MASTERARBEIT

AUSWIRKUNGEN VON ELASTIZITÄT AUF PERFORMANZ UND KOSTEN IN EINEM MANDANTENFÄHIGEN, CONTAINERBASIERTEN WEBSHOP-SYSTEM

ausgeführt am

Studiengang
Informationstechnologien und Wirtschaftsinformatik

Von: Florian Winkler
Personenkennzeichen: 1910320027

Graz, am 04. Dezember 2020 .....

Unterschrift

EHRENWÖRTLICHE ERKLÄRUNG

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

.....

Unterschrift

DANKSAGUNG

Mein Dank gilt den Personen, die mir mein Studium und das Verfassen dieser Arbeit ermöglicht haben.

Mein größter Dank ergeht …

…an meine Familie für den allgegenwärtigen und bedingungslosen Rückhalt
…an meinen Betreuer DI Markus Petelinc, BSc für die Ratschläge und Unterstützung
…an die Geschäftsführer der TAC für das Entgegenkommen und die Flexibilität
…an meine Studienkollegen für das Teamwork im Studium
…an meine KorrekturleserInnen für die Zeit und den produktiven Input
…an meine Freunde für den Ausgleich, um wieder einen klaren Kopf zu bekommen

KURZFASSUNG

Der permanent wachsende Online-Markt steigert die Nachfrage erschwinglicher Webshop-Systeme, damit Webshops nicht nur dem Online-Auftritt von Großunternehmen vorbehalten bleiben, sondern auch im breiten Markt der KMU Einzug finden. Gleichzeitig haben sich mit dem Aufstieg von Cloud-Plattformen und dem häufig damit verbundenen „Pay-per-Use“-Prinzip ganz neue Anforderungen an die Effizienz der Ressourcennutzung von Software-Systemen ergeben. Zentrales Ziel dabei ist die Einsparung von Kosten bei gleichzeitiger Erhaltung der Performanz. Die Systemeigenschaft Elastizität ermöglicht es, diesen speziellen Anforderungen gerecht zu werden. Für die Umsetzung von Elastizität in einem mandantenfähigen Webshop-System bietet sich die Bereitstellung in der Cloud in Verbindung mit Container-Technologien und der Verwendung eines geeigneten Cluster-Management-Systems an.

Mit diesem Hintergrund ist das Ziel der vorliegenden Arbeit die Messung der Auswirkungen von Elastizität auf Performanz und Kosten in einem mandantenfähigen, containerbasierten Webshop-System. Um diese Messung durchzuführen, werden zunächst die Rahmenbedingungen abgesteckt sowie die zugehörigen Metriken und Messmethoden vorgestellt. Anschließend werden zwei unabhängige Testsysteme aufgebaut, wovon sich das eine nur durch seine elastischen Eigenschaften vom anderen unterscheidet. Die beiden Testsysteme werden denselben Auslastungen ausgesetzt, um jeweils die Performanz und Kosten zu ermitteln.

Durch den Vergleich der Ergebnisse lässt sich eine deutliche Verbesserung der Performanz durch Elastizität belegen. Allerdings wird trotz der verringerten durchschnittlichen Antwortzeit die maximale Antwortzeit erhöht. In Bezug auf die Kosten wird zwar eine leichte Tendenz zu einer Erhöhung durch Elastizität gemessen, jedoch ist das Ergebnis nicht deutlich genug, um eine positive oder negative Auswirkung eindeutig festzustellen. Jedenfalls lässt sich aus den Ergebnissen ableiten, dass der positive Effekt von Elastizität auf die Performanz größer ist als die Veränderung der Kosten.

ABSTRACT

Due to the constantly growing online market, the demand for affordable webshop-systems is also increasing, so that web-shops not only remain in the online presence of large companies, but also find their way into the broad SME market. At the same time, with the rise of cloud platforms and the often associated “pay-per-use”-principle, completely new requirements have arisen in terms of the efficiency of the use of resources in software-systems in order to save costs and at the same time maintain the performance of a system. The elasticity system property makes it possible to meet these special requirements. For the implementation of elasticity in a multi-tenant webshop-system, provision in the cloud in connection with container-technologies and the use of a suitable cluster-management-system is suggested.

With this background, the aim of this thesis is to measure the effects of elasticity on performance and costs in a multi-tenant, container-based webshop-system. For this purpose, a method for measuring the performance and costs of such a system is presented.

In order to measure the effects, two independent test systems are set up while one only differs from the other in terms of its elastic properties. The two test systems are then exposed to the same workloads and the performance and costs are measured in each case.

By comparing the results, a notable improvement in performance can be recorded through elasticity. However, despite the reduced average response time, the maximum response time is increased. Comparing costs, although a slight tendency towards higher costs due to elasticity can be measured, the result is not clear enough determine a positive or negative effect. In any case, it can be derived from the results that the positive effect of elasticity on performance is greater than the effect on costs.

INHALTSVERZEICHNIS

1 EINLEITUNG

Der E-Commerce Sektor befindet sich, genauso wie Cloud-Plattformen, seit Jahren im Aufschwung. Daraus resultierend werden immer mehr moderne Software-Systeme, darunter auch Webshop-Systeme, auf Cloud-Plattformen bereitgestellt. Unter anderem ist ein Grund dafür das „Pay-per-Use“-Prinzip, nach dem die vermeintlich unbegrenzten Ressourcen der Cloud nach Bedarf erweiterbar sind und nutzungsbasiert verrechnet werden. Aufgrund dieser Abrechnung per Nutzung werden die Anforderungen an die Effizienz der Ressourcennutzung nach oben geschraubt. Ungenutzte Ressourcen sollen so weit als möglich vermieden werden, um Kosten zu sparen. Gleichzeitig werden aber die Anforderungen an Performanz und Skalierbarkeit nicht gemindert. Die Antwort auf dieses Dilemma ist Elastizität. Mit dieser Eigenschaft ist ein System in der Lage, Ressourcen bedarfsorientiert zu allokieren und wieder freizugeben, wenn sie nicht mehr benötigt werden. Da die Bereitstellung und Freigabe von Ressourcen in der Cloud relativ einfach zu bewerkstelligen ist, bietet sich Elastizität nicht nur an, um Kosten auf einer Cloud-Plattform einzusparen, sondern umgekehrt bieten sich Cloud-Plattformen auch für dessen Umsetzung an.

Eine weitere Technologie, welche es in den vergangenen Jahren zu erhöhter Popularität geschafft hat, ist Containerisierung. Sie ist eine schlanke Form der Virtualisierung und stellt eine leichtgewichtige Alternative zu virtuellen Maschinen dar. Infolge der hohen Nutzung von Virtualisierung in Cloud-Umgebungen haben Container-Technologien ebenso von dem Aufschwung profitiert.

Aufgrund des sich wandelnden Marktes und dadurch steigenden E-Commerce Anteiles, gehört ein Webshop im Online-Auftritt eines Unternehmens mittlerweile zum Standard und wird vom Konsumenten meist auch erwartet. Dadurch sind Webshops schon lange nicht mehr ein Vertriebsinstrument, welches großen Unternehmen und Konzernen vorbehalten ist, sondern auch im Bereich der KMU weite Verbreitung findet. Da Eigenentwicklungen oder Individuallösungen von Webshops für KMU großteils zu aufwändig oder zu teuer sind, werden Standard-Lösungen von Webshop-System Anbietern verwendet. Die einzelnen Webshops solcher Standard-Lösungen können als Mandanten in einem mandantenfähigen Webshop-System abgebildet werden.

In den beiden vorangegangenen Bachelorarbeiten wurde das Thema Skalierbarkeit in Verbindung mit Microservices und monolithischer Architektur behandelt. Elastizität kann als automatische Skalierbarkeit und somit als weiterer Entwicklungsschritt in diesem Bereich angesehen werden. Die vorliegende Arbeit stellt darauf aufbauend die Eigenschaften von Elastizität und deren mögliche Performanz- und Kostenvorteile in den Fokus.

Zielgruppe dieser Arbeit sind in erster Linie Betreiber von mandantenfähigen Webshop-Systemen, die über Elastizität Performanz- und Kostenvorteile erreichen wollen. Im weiteren Sinne werden auch Anbieter von branchenunabhängigen mandantenfähigen oder

containerbasierten Systemen angesprochen. In jedem Fall soll diese Arbeit als generelle Informationsquelle zur Anwendung von Elastizität in solchen Systemen dienen sowie mögliche Ansätze der Umsetzung aufzeigen. Des Weiteren wird untersucht, ob und welche Optimierungspotenziale bezüglich Performanz und Kosten in solchen Systemen über Elastizität erreicht werden können.

1.1 Zielsetzung und Forschungsfrage

Entsprechend dem einleitenden Hintergrund liegt der Fokus der Arbeit auf der Elastizität von Software-Systemen, insbesondere von mandantenfähigen Webshop-Systemen. Dazu werden die theoretischen Grundlagen für die behandelten Teilgebiete ausgearbeitet, um auf deren Basis Möglichkeiten und Ansätze für die Implementierung eines solchen Systems vorzuschlagen und zu diskutieren.

