkzeug JMeter genutzt, um Informationen über das Verhalten der CPUs und des Arbeitsspeichers zu gewinnen. Nach der Durchführung der Tests fand eine statistische Analyse statt, die indiziert, dass die Daten eine geringe Aussagekraft haben. Die wichtigsten Beobachtungen, die sich bei Betrachtung der in Kapitel 4 vorgestellten Graphen ergeben haben, sollen nachfolgend zusammengefasst wiedergegeben werden.

Das Monday Webforms Backend lastet vor allem die CPU aus, wobei man zwischen den verschiedenen Testszenarien große Unterschiede feststellen kann. Der Arbeitsspeicher war hingegen über die gesamte Testdauer hinweg gering ausgelastet. Sowohl für den Tomcat- als auch den Datenbankserver unterschied sich die Auslastung des Arbeitsspeichers während der Durchführung der einzelnen Tests nicht wesentlich von der des Idle-Verhaltens (Unterschied $< 4\%$). Hierbei muss jedoch erwähnt werden, dass lediglich eine Datenbank mit wenig Einträgen verwendet wurde, die nach jedem Testdurchlauf geleert wurde. Eine Möglichkeit für weitere Testfälle wäre demnach eine Datenbank mit vielen Einträgen zu verwenden, was einen Anstieg der Arbeitsspeicherauslastung zur Folge haben könnte.
Die CPU-Auslastung beider Server hängt mit der Anzahl der für die virtuellen Maschinen konfigurierten CPUs zusammen. Eine klare Tendenz im Sinne von mehr verwendete CPUs haben eine geringere CPU-Auslastung zur Folge ist nicht zu erkennen. Deutlich zu erkennen ist jedoch, dass die Testfälle mit vier verwendeten CPUs eine deutlich geringere CPU-Auslastung aufweisen als die mit zwei oder einer verwendeten CPU. Die Anzahl der eingesetzten CPUs äußert sich ebenfalls in den Antwortzeiten der saveForm-Methode sowie in der Anzahl der Schreibvorgänge pro Sekunde, welche die Datenbank betreffen. Hier ist eine klare Tendenz zu erkennen: Je mehr CPUs die Server verwenden desto kleiner werden die Antwortzeiten und desto größer wird die Anzahl der Schreibvorgänge pro Sekunde.
Steht das System unter starker Last, äußert sich dies zunächst in der CPU-Auslastung des Tomcatservers, bei vergleichsweise sehr niedriger CPU-Auslastung der Datenbank. Nach einiger Zeit verringert sich die Auslastung des Tomcatservers bei gleichbleibender Last, was vermutlich auf einen Caching-Mechanismus zurückzuführen ist. Dies hat zur Folge, dass vermehrt Anfragen an die Datenbank gelangen können, was sich in einem Anstieg der CPU-Auslastung bemerkbar macht.

Ein Totalausfall des Systems konnte im Rahmen dieser Bachelorarbeit nicht herbeigeführt werden. Dies liegt unter anderem daran, dass bei hoher Threadzahl (>50) zur Lasterzeugung, dieser eine oder mehrere Socket-Exceptions geworfen hat (vermutlich aufgrund extrem hoher Antwortzeiten und hoher Anzahl an Verbindungen zum Tomcat). Diese hatten die Schließung einiger Last erzeugenden Threads zur Folge, wodurch die eintreffende Last abnahm.
In dieser Arbeit wurde mit einer fixen Dateianhanggröße von ca. 900 KB getestet. Um mehr Aufschluss über das Systemverhalten in Bezug auf die Anhanggröße zu bekommen, wären weitere Testszenarien sinnvoll, die eine Variation dieser beinhalten.

Weiterhin sollte das System auf sein Langzeitverhalten hin überprüft werden. Hier könnte ein Dauerlasttest stattfinden, der über einen langen Zeitraum hinweg (>48 Stunden) vollzogen wird.

Abschließend soll erwähnt werden, dass die Antwortzeiten, die für das Monday Webforms Backend ermittelt wurden, nicht mit den Antwortzeiten realer Systemumgebungen gleichzusetzen sind. Dies liegt daran, dass das Backend im Normalfall nicht, wie in der Testumgebung dieser Arbeit alleinstehend arbeiten wird. Realistische Umgebungen werden wie in 3.1.1 bereits erwähnt, zusätzlich eine Frontendschicht beinhalten, beispielsweise ein Webformsformular einer CoreMedia- oder FirstSpirit-Instanz und die in 3.1.2 erwähnte Reporting-Webapp. In 2.5 wurde beschrieben, dass die Antwortzeiten einer Anwendung zum großen Teil von der Clientseite abhängen. Somit würden die endgültigen Antwortzeiten für einen Nutzer unter den gleichen Testfällen und unter Verwendung der Frontendschicht höher ausfallen als die für das Backend ermittelten. In diesem Sinne wäre es ebenfalls interessant, das komplette System unter praxisnahen Bedingungen zu testen.

A Anhang Datenträger

Auf dem beigefügten Datenträger befindet sich der Quelltext der entwickelten Lastanwendung. Dieser liegt im Ordner LoadGenerator.