Aufbauend auf die theoriebasierten Zielsetzungen werden empirische Tests durchgeführt, anhand derer die Auswirkung von Elastizität auf die Performanz und Kosten eines mandantenfähigen, containerbasierten Webshop-Systems gemessen werden soll. Die daraus erhobenen Messwerte werden zu gesammelten Ergebnissen konsolidiert und sollen so eine Aussage über die Auswirkungen von Elastizität auf Performanz und Kosten ermöglichen. In diesem Kontext wird die folgende Forschungsfrage definiert.

> *Welche Auswirkungen hat Elastizität auf Performanz und Kosten in einem mandantenfähigen, containerbasierten Webshop-System?*

1.2 Methodik

Im ersten Schritt wird die theoretische Basis beleuchtet. Über Literaturrecherche werden Informationen zu den einzelnen Teilgebieten zusammengetragen. Im Detail bezieht sich dies zu Beginn auf die Definition von Elastizität. In diesem Zuge wird auch der Begriff Skalierbarkeit behandelt, auf dem Elastizität aufbaut. Im Folgenden werden die Systemeigenschaften des in dieser Arbeit behandelten Systemtyps näher beschrieben. Dementsprechend werden Mandantenfähigkeit und Container-Technologien, sowie deren Eigenschaften, Vorteile und Nachteile in Verbindung mit Elastizität erörtert. Anschließend werden die Grundlagen zu Cluster-Management-Systemen dargelegt, wobei Definition, Ziele und Anforderungen, sowie der Zusammenhang mit Elastizität ausgeführt wird.

Nach dem Aufbau des theoretischen Fundaments zu den einzelnen Themen werden die Rahmenbedingungen für die Durchführung der Tests festgelegt. Dazu gehören zunächst die verwendeten Metriken für Performanz und Kosten sowie die verwendeten Messmethoden. Beide basieren auf weiterer Literaturrecherche kombiniert mit gezielten Abwandlungen und Änderungen der in der Literatur beschriebenen Ansätze, um eine bessere Anwendbarkeit und Vergleichbarkeit mit den verwendeten Testsystemen zu erreichen.

Anschließend wird ein statisches Referenzsystem, sowie ein elastisches Vergleichssystem aufgebaut, welche für die Durchführung der Tests herangezogen werden. Dabei wird darauf geachtet, dass beide Systeme grundlegend gleich sind und sich das elastische gegenüber dem statischen nur aufgrund seiner elastischen Eigenschaften unterscheidet. Des Weiteren werden die für die Durchführung der Tests verwendeten Auslastungsmuster beschrieben sowie die zugehörigen Testpläne erstellt. Vor dem Beginn der Tests wird die folgende, mit der Forschungsfrage im Einklang stehende Hypothese aufgestellt, die mithilfe der Testergebnisse bestätigt oder widerlegt werden soll.

> *Die Performanz und Kosten eines mandantenfähigen, containerbasierten Webshop-Systems werden über Elastizität verbessert.*

Für die Abwicklung der Tests werden zwei vollkommen unabhängige Test-Umgebungen für das statische und elastische Testsystem bereitgestellt. Dieselben Testpläne werden in beiden Systemen ausgeführt und die vorab definierten Metriken erhoben. Auf Basis der gesammelten Messwerte werden die konsolidierten Ergebnisse berechnet und die Auswertungen vorgenommen, um das statische und elastische System zu vergleichen.

Aufgrund dieser Resultate wird abgeleitet, ob über Elastizität in einem mandantenfähigen, containerbasierten Webshop-System messbare Vorteile für Performanz und Kosten erzielt werden können. Mit diesen Erkenntnissen wird schlussendlich die Hypothese verifiziert sowie die Antwort auf die Forschungsfrage gegeben.

1.3 Aufbau

In Kapitel 2 werden die Grundlagen zu den in der Arbeit behandelten, übergreifenden Konzepten und Technologien beleuchtet. Dies betrifft in erster Linie die Themen Skalierbarkeit und Elastizität sowie in weiterer Folge Mandantenfähigkeit, Container-Technologien und Cluster-Management-Systeme. In jedem Teilgebiet werden die Zusammenhänge untereinander, und im Speziellen mit Elastizität, aufgegriffen.

Das Kapitel 3 beschäftigt sich mit der Vorbereitung der Tests und Festlegung aller Rahmenbedingungen. Zunächst werden die verwendeten Technologien genannt, bevor die Definition der Performanz- und Kosten-Metriken und die zugehörigen Messmethoden erläutert werden. In diesem Zuge wird eine alternative Methode zur Bestimmung der benötigten Ressourcen vorgestellt. Anschließend werden der Aufbau und die Eigenschaften der beiden Testsysteme sowie die durchzuführenden Testpläne beschrieben. Abschließend wird im Kapitel 3 die den Tests zugrundeliegende Hypothese aufgestellt und begründet.

Nach Durchführung der Tests werden in Kapitel 4 die Ergebnisse präsentiert. Dazu werden je Testlauf Liniendiagramme zum Verlauf der relevanten Metriken dargestellt. Im Zuge der Auswertung werden Vergleichswerte zwischen dem statischen und elastischen System aus den Messwerten abgeleitet, welche in weiterer Folge zu gesammelten Ergebniswerten konsolidiert werden. Anschließend wird das Kapitel 4 mit der Analyse der Ergebnisse sowie den daraus gewonnen Erkenntnissen abgeschlossen.

In Kapitel 5 werden die Analysen und Erkenntnisse aus Kapitel 4 zusammengefasst und bewertet. In diesem Zuge erfolgt die Verifizierung der Hypothese sowie die Beantwortung der Forschungsfrage. Abschließend werden die methodischen und technischen Limitierungen in der vorliegenden Arbeit aufgezeigt sowie ein vorsichtiger Ausblick für zukünftige Entwicklungen und deren Auswirkungen gewagt.

2 GRUNDLAGEN

Um ein gemeinsames Verständnis der behandelten Konzepte und Technologien zu schaffen, werden diese zunächst definiert und näher erläutert. Dazu wird in erster Linie Elastizität aufbauend auf Skalierbarkeit behandelt. In weiterer Folge wird auf die Bereiche der Mandantenfähigkeit, Container-Technologien und Cluster-Management-Systeme eingegangen. In diesen einzelnen Teilgebieten werden jeweils auch die Zusammenhänge mit Elastizität erörtert.

2.1 Skalierbarkeit

Seit dem Aufkommen der ersten Software-Systeme haben sich diese ständig weiterentwickelt und sind fortlaufend gewachsen. Dies ist unter anderem darauf zurückzuführen, dass sich die Ansprüche an die Systeme verändert haben, wie beispielsweise in Bezug auf gleichzeitige Benutzer sowie steigende Auslastungen. Um diesen Anforderungen gerecht zu werden, wurde Skalierbarkeit zu einem wichtigen Thema in der Informationstechnologie.

Auf die Frage der Begriffsbestimmung von Skalierbarkeit gehen die Antworten in der Literatur auseinander, sodass keine einheitliche, allgemeingültige Definition festgelegt werden kann. Vielmehr muss der Begriff Skalierbarkeit im jeweiligen Kontext und in Bezug auf die jeweils gegebenen Anforderungen abgestimmt werden. Demzufolge sind auch die Metriken, über die Skalierbarkeit definiert wird, vom Anwendungsfall abhängig. (Bondi, 2000; Lehrig et al., 2015) Umso wichtiger ist es, für den betreffenden Kontext die Auslegung von Skalierbarkeit abzustecken, um ein klares Verständnis sicherzustellen.

In der vorliegenden Arbeit wird auf die Definition von Skalierbarkeit nach Liu (2009) zurückgegriffen. Darin wird der enge Zusammenhang zwischen Performanz und Skalierbarkeit beschrieben. Performanz ist die Fähigkeit eines Software-Systems, eine Aufgabe schnell und effizient zu verarbeiten. In weiterer Folge wird Skalierbarkeit als Trend der Performanz bei steigender Auslastung definiert. Dementsprechend äußert sich die optimale Skalierbarkeit eines Systems in einer gleichbleibenden Performanz bei steigender Auslastung.

2.1.1 Vertikale und horizontale Skalierung

Unabhängig von der gewählten Definition kann zwischen vertikaler und horizontaler Skalierbarkeit unterschieden werden, wie in Abbildung 2-1 schematisch dargestellt. Vertikale Skalierbarkeit bezieht sich dabei auf eine Erweiterung der Ressourcen innerhalb eines Knotens. Das heißt, es werden innerhalb desselben physischen Gerätes oder derselben virtuellen Maschine mehr Ressourcen zur Verfügung gestellt. Meist geschieht dies in Form von zusätzlichen CPUs (Central Processing Unit), Laufwerken und Speicherkapazitäten (Bass et al., 2013). Implizit ergibt sich daraus, dass die vertikale Skalierung durch die physischen Grenzen des aktuellen Technologiestandes beschränkt ist.

Bei horizontaler Skalierung wird hingegen mit zusätzlichen logischen Knoten gearbeitet, die auf weiteren physischen oder virtuellen Maschinen bereitgestellt werden. Beispielsweise werden zusätzliche Applikationsserver, Webserver oder Datenbankserver eingerichtet, über welche die aufkommende Workload auf das System verteilt wird. Mit dieser Art der Skalierung sind höhere Potenziale gegeben, da über die mehrfache Bereitstellung von logisch getrennten Knoten in Summe höhere Kapazitäten erreicht werden können als auf einer einzelnen physischen Maschine. (Bass et al., 2013; Wolff, 2017)

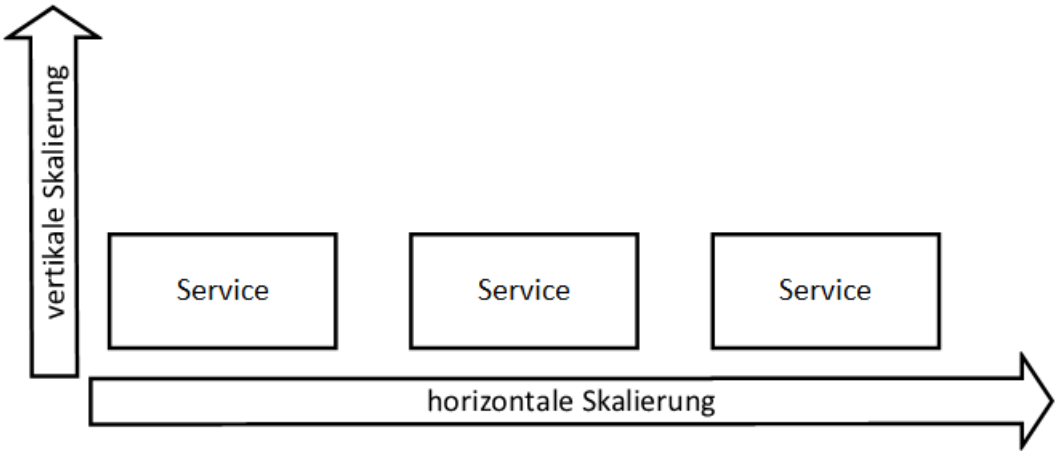


Abbildung 2-1 horizontale und vertikale Skalierung (in Anlehnung an Wolff, 2017)

Trotz des Vorteiles der höheren Potenziale von horizontaler Skalierung ist zu beachten, dass diese mit einem zusätzlichen Overhead erkauft werden muss. Ein Teil dieses Overheads ist das Betreiben von Load Balancing, um die Möglichkeit zu schaffen, die Last auf die verfügbaren Knoten zu verteilen. Neben der Auswahl einer geeigneten Verteilerlogik muss laufend die Verfügbarkeit der einzelnen Knoten geprüft werden. Weiters müssen die Aufgaben, Ressourcen und Aktivitäten der Knoten koordiniert und gesteuert werden. Dies wird meist von speziell dafür entwickelten Cluster-Management-Systemen übernommen.

2.1.2 Load-Balancing

Allgemein formuliert ist Load Balancing das gleichmäßige Verteilen von n Objekten auf z mögliche Ziele. Im technischen Kontext, und im speziellen angewendet auf verteilte Systeme, wird es als gleichmäßige Verteilung von t Aufgaben auf p Prozessoren formuliert. Das Ziel dabei ist die Minimierung der Ausführungszeit. (Marinescu, 2018)

Marinescu (2018) referenziert bei der Beschreibung des Load-Balancing Problems auf das „balls-and-bins“-Modell, in dem es darum geht, eine bestimmte Anzahl von Bällen optimal in eine vorgegebene Anzahl von Behältern zu verteilen. Im Rahmen dieser Arbeit wird nachfolgend Load-Balancing als die Verteilung von r Anfragen auf n Knoten beschrieben.

In der einfachsten Form werden die Anfragen nach reinem Zufallsprinzip auf die Knoten verteilt. Die maximale Auslastung $\max(L_i)$, gegeben $1 \leq i \leq r$, kann somit mit hoher Wahrscheinlichkeit wie folgt definiert werden. Die Kommunikation zwischen den Systemkomponenten ist in dieser Formel nicht berücksichtigt. (Azar et al., 1994; Marinescu, 2018)

$$\max(L_i) \approx \frac{\log r}{\log \log r}$$

Dieser Ansatz ist erweiterbar, indem die Anfragen sequenziell verteilt werden und für jede eine Vorauswahl von zwei zufälligen Knoten getroffen wird. Die Anfrage wird dann jenem Knoten zugewiesen, der aktuell eine geringere Auslastung hat. Dabei ergibt sich für $\max(L_i{}^2)$, gegeben $1 \leq i \leq r$, folgende Formel. . (Azar et al., 1994; Marinescu, 2018)

$$\max(L_i^2) \approx \frac{\log \log r}{\log 2} + O(1)$$

Noch bessere Ergebnisse werden bei einer Vorauswahl von $d > 2$ Knoten erreicht. In dieser Form wird das Vorgehen auch als Greedy-Algorithmus bezeichnet. (Azar et al., 1994; Marinescu, 2018)

$$\max(L_i^d) \approx \frac{\log \log r}{\log d} + O(1)$$

Horizontale Skalierbarkeit und Load-Balancing sind für verteilte Systeme in der praktischen Anwendung untrennbar. Wie in Kapitel 2.1.1 erwähnt, muss horizontale Skalierbarkeit mit zusätzlichem Overhead erkauft werden. Ein Teil dieses Overheads ist im Load-Balancing enthalten, welches zwar die Verteilung der Workload ermöglicht, jedoch nur unter dem Gesichtspunkt von mehr Koordination und zusätzlicher Kommunikation. Je besser die Lastverteilung ist, umso kleiner wird die maximale Last auf einem Knoten, wodurch sich die Verarbeitungszeit insgesamt verringert. Da Kommunikation im Vergleich zu normalen Rechenoperationen teuer ist ergibt sich ein Kompromiss zwischen optimaler Lastverteilung und zusätzlichem Kommunikationsaufwand. (Marinescu, 2018)

2.2 Elastizität

Wie für Skalierbarkeit gibt es auch für Elastizität in der Literatur keine allgemeingültige Definition. Viele bestehende Begriffsbestimmungen von Elastizität sind unspezifisch und inkonsistent. Dies macht es unter anderem schwierig, Elastizität von den verwandten Begriffen Effizienz und Skalierbarkeit abzugrenzen. (Herbst et al., 2013)

In der ursprünglichen Definition von Elastizität in der Physik handelt es sich um die Eigenschaft eines Materials, nach einer Deformation zu seiner ursprünglichen Form zurückzukehren (Chiang & Wainwright, 2005). Im Kontext von Informationstechnologie kann diese Eigenschaft eines Materials sinngemäß auf das Verhalten eines Software-Systems in einer Cloud-Umgebung umgelegt werden. Wie ein elastisches Material eine ursprüngliche Form hat, besitzt auch das elastische Software-System einen Ausgangszustand. Im Rahmen der vorliegenden Arbeit wird dieser über die standardmäßig zur Verfügung stehenden Ressourcen nach dem Start eines Systems definiert. Die Deformation aus dem physikalischen Kontext wird über Krafteinwirkung auf das Material hervorgerufen. Im Software-System bezieht sich diese auf die Workload. Bei steigender Belastung passt sich ein elastisches Software-System an diese Gegebenheiten an, indem zusätzliche Ressourcen zur Verfügung gestellt werden, um die Abarbeitung der

aufkommenden Workload zu bewerkstelligen. Je höher diese steigt, umso mehr Ressourcen werden allokiert. Diese Bereitstellung zusätzlicher Ressourcen im Software-System bezieht sich auf die Deformation des Materials im physikalischen Kontext. Bei sinkender Auslastung hingegen verhält sich das Software-System in umgekehrter Weise und gibt Ressourcen wieder frei. So wie ein elastisches Material bei schwindender Krafteinwirkung wieder in seine ursprüngliche Form zurückfindet, begibt sich auch ein elastisches Software-System bei fallender Auslastung in seinen Ausgangszustand zurück.