Die Messergebnisse befinden sich im Ordner Messergebnisse. Im Unterordner R befinden sich der Quelltext zur Auswertung der Messergebnisse, im Unterverzeichnis Graphs befinden sich die Graphen der Messergebnisse.

Der Ordner Batchfiles enthält ein Batchskript, das in dieser Arbeit verwendet wurde, um redundante Arbeitsschritte für die Performancetests automatisiert ablaufen zu lassen.

Der Ordner Bilder enthält alle erstellten Bilder, die in der Bachelorarbeit verwendet wurden.

Eine elektronische Version der Arbeit im PDF-Format befindet sich im Verzeichnis Ausarbeitung. Das gesamte LaTex-Projekt ist unter dem Ordner LaTex zu finden.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Bar07] Barber, Scott: *How to Identify the Usual Performance Suspects.* http://www.logigear.com/magazine/test-process-improvement/how-to-identify-the-usual-performance-suspects/, 2007. – letzter Zugriff: 16.12.2015

[Bre] Bredner, Barbara: *Statistische Beratung und Lösungen - Prüfung auf Normalverteilung.* http://www.bb-sbl.de/tutorial/verteilungen/ueberpruefungnormalverteilung.html. – letzter Zugriff: 16.12.2015

[Bri] Brink, Sascha: *Was ist AngularJS?* https://angularjs.de/artikel/was-ist-angularjs. – letzter Zugriff: 16.12.2015

[Chea] Chece, Sebastian: *Performancetests ohne "Last".* https://www.testing-board.com/performancetests-ohne-last/. – letzter Zugriff: 16.12.2015

[Cheb] Chece, Sebastian: *Warum Last- und Performancetest? Gute Performance macht den Unterschied!* https://www.testing-board.com/warum-last-und-performancetest. – letzter Zugriff: 16.12.2015

[Chm09] Chmielewski, P.: *Methoden zur kontinuierlichen Performancemessung bei der agilen Entwicklung von Webanwendungen.* http://www.inf.fu-berlin.de/inst/ag-se/theses/Chmielewski09-performancemessung.pdf, 2009. – letzter Zugriff: 16.12.2015

[Cor] *Implementierungspartner.* http://www.coremedia.com/web-content-management/partner/implementierungs-partner/-/6152/6152/-/_h652o9/-/index.html. – letzter Zugriff: 18.12.2015

[Dir06] Dirlewanger, Werner: *Measurement and Rating of Computer Systems Performance and of Software Efficiency.* http://www.uni-kassel.de/upress/publik/Leseprobe_Dirlewanger.pdf, 2006. – letzter Zugriff: 16.12.2015

[Fir] *eSpirit Partner Übersicht.* http://www.e-spirit.com/de/partner/partner-startpage.html. – letzter Zugriff: 18.12.2015

[Fra07] Franz, Klaus: *Handbuch zum Testen von Webapplikationen.* Springer-Verlag, 2007

[Haa] Haarmann, M.: *Scaling Walkthrough*. http://de.slideshare.net/derwildemomo/facebook-10025711. – letzter Zugriff: 16.12.2015

[HG97] Hu, Lei ; Gorton, Ian: *Performance Evaluation for Parralel Systems: A survey*. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.12.9079&rep=rep1&type=pdf, 1997. – letzter Zugriff: 16.12.2015

[Hol02] Holtkamp, Heiko: *Einführung in TCP/IP*. Technisch In: Queue 6, Universität Bielefeld, 2002. – letzter Zugriff: 16.12.2015

[ITW] ITWissen: *Latenz*. http://www.itwissen.info/definition/lexikon/Latenz-latency.html. – letzter Zugriff: 16.12.2015

[JHK05] Junick, S. ; Herrmann, P. ; Karge, I.: *Lastttests unter „Linux for zSeries“*. K.G. Saur Verlag, München, 2005. – letzter Zugriff: 16.12.2015

[JMS] *JMS Grundlagen*. http://www.straub.as/java/jms/basic.html. – letzter Zugriff: 16.12.2015

[Kel13] Keller, Daniela: *Statistischer Vergleich von mehr als zwei Gruppen*. http://www.statistik-und-beratung.de/2013/08/statistischer-vergleich-von-mehr-als-zwei-gruppen/, 2013. – letzter Zugriff: 16.12.2015

[Kin08] King, Andrew B.: *Website Optimization*. O'Reilly Media, http://www.websiteoptimization.com, 2008. – letzter Zugriff: 16.12.2015

[KR13] Kuhn, Daniel ; Raith, Michael: *Performante Webanwendungen*. dpunkt.verlag GmbH, 2013

[Lei08] Leighton, Tom: *Improving Performance on the Internet*. In: Queue 6, 2008. – letzter Zugriff: 16.12.2015

[lex09] lexikon.martinvogel.de: *Bottleneck*. http://lexikon.martinvogel.de/bottleneck.html, 2009. – letzter Zugriff: 16.12.2015