2.2.1 Quantifizierung

In der Literatur gibt es nicht nur verschiedene Definitionen, sondern auch verschiedene Bezeichnungen für Elastizität. So werden teilweise die Begriffe dynamische Skalierung (Dragoni et al., 2017), Elastic Scale-Out (Minhas et al., 2012) oder Autoscaling (Chen & Bahsoon, 2018) verwendet. Die Grundprinzipien bleiben bei den verschiedenen Auslegungen weitgehend gleich, jedoch unterscheiden sich die Ansätze in den maßgebenden Metriken sowie den Methoden zur Messung und Quantifizierung. Ebenso wird an vielen Stellen in der Literatur der enge Zusammenhang mit Cloud-Computing beschrieben, bei dem Elastizität als einer der Schlüsselvorteile gilt. (Herbst et al., 2013; Kuperberg et al., 2011; Mell & Grance, 2011)

Kuperberg et al. (2011) behandeln die Definition und Quantifizierung der Elastizität von Ressourcen in Cloud-Umgebungen. Im Allgemeinen beschreiben sie Elastizität als die Eigenschaft, die für Anwendungen bereitgestellten Ressourcen automatisch, dynamisch, flexibel und wiederholend anzupassen. Als Besonderheit wird explizit erwähnt, dass diese Adaptierungen während der Laufzeit erfolgen. Zudem wird auch auf die wirtschaftlichen Aspekte hingewiesen, nach denen Elastizität die Produktivität und Ressourcennutzung verbessern und damit, unter Einhaltung von SLA (Service Level Agreement) und QoS (Quality of Service), Kosten einsparen kann. Diese Definition steht auch im Einklang mit dem von Marinescu (2018) genannten Ziel der Elastizität, nach dem mit den automatischen Ressourcenanpassungen optimale Kosten und Antwortzeiten erreicht werden sollen.

Wie auch in Kapitel 2.2.5 beschrieben, gilt Elastizität als eine auf Skalierbarkeit basierende Eigenschaft. Dementsprechend sind die Metriken von Skalierbarkeit auch die Basis derer für Elastizität. Allerdings ist dabei nicht nur wichtig, wie gut sich ein System unter Verwendung zusätzlicher Ressourcen an steigende Auslastungen anpassen kann, sondern auch, wie schnell und präzise das bewerkstelligt werden kann. Dieser zusätzliche zeitliche Aspekt erweitert die herkömmlichen Metriken für Skalierbarkeit und bildet damit die Grundlage für die Messung von Elastizität. (Kuperberg et al., 2011)

In Kapitel 2.2.3 werden unter anderem die beiden Kernaspekte Reaktionszeit und Präzision behandelt, nach denen Elastizität bewertet werden kann. Mit der aufbauenden, detaillierteren Definition von Kuperberg et al. (2011) werden diese Gesichtspunkte genauer beschrieben. Demnach ist eine Metrik für Elastizität von mehreren Faktoren abhängig. Diese werden für die Verwendung in dieser Arbeit teilweise erweitert und angepasst und nachstehend beschrieben.

Workload: Die Workload ergibt sich aus der Summe der von einem System zu bewältigenden Aufgaben, welche verschiedene Formen annehmen können. In dieser Arbeit bezieht sich die Workload auf die Anzahl eingehender Anfragen auf ein System.

Auslastung: Die Auslastung beschreibt, wie stark ein System oder eine Systemkomponente (z.B.: eine virtuelle Maschine (VM)) ausgelastet ist. In der Regel steigt die Auslastung zusammen mit der Workload.

Performanz: Performanz beschreibt, wie schnell und effizient ein Software-System eine Aufgabe bewältigen kann (Liu, 2009). Darauf basierend muss für die Performanz eines Systems eine konkrete Metrik definiert werden, nach der sie gemessen wird. Im Rahmen dieser Arbeit ist die dafür verwendete Metrik die Antwortzeit eines Systems.

Antwortzeit: Diese beschreibt die Zeitdauer in Millisekunden, welche zwischen dem Absenden einer Nachricht von einem Client an ein Software-System und dem Erhalten einer Antwort verstreicht.

Elastizitätsdimension: Dies ist eine festgelegte Dimension, in welche ein elastisches System skalieren kann. Einer Elastizitätsdimension werden zumindest ein oder mehrere Ressource-Typen zugewiesen. (Herbst et al., 2013)

Ressource-Typ: Ressource-Typen konkretisieren die Elastizitätsdimension und beschreiben, welche Ressourcen eines Systems skalieren können, beispielsweise CPU und RAM (Random Access Memory). (Herbst et al., 2013)

Skalierungseinheit: Die Skalierungseinheit gibt die kleinstmöglichen Anpassungsschritte zu einer Elastizitätsdimension an. In der Regel sind das diskrete Werte, welche sich beispielsweise auf die Anzahl von CPUs oder virtuellen Maschinen beziehen. (Herbst et al., 2013)

Entscheidungsprozess: Der Entscheidungsprozess legt fest, unter welchen Bedingungen zusätzliche Ressourcen bereitgestellt oder überflüssige Ressourcen freigegeben werden sollen. (Herbst et al., 2013)

Trigger-Zeitpunkt: Dieser beschreibt einen Zeitpunkt, zu dem ein Entscheidungsprozess eine Anpassung der verfügbaren Ressourcen auslöst. Dieser Zeitpunkt leitet die Rekonfigurationsphase ein. (Kuperberg et al., 2011)

Rekonfigurationsphase: Innerhalb dieser Phase erfolgt die Bereitstellung oder Freigabe der Ressourcen. Die Dauer dieser Phase wird als Rekonfigurationsdauer bezeichnet. (Kuperberg et al., 2011)

Rekonfigurationszeitpunkt: Das Ende der Rekonfigurationsphase stellt den Rekonfigurationszeitpunkt dar. Das heißt, ab diesem Zeitpunkt sind die Ressourcen für die Anwendung bei einer Bereitstellung verfügbar oder bei einer Freigabe vollständig entfernt. (Kuperberg et al., 2011)

Effekt der Rekonfiguration: Dieser beschreibt die Auswirkung der Rekonfiguration auf die Performanz des Systems, das heißt im Rahmen dieser Arbeit den Effekt auf die Antwortzeit (Kuperberg et al., 2011). Wie groß der Effekt ist, hängt unter anderem von der Elastizitätsdimension und von der Skalierungseinheit ab.

Reaktionszeit: Reaktionszeit bezieht sich auf die Zeit, die ein System braucht, um von einem unterversorgten in einen optimalen oder überversorgten Zustand, oder von einem überversorgten in einen optimalen oder unterversorgten Zustand zu gelangen. (Herbst et al., 2013)

Präzision: Präzision steht für die absolute Abweichung der aktuell bereitgestellten Ressourcen von den aktuell tatsächlich benötigten Ressourcen. (Herbst et al., 2013)

Basierend auf diesen Begriffen quantifizieren Herbst et al. (2013) Elastizität aufgrund von Präzision und Reaktionszeit. Eine andere Herangehensweise ist die Kombination des Effekts der Rekonfiguration, der Häufigkeit der Konfigurationspunkte und der Reaktionszeit. Die Effekte der Rekonfiguration sind abhängig von den Skalierungseinheiten, wodurch wiederum die Granularität der Skalierung festgelegt wird. (Kuperberg et al., 2011)

Bildet man ein System mit fest definierten Ressourcen ohne jegliche Skalierbarkeit oder Elastizität ab, so steigt die Antwortzeit in diesem System zusammen mit der Workload. Steigt die Auslastung des Systems in einen Zustand der Überlastung, so wächst die Antwortzeit überproportional und verzeichnet damit eine sinkende Performanz. Das Verhalten zwischen Workload und Antwortzeit in so einem Fall ist in Abbildung 2-2 zu sehen.

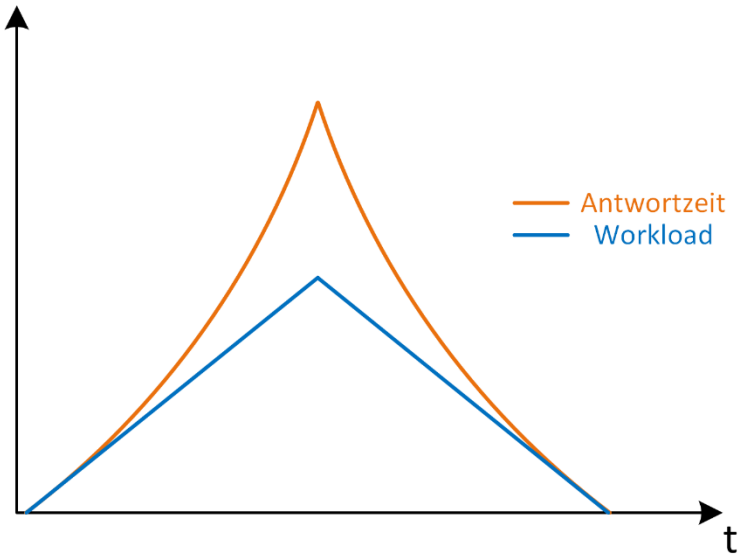


Abbildung 2-2 Zusammenhang zwischen Workload und Antwortzeit in einem System ohne Skalierbarkeit (in Anlehnung an Kuperberg et al., 2011)

Bleibt hingegen die Workload gleich und ändert man nur die zur Verfügung stehenden Ressourcen in einem System, verhält sich die Antwortzeit in umgekehrter Weise, sodass mehr Ressourcen zu kürzeren und weniger Ressourcen zu längeren Antwortzeiten führen. Dieser Zusammenhang ist in Abbildung 2-3 dargestellt.

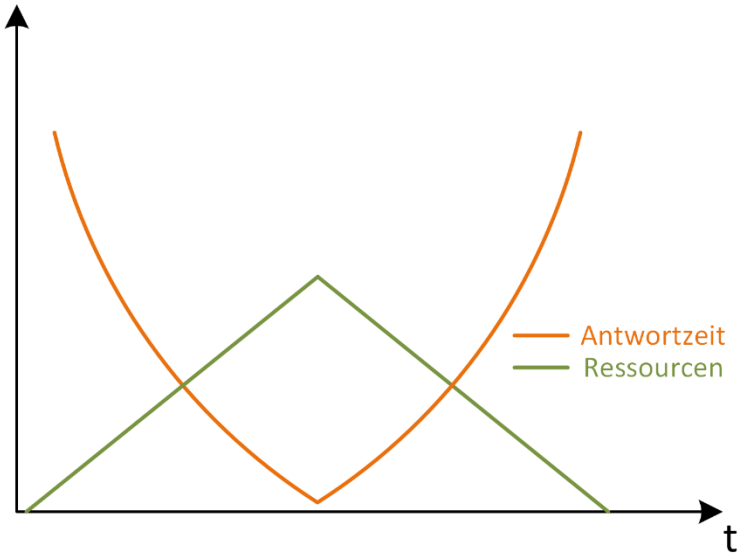


Abbildung 2-3 Zusammenhang zwischen Antwortzeit und Ressourcen in einem System bei gleichbleibender Workload (in Anlehnung an Kuperberg et al., 2011)

In einem elastischen System erfolgt eine laufende Überprüfung und Rekonfiguration der zur Verfügung stehenden Ressourcen, wodurch eine niedrig bleibende Antwortzeit sichergestellt werden soll. Zusätzliche Ressourcen werden dann zur Verfügung gestellt, wenn beispielsweise die Auslastung des Systems oder die Antwortzeit über einen definierten Schwellwert steigen. In so einem Fall wird die Antwortzeit im System bis zum Rekonfigurationszeitpunkt weiter steigen. Danach hat das System mehr Ressourcen zur Verfügung und kann somit die Anfragen abhängig vom Effekt der Rekonfiguration wieder in kürzerer Zeit abarbeiten. Bei laufend steigender Workload wiederholt sich dieser Vorgang wie in Abbildung 2-4 gezeigt. Dabei ist erkennbar, dass die Präzision der Elastizität unter anderem von der Granularität der Skalierungseinheiten abhängt.

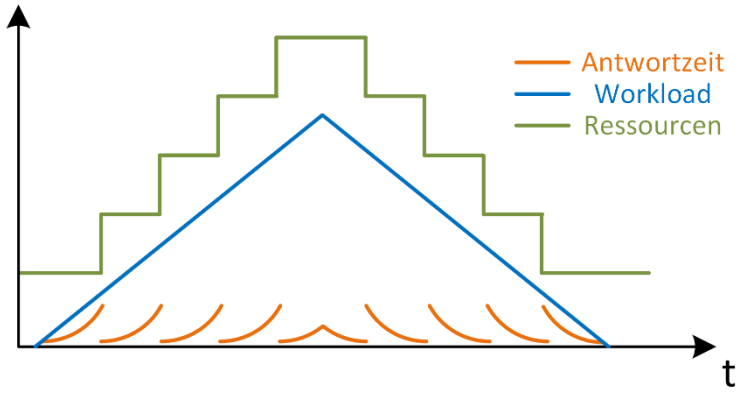


Abbildung 2-4 Zusammenhang zwischen Antwortzeit, Workload und Ressourcen in einem elastischen System (in Anlehnung an Kuperberg et al., 2011)

2.2.2 Entscheidungsprozesse und Dimensionen

Herbst et al. (2013) zeigen in ihrer Arbeit Inkonsistenzen und verschiedene Auslegungen von Elastizität auf. Nach einer Aufstellung und Konsolidierung der zentralen Aspekte wird eine darauf basierende Definition festgelegt. Nach dieser ist Elastizität der Grad, zu dem ein System in der Lage ist, sich über automatische Bereitstellung oder Freigabe von Ressourcen an verändernde Auslastungen anzupassen, sodass die bereitgestellten Ressourcen zu jedem Zeitpunkt möglichst präzise den geforderten Ressourcen entsprechen. Diese Beschreibung ist allgemein gehalten und bezieht sich nur auf die Grundbegriffe von Ressourcen und Auslastung. Es wird auf keine spezifischen Eigenschaften des Systems oder auf bestimmte Architekturen eingegangen.

Die automatische Bereitstellung und Freigabe der Ressourcen erfolgt mithilfe zuvor festgelegter Entscheidungsprozesse, in denen Bedingungen für die Skalierung definiert werden. Für ein elastisches System gibt es zumindest einen, möglicherweise mehrere davon. (Herbst et al., 2013) Da in diesen Entscheidungsprozessen die Skalierungsbedingungen und damit die maßgebenden Metriken deklariert werden, haben sie gemeinsam mit den Quantifizierungsmethoden entsprechenden Einfluss auf das Elastizitätsverhalten des Systems.

Des Weiteren werden jedem Entscheidungsprozess eine oder mehrere Elastizitätsdimensionen zugewiesen, welche die Dimensionen des Skalierungsprozesses bestimmen. Jede dieser Dimensionen kann wiederum einen oder mehrere Ressource-Typen beinhalten. Beispiele dafür sind CPU- oder RAM-Kapazitäten. Werden mehrere Ressource-Typen nur gemeinsam bereitgestellt oder freigegeben, so sind diese in einem Ressource-Container, wie beispielsweise einer virtuellen Maschine, zusammengefasst. (Herbst et al., 2013)

2.2.3 Optimale Elastizität

Optimale Elastizität beschreibt den Fall, dass ein System in Bezug auf alle Elastizitätsdimensionen ohne Limitierungen skalierbar ist und Ressourcen unmittelbar in genau der benötigten Menge bereitstellt oder freigibt, um jederzeit exakt die aktuellen Anforderungen zu erfüllen. (Herbst et al., 2013) Dieses Verhalten ist dann erreichbar, wenn die beiden Kernaspekte Reaktionszeit und Skalierungseinheit einen sich an null annähernden Wert annehmen. Im Umkehrschluss folgt daraus, dass optimale Skalierbarkeit durch die Dauer der Reaktionszeiten und Granularität der Skalierungseinheiten eingeschränkt wird. Je kürzer die Reaktionszeiten und je kleiner die Skalierungseinheiten, umso näher wird optimale Elastizität erreicht.

In einem System mit optimaler Elastizität würde sich die Antwortzeit, unabhängig von der Workload, nicht erhöhen und dauerhaft auf einem Minimum verbleiben, wie in Abbildung 2-5 dargestellt. Diese Beschreibung der optimalen Elastizität deckt sich auch mit der Beschreibung von optimaler Skalierbarkeit nach Liu (2009), wonach optimale Skalierbarkeit gegeben ist, wenn die Performanz eines Systems bei steigender Last eine flache Kurve zeichnet. Da dies in der Praxis nicht oder nur mit unverhältnismäßigem Aufwand umsetzbar ist, bleibt optimale Elastizität ein unrealistisches, theoretisches Modell. (Kuperberg et al., 2011)

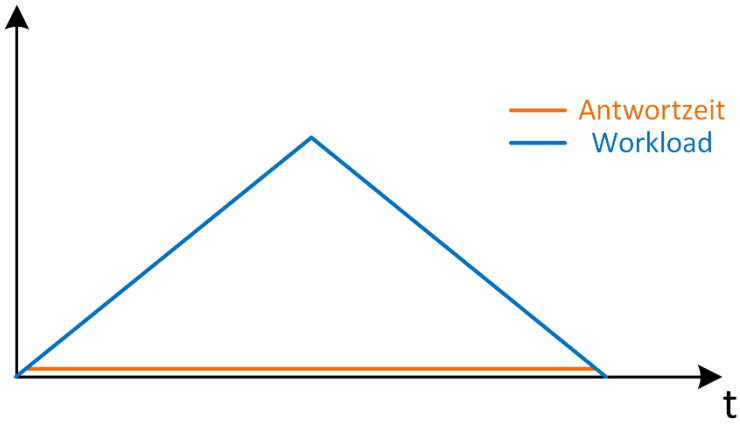


Abbildung 2-5 Zusammenhang zwischen Antwortzeit und Workload bei optimaler Elastizität (in Anlehnung an Kuperberg et al., 2011)

Wie in Kapitel 2.2.1 beschrieben, ist eine weitere Möglichkeit zur Quantifizierung von Elastizität die Kombination aus dem Effekt und der Häufigkeit der Rekonfiguration sowie der Reaktionszeit. Basierend auf diesen drei Aspekten wird optimale Skalierbarkeit mit ständigen Rekonfigurationen, kurzen Reaktionszeiten und kleinen Effekten beschrieben (Kuperberg et al., 2011). Diese Auslegung von optimaler Elastizität deckt sich sinngemäß auch mit jener von Herbst et al. (2013), allerdings wird Präzision in die Häufigkeit und die Effekte der Rekonfigurationen heruntergebrochen.