[Lin06] Linden, Greg: *Make Data Useful*. http://glinden.blogspot.de/2006/11/marissa-mayer-at-web-20.html, 2006. – letzter Zugriff: 16.12.2015

[May08] Mayer, Marissa: *Google I/O 2008 Keynote*. 2008. – letzter Zugriff: 16.12.2015

[NBK02] Nahum, Erich ; Barzilai, Tsipora ; Kandlur, Dilip D.: *Performance issues in WWW servers*. http://www.cs.columbia.edu/~nahum/papers/ton02-www-camera.pdf, 2002. – letzter Zugriff: 16.12.2015

[pos11] *Häufig gestellte Fragen (FAQ) zu PostgreSQL*. http://sql-info.de/postgresql/FAQ_german.html, 2011. – letzter Zugriff: 16.12.2015

[qui14] *Pilawa: „Am liebsten würde ich jetzt Tschüss sagen'".* http://www.focus.de/kultur/kino_tv/joerg-pilawa-quizduell-app-desaster-quizduell-rueckzugswunsch-nach-applaus_id_3863655.html, 2014. – letzter Zugriff: 16.12.2015

[rad] RADWARE: *What Does FastView Do?* http://www.radware.com/Products/FastView/. – letzter Zugriff: 16.12.2015

[Res08] RESIG, John: *querySelectorAll in Firefox 3.1.* http://ejohn.org/blog/queryselectorall-in-firefox-31/, 2008. – letzter Zugriff: 16.12.2015

[RES15] *Representational State Transfer.* https://de.wikipedia.org/wiki/Representational_State_Transfer, 2015. – letzter Zugriff: 16.12.2015

[Ris05] RISSE, T.: *Design and Configuration of Distributed Job Processing Systems.* http://tuprints.ulb.tu-darmstadt.de/665/1/thomas_risse_diss.pdf, 2005. – letzter Zugriff: 16.12.2015

[Sch06] SCHLOSSNAGLE, T.: *Scalable internet architectures.* Sams Indianapolis, IN, USA, 2006. – letzter Zugriff: 16.12.2015

[Sch12a] SCHIRMER, Sven: *Die zehn grössten Fehler im Software-Test-Management.* http://www.computerwoche.de/a/die-zehn-groessten-fehler-im-software-test-management,1877489, 2012. – letzter Zugriff: 16.12.2015

[Sch12b] SCHULZ, Andreas: *Last- und Performance-Tests optimal durchführen.* http://www.it-administrator.de/themen/netzwerkmanagement/fachartikel/117300.html, 2012. – letzter Zugriff: 16.12.2015

[SL10] SPILLNER, Andreas ; LINZ, Tilo: *Basiswissen Softwaretest.* dpunkt.verlag GmbH, 2010

[Sou08] SOUDERS, Steve: *High Performance Web Sites.* In: Queue 6, 2008. – letzter Zugriff: 16.12.2015

[Spr15] *Spring (Framework).* https://de.wikipedia.org/wiki/Spring_(Framework), 2015. – letzter Zugriff: 16.12.2015

[Sto08] STOLZE, Jirk: *Performancetests und Bottleneck-Analyse in Mehrschichtarchitekturen.* http://www.performance-test.de/performancetests-begriffsdefinitionen.htm, 2008. – letzter Zugriff: 16.12.2015

[Tes15] TESTAUTOMATISIERUNG.ORG: *Bedeutung von Performance im Privat- und Freizeitbereich.* https://www.testautomatisierung.org/warum-last-und-performancetest/, 2015. – letzter Zugriff: 16.12.2015

[Tra15] *Transaktion (Informatik).* `https://de.wikipedia.org/wiki/Transaktion_(Informatik)`, 2015. – letzter Zugriff: 16.12.2015

[Vog09] Vogel, Ronny: *Begriffe erklärt: Lasttest, Stresstest,...* `https://blog.xceptance.com/2009/09/16/begriffe-erklart-lasttest-stresstest/`, 2009. – letzter Zugriff: 16.12.2015

[WFM06] Whitworth, Brian ; Fjermestad, Jerry ; Mahinda, Edward: *The web of system performance.* `http://www.brianwhitworth.com/158729/WOSP-CACM-2006.pdf`, 2006. – letzter Zugriff: 16.12.2015

[Wir] Wirtschaftslexikon, Gabler: *Content Management System (CMS).* `http://wirtschaftslexikon.gabler.de/Definition/content-management-system-cms.html`. – letzter Zugriff: 16.12.2015

[YF10] Yang, Helen ; Faris, Noelle: *New Study Reveals the Impact of Travel Site Performance on Consumers.* `http://www.akamai.com/html/about/press/releases/2010/press_061410.html`, 2010. – letzter Zugriff: 16.12.2015

Ich versichere, die vorliegende Arbeit selbstständig ohne fremde Hilfe verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt zu haben. Die aus anderen Werken wörtlich entnommenen Stellen oder dem Sinn nach entlehnten Passagen sind durch Quellenangaben eindeutig kenntlich gemacht.

Ort, Datum Christian Lehr