Bei diesen Überlegungen muss berücksichtigt werden, dass es nicht unbedingt sinnvoll ist die Skalierungseinheiten beliebig klein zu definieren. Zum einen ist dies aus technischer Sicht nicht immer möglich, da die meisten Elastizitätsdimensionen nur als diskrete Einheiten skaliert werden können (Kuperberg et al., 2011). Dies hat beispielsweise bei der Skalierung nach virtuellen Maschinen eine klare Auswirkung, da diese im abgeschlossenen Skalierungsprozess nur ganz oder gar nicht bereitgestellt werden.

Des Weiteren muss bei der Wahl der Skalierungseinheiten sowie dem Intervall der Entscheidungsprozesse berücksichtigt werden, dass die Prozesse für die Entscheidungen sowie der Bereitstellung und Freigabe von Ressourcen selbst auch Aufwand verursachen. Wären die Skalierungseinheiten beliebig klein gewählt, müssten auch die Entscheidungsprozesse in entsprechend kurzen Intervallen ausgeführt werden, um die gewünschte Spontanität der Ressourcenanpassung in den definierten kleinen Einheiten zu erreichen. Diese kontinuierlichen Entscheidungsprozesse und Ressourcenanpassungen würden zu einem großen Overhead führen. Deshalb müssen die Skalierungseinheiten und -intervalle an die Elastizitätsdimensionen angepasst und in sinnvollen Größen gewählt werden.

Neben der Einschränkung der optimalen Elastizität aufgrund der Skalierungseinheiten und -intervalle werden weitere Limitierungen infolge der Zeitverzögerung in der Anpassung der Ressourcen verursacht. Zum einen ergeben sich diese angesichts des Entscheidungsprozesses, welcher die Berechnung zur Ressourcenänderung erst auf Basis der gemessenen Überwachungsdaten treffen kann. Anschließend kommt es zu einer weiteren Verzögerung durch die technische Bereitstellungs- oder Freigabedauer der Ressourcen. Diese kann abhängig von der Elastizitätsdimension und der Skalierungseinheit wiederum sehr unterschiedlich sein.

2.2.4 Elastizität und Zustände

In der Arbeit von Minhas et al. (2012) wird explizit auf die Elastizität von Datenbanken eingegangen, allerdings wird dabei der Begriff Elastic Scale-Out verwendet. In diesem Zuge wird Elastic Scale-Out als die Fähigkeit bezeichnet, Rechenkapazitäten bei unterschiedlicher Last nach Bedarf zu erhöhen oder zu verringern. Wesentlicher Aspekt bei der Beschreibung von Elastizität nach Minhas et al. (2012) ist die Unterscheidung in zustandsbehaftete und zustandslose Systeme, wobei zustandslose Systeme vergleichsweise einfach skaliert und auch elastisch aufgebaut werden können.

Mithilfe von Zustandslosigkeit ist es möglich, die Knoten in einem Cluster vollkommen unabhängig voneinander hinzuzufügen oder wegzunehmen. Somit müssen neu hinzugefügte Knoten mit keinem Zustand angereichert werden, um Anfragen bearbeiten zu können. Das heißt, dass keine Daten auf einen neuen Knoten kopiert werden müssen. Ein zusätzlicher Knoten kann somit direkt nach dessen Start Anfragen bearbeiten und erhöht unmittelbar die Gesamtkapazität des Systems. Dasselbe gilt auch beim Entfernen eines Knotens aus dem Cluster. Es müssen keine Daten von einem abzubauenden Knoten auf andere kopiert werden und die freigegebenen Kapazitäten reduzieren direkt die Gesamtkapazität des Systems.

Zustandslosigkeit und die damit einhergehende hohe Unabhängigkeit zwischen Knoten ist eine Eigenschaft, welche in elastischen Systemen anzustreben ist. Da der Zweck einer Datenbank allerdings die Speicherung von Zuständen ist, ist Zustandslosigkeit hier im engeren Sinn ein Widerspruch. Dementsprechend ist nach Minhas et al. (2012) die Skalierung, und im speziellen die elastische Skalierung, von Datenbank-Systemen schwieriger als für andere, zustandslose Systeme.

Ein Versuch, diesen Nachteil der Zustandsbehaftung in modernen Datenbank-Systemen zu umgehen, ist das BASE Modell als Gegenstück zum ACID Modell. Ersteres ermöglicht durch seine gelockerten Prinzipien eine höhere Unabhängigkeit zwischen einzelnen Knoten und Datenbank-Partitionen. Dadurch erhöht sich neben der Performanz auch die Skalierbarkeit, sodass in weiterer Folge Elastizität leichter umsetzbar ist. Diese Vorteile müssen im BASE Modell allerdings durch Abschläge in der Datenkonsistenz erkauft werden.

Auffallend in der Arbeit von Minhas et al. (2012) in Zusammenhang mit der Definition von Elastic Scale-Out ist, dass der Inhalt zwar auf die Elastizität von Datenbanken abzielt, jedoch explizit nur Rechenleistung als entscheidende Metrik genannt wird. Eine bedarfsorientierte Anpassung von I/O- und Speicherkapazitäten, sowohl im Sinne von Festplatte als auch Hauptspeicher, wird nicht berücksichtigt. Begründet ist dies vermutlich durch die Tatsache, dass in der besagten Arbeit die Erstellung eines elastischen Datenbanksystems mithilfe von VoltDB behandelt wird und diese eine in-memory Datenbank ist. Des Weiteren wird Rechenleistung nicht im Sinne von CPU deklariert, sondern als die Anzahl der aktiven Knoten im System, das heißt, zusätzliche Rechenleistung bezieht sich auf die Bereitstellung weiterer Knoten. Eine explizite Spezifikation dieser Knoten in Bezug auf CPU, RAM oder Festplattenspeicher wird nicht angegeben. Hinsichtlich dieser Limitierungen bieten sich die Inhalte der Arbeit von Minhas et al. (2012) an, um in zukünftigen Arbeiten aufgegriffen und weiterverfolgt zu werden. Mögliche Anhaltspunkte

können dabei die Messung von Elastizität auf Basis von speicherbezogenen Metriken sowie ein Vergleich zwischen relationalen und NoSQL-Datenbanken sein.

2.2.5 Abgrenzung zu Skalierbarkeit

In Kapitel 2.1 wird die Definition von Skalierbarkeit nach Liu (2009) beschrieben. Generell stehen die beiden Begriffe Skalierbarkeit und Elastizität in engem Zusammenhang. Skalierbarkeit kann als Überbegriff und als Grundlage für Elastizität gesehen werden. Herbst et al. (2013) nennen die Skalierbarkeit von Hardware, Virtualisierung und Software als Voraussetzung, um über Elastizität sprechen zu können. Im Umkehrschluss ist Elastizität eine auf Skalierbarkeit aufbauende und erweiternde Eigenschaft.

Minhas et al. (2012) bestätigen dies auch in ihrer Arbeit. Dabei wird auf die Komplexität von Elastizität von Datenbanken hingewiesen. Als Herangehensweise wird empfohlen, vorerst ein skalierbares Datenbank-System aufzubauen, welches eine feste Anzahl von Knoten hat. In so einem skalierbaren System wird die Workload vorerst nur über Replikation oder Partitionierung auf die bestehenden Knoten aufgeteilt. Ausgehend davon kommen in weiteren Schritten erst elastische Eigenschaften hinzu.

Eine klare Unterscheidung zwischen Skalierbarkeit und Elastizität kann aufgrund der Variabilität der Ressourcennutzung zur Laufzeit getroffen werden. Skalierbarkeit beschreibt, wie sich die Performanz eines Systems bei steigender Auslastung entwickelt (Liu, 2009). Allerdings wird dabei davon ausgegangen, dass die zur Verfügung stehenden Ressourcen fest definiert sind und nicht auslastungsabhängig angepasst werden. Dementsprechend bezieht sich die Skalierbarkeit eines Systems immer auf eine bestimmte Anzahl von zur Verfügung stehenden Ressourcen. Wird in weiterer Folge die Skalierbarkeit bei unterschiedlichen Ausbaustufen gemessen, kann abgeleitet werden, wie gut ein System zusätzliche Ressourcen nutzen kann, um die Performanz zu erhalten. (Winkler, 2019)

Im Gegensatz dazu bezieht sich Elastizität auf die Variabilität der Ressourcen und wie gut diese zur Laufzeit angepasst werden können. Folglich beinhaltet Elastizität sowohl den zeitlichen Aspekt, wie schnell und wie oft die Anpassung der Ressourcen erfolgt, als auch wie granular (Herbst et al., 2013). Somit geht Elastizität nicht wie Skalierbarkeit auf eine fest definierte Ressourcenanzahl zurück, sondern darauf, wie präzise der Ressourcenbedarf des Systems durch Anpassungen zur Laufzeit zu einem bestimmten Zeitpunkt oder über einen bestimmten Zeitraum gedeckt wird.

2.2.6 Voraussetzungen und Begünstigungen für Elastizität

Einige Systemeigenschaften werden für die Umsetzung von Elastizität vorausgesetzt, andere wiederum sind zwar nicht erforderlich, begünstigen jedoch das elastische Verhalten. Wie in Kapitel 2.2.5 beschrieben, ist Skalierbarkeit eine Voraussetzung für Elastizität und in gleicher Weise hat Skalierbarkeit ebenso obligatorische sowie begünstigende Eigenschaften. Merkmale,

welche die Skalierbarkeit eines Systems verbessern, verbessern in weiterer Folge auch die Elastizität.

Eine der zentralen Eigenschaften, welche Skalierbarkeit ermöglicht und begünstigt, ist Unabhängigkeit zwischen den zu skalierenden Komponenten und deren Replikaten. Über die Unabhängigkeit einzelner Teile eines Systems ergibt sich die Möglichkeit, diese auf getrennten virtuellen oder physischen Maschinen bereitzustellen. Dies beginnt bereits bei einer Gliederung des Systems in eine 3-Schichten-Architektur, in der beispielsweise die Applikation und die Datenbank auf getrennte Server aufgeteilt werden. Je weiter das System in kleinere, unabhängige Komponenten heruntergebrochen wird, umso höher ist die Flexibilität für die verteilte Bereitstellung. Aufgrund dieser Verteilung wird für Skalierbarkeit und Elastizität implizit ein verteiltes System vorausgesetzt (Marinescu, 2018).

Dieser Ansatz spiegelt sich auch in der populär gewordenen Microservice-Architektur wider. Allerdings ist das Herunterbrechen eines Systems in beliebig kleine Komponenten nur bis zu einem gewissen Grad sinnvoll, weil dadurch Abschläge in der Performanz und höhere Komplexität in Kauf genommen werden müssen. (Fowler, 2015; Namiot & Sneps-Sneppe, 2014; Taibi et al., 2017; Wolff, 2017)

Unabhängigkeit zwischen Komponenten ist ein abstrakter Begriff, welcher über verschiedene Konzepte konkretisiert und umgesetzt werden muss. Ein Konzept, um diese Unabhängigkeit zu erreichen, ist lose Koppelung. Dadurch sind die einzelnen Systemelemente untereinander austauschbar, sodass Änderungen eines Elements keine Anpassung anderer erfordern. Vergleichsweise weist ein monolithisches System, welches als einzige, zusammenhängende Einheit ausgeführt wird, keinerlei horizontale Skalierbarkeit auf. Sämtliche Komponenten in solch einem System sind voneinander abhängig und unterliegen einer engen Koppelung. Je weiter es gelingt, einzelne Komponenten des Systems voneinander zu entkoppeln, umso besser ist horizontale Skalierung möglich.

Nach der Definition in Kapitel 2.1.1 wird horizontale Skalierung über die Replikation von logischen Knoten erreicht. Dementsprechend müssen als dessen Grundlage einzelne Komponenten repliziert und diese Replikate parallel ausgeführt werden können. Dieser Ansatz kann über das Konzept der Zustandslosigkeit umgesetzt werden, welche erreicht wird, indem eine Komponente in sich keinen Zustand speichert, sondern diesen nur aus einer eingehenden Nachricht oder einer persistierten Datenschicht liest. Eine zustandslose Komponente kann beliebig oft repliziert werden, während jedes Replikat in der Lage ist, eine Aufgabe unabhängig und parallel zu anderen auszuführen. Im Unterschied dazu sind zustandsbehaftete Komponenten schwieriger zu skalieren, da stets auf die Konsistenz der Daten über alle Replikate geachtet werden muss (Minhas et al., 2012). Des Weiteren hat die Zustandslosigkeit von Komponenten positive Auswirkungen auf die Performanz des Systems während der Skalierung. Werden Daten innerhalb einer Komponente gespeichert, so müssen diese beim Starten einer neuen Instanz geladen werden. Umgekehrt müssen Daten einer freizugebenen Instanz auf andere übertragen werden, um Datenverlust zu vermeiden. Der Aufwand für diese Datenmigrationen während der Skalierung wird durch Zustandslosigkeit eliminiert, wodurch eine schnellere Bereitstellung und Freigabe von Replikaten möglich ist. (Laux et al., 2014)

Als weitere Voraussetzung für Elastizität gilt zumindest ein Anpassungsprozess, welcher die verfügbaren Ressourcen des Systems entsprechend der aktuellen Auslastung anpasst. Elastizität bezieht sich grundlegend auf die bedarfsorientierte Ressourcenanpassung zur Laufzeit, wobei diese im engeren Sinne automatisch passiert, im weiteren Sinne aber auch manuelle Schritte beinhalten kann. Ohne solch einen Anpassungsprozess, ist ein skalierbares System nicht in der Lage, sich elastisch zu verhalten, da Skalierbarkeit allein keinen zeitlichen Aspekt beinhaltet. (Herbst et al., 2013)

2.2.7 Messung von Elastizität

Da Elastizität, wie in Kapitel 2.2.5 beschrieben, eine auf Skalierbarkeit aufbauende Eigenschaft ist, leiten sich die Metriken auch von jenen der Skalierbarkeit ab. Generell gibt es in der Literatur verschiedene Ansätze zur Messung, jedoch sind die ausschlaggebenden Aspekte in vielen Fällen auf Auslastung und Performanz zurückzuführen, wie dies auch von Liu (2009) beschrieben wird. Bereits in der vorangegangenen Arbeit wurde eine Metrik für Skalierbarkeit auf Basis der Performanz hinsichtlich Antwortzeit und Workload in Form von gleichzeitigen Benutzern definiert (Winkler, 2019).

Für die Messung von Elastizität werden diese Ansätze um zwei wesentliche Faktoren erweitert. Einer davon ist, dass nicht nur die Performanz in einem System berücksichtigt wird, sondern ebenso die Kosten. Denn sofern ein System in der Lage ist, zusätzlich bereitgestellte Ressourcen zu nutzen, dann kann eine gute Performanz durch beliebig hohe Ressourcenerweiterungen in jedem Fall erhalten werden. Allerdings würde so ein Vorgehen hohe Ressourcenkosten mit sich bringen und ist somit nicht wirtschaftlich.

Hier setzt Elastizität an und berücksichtigt ebenso die Ressourcen, welche bereitgestellt sind, aber nicht benötigt werden. So wird einerseits die Performanz des Systems gemessen, um eine mögliche Unterversorgung festzustellen und andererseits werden die bereitgestellten Ressourcen mit den benötigten verglichen, um eine mögliche Überversorgung zu ermitteln. Ist weder eine Über- noch eine Unterversorgung gegeben, dann ist die Ressourcenbereitstellung im System optimal. (Islam et al., 2012; Weinman, 2011)

Die beiden unterschiedlichen Zustände von Über- und Unterversorgung können auch als Elastizität aus Kunden- oder Anbietersicht verstanden werden. Die Messung der Unterversorgung entspricht der Elastizität aus Kundensicht, da es für den Kunden vorwiegend relevant ist, dass die gewünschte Performanz des Systems unabhängig von der Auslastung erhalten bleibt (Almeida et al., 2013). Folglich kann aus reiner Kundensicht ein System bereits als elastisch bezeichnet werden, wenn dieses skalierbar ist und genügend Ressourcen zur Verfügung hat. Im Umkehrschluss kann die Messung der Unterversorgung als Elastizität aus Sicht des Anbieters verstanden werden, da der Anbieter Interesse daran hat, unnötige Kosten für den Betrieb des Systems zu vermeiden (Almeida et al., 2013).

Der zweite wesentliche Punkt ist, wie in Kapitel 2.2.5 erwähnt, der zeitliche Aspekt (Herbst et al., 2013). Die Auslastung eines Systems verändert sich permanent und die zuvor erläuterten Messungen von Über- und Unterversorgungen werden immer zu einem bestimmten Zeitpunkt

oder über einen bestimmten Zeitraum durchgeführt. Mit Elastizität wird versucht, zu jedem Zeitpunkt die Ressourcen an die sich verändernde Auslastung anzupassen und so die Dauer der Zustände von Über- und Unterversorgungen möglichst gering zu halten.

2.3 Mandantenfähigkeit

Mandantenfähigkeit ist die Eigenschaft eines Systems, mehrere Mandanten gleichzeitig mit derselben Code-Basis zu bedienen und auf deren individuelle Anforderungen durch mandantenspezifische Konfigurationen einzugehen. In Verbindung mit der steigenden Popularität von SaaS (Software-as-a-Service) Anwendungen wird dieser Ansatz immer häufiger von Anbietern in Cloud-Umgebungen umgesetzt. Das Hauptziel dabei ist die Kostenreduktion für Betrieb, Wartung, Lizenzierung und Bereitstellung von Software. (Gao et al., 2013)

Wie die zuvor in diesem Kapitel behandelten Begriffe sind auch die Definitionen für Mandantenfähigkeit in der Literatur wage und nicht eindeutig formuliert. Um ein gemeinsames Verständnis zu schaffen, definieren Bezemer und Zaidman (2010) die Begriffe Mandant und Mehrmandantensystem. Demnach nutzen mehrere Mandanten in einem Mehrmandantensystem dieselbe Anwendungs- und Datenbankinstanz und teilen somit auch dieselben Hardware Ressourcen. Einzelne Mandanten können entsprechend den unterschiedlichen Anforderungen konfiguriert werden, sodass diese für den Benutzer wie eine Applikation in einer eigenständigen Umgebung wahrgenommen wird. In weiterer Folge ist ein Mandant die organisatorische Einheit, die eine Applikation innerhalb eines Mehrmandantensystems nutzt. Typischerweise sind Mandanten in mehrere Benutzer unterteilt, welche die Stakeholder der jeweiligen Organisation darstellen. (Bezemer & Zaidman, 2010)

2.3.1 Arten von Mehrmandantensystemen

Bezemer und Zaidman (2010) unterscheiden Mehrmandantensysteme in die drei verschiedenen Arten Multi-User-Systeme, Multi-Mandanten-Systeme und Multi-Instanz-Systeme. Die Unterscheidung basiert darauf, welche Schichten des Systems exklusiv von einem Mandanten genutzt oder von mehreren Mandanten geteilt werden. Der Begriff Multi-User-System bezieht sich hierbei auf ein Mehrmandantensystem, in dem sich die Mandanten den gesamten System-Stack sowie auch die Systemkonfiguration teilen. Dementsprechend bezieht sich ein User in diesem Kontext auf einen Mandanten, der wiederum mehrere Endbenutzer der Software beinhalten kann. Maßgeblich hierbei ist, dass alle Endbenutzer mandantenübergreifend dasselbe System mit derselben Systemkonfiguration nutzen. Teilen sich mehrere Mandanten dieselbe Applikationsinstanz und haben aber individuelle Konfigurationen, so wird dies als klassisches Multi-Mandanten-System bezeichnet. Wird für jeden Mandant eine eigene Applikationsinstanz zur Verfügung gestellt, so handelt es sich um ein Multi-Instanz-System.

Gao et al. (2013) beschreiben ein Spektrum von Mandantenfähigkeit für SaaS noch detaillierter mit fünf verschiedenen Typen, die sich, ähnlich wie nach Bezemer und Zaidman (2010), jeweils im Grad der Isolation und Exklusivität der Nutzung von Infrastruktur, Datenbank, Applikation und

Konfiguration unterscheiden. Wie in Abbildung 2-6 ersichtlich, bezieht sich der Typ 1 eines Mehrmandantensystems auf ein Multi-User-System nach Bezemer und Zaidman (2010), in dem alle Schichten des Systems von allen Mandanten geteilt werden. Der Typ 2 entspricht dem Multi-Mandanten-System, da hier die Applikationsschicht geteilt wird, aber für jeden Mandanten eine individuelle Konfiguration möglich ist.

Nach der Definition von Bezemer und Zaidman (2010) weist sich ein Multi-Instanz-System durch eine eigene Applikationsinstanz pro Mandant aus. Entsprechend Abbildung 2-6 trifft dies auf die Typen 3, 4 und 5 zu. Da Gao et al. (2013) zusätzlich nach den Schichten Infrastruktur und Datenbank unterscheiden, ergeben sich diese drei Typen von Multi-Instanz-Systemen. Ein Mehrmandantensystem vom Typ 5 stellt für jeden Mandanten exklusiv den gesamten System-Stack mit Infrastruktur, Datenbank, Applikation und Konfiguration zur Verfügung, sodass jeder Mandant vollkommen isoliert von anderen ausgeführt wird. Die Typen 3 und 4 bilden eine weitere Abstufung zwischen exklusiver oder gemeinsamer Nutzung der Datenbankschicht.

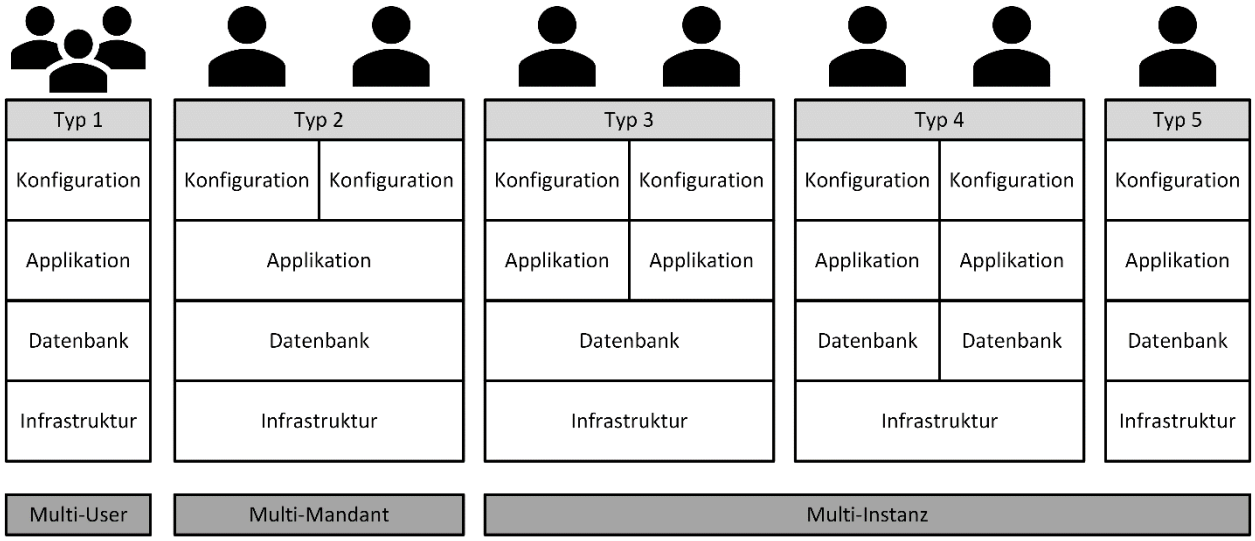


Abbildung 2-6 Spektrum von Mehrmandantensystemen (in Anlehnung an Gao et al.,2013 und Bezemer und Zaidman, 2010)

Wie in Abbildung 2-6 ersichtlich, stellen Typ 1 und Typ 5 keine Mehrmandantensysteme im Sinne der unter Kapitel 2.3 gegebenen Definition dar. Nach dieser wird Mandantenfähigkeit durch die Eigenschaft eines Systems definiert, mehrere Mandanten gleichzeitig mit derselben Code-Basis zu bedienen und auf individuelle Anforderungen eingehen zu können. In diesem Sinne besitzt ein System vom Typ 1 keine Mandantenfähigkeit, da es nicht in der Lage ist, unterschiedliche Anforderungen der Mandanten durch verschiedene Konfigurationen abzudecken. Es gibt nur eine gemeinsame Konfiguration für alle Beteiligten, weshalb diese in so einem System eher als Benutzer und nicht als Mandanten angesehen werden können. Weiters ist ein System vom Typ 5 aufgrund der fehlenden Fähigkeit, mehrere Mandanten innerhalb desselben Systems abbilden zu können, nicht mandantenfähig. Da aber davon auszugehen ist, dass es in einem Typ 5 System mehrere Benutzer gibt, kann dieses ebenso wie ein Typ 1 System als Multi-User System bezeichnet werden. Die Unterschiede zwischen Typ 1 und Typ 5 Systemen sind somit technisch betrachtet minimal und rein organisatorischer Natur. Aus Anwendersicht bedeutet das, dass die Benutzer in einem Typ 5 System derselben Organisation angehören während die Benutzer in

einem Typ 1 System aus verschiedenen Organisationen sein können und somit unabhängig voneinander agieren. Aus der Sicht des Anbieters liegt der Unterschied in der Anzahl der bereitgestellten Systeme. Typ 1 stellt ein großes System für alle Kunden zur Verfügung, Typ 5 jeweils ein System pro Kunde.

Wie einleitend erwähnt, beziehen sich die verschiedenen Typen im Spektrum von Mehrmandantensystemen jeweils auf einen unterschiedlichen Grad der gemeinsamen Nutzung von Technologieschichten. Je höher der Grad der gemeinsamen Nutzung ist, umso geringer sind die Aufwände für Bereitstellungen, Wartungen und Updates und in weiterer Folge die zugehörigen Kosten. Je geringer hingegen der Grad der gemeinsamen Nutzung ist, desto höher ist die Unabhängigkeit zwischen den Mandanten. Diese Unabhängigkeit führt zu einer höheren Skalierbarkeit und eröffnet Möglichkeiten zur Elastizität. Bei der Auswahl des Typen für ein Mehrmandantensystem kommt es also zu einem Kompromiss zwischen Kosten und Skalierbarkeit. Dieser Kompromiss wird im Kapitel 2.4.4 weiter behandelt.

2.3.2 Auswirkungen von Mandantenfähigkeit

Ein grundlegender Vorteil eines Mehrmandantensystems ist die Erleichterung der Bereitstellung eines Systems für den Anbieter (Bezemer & Zaidman, 2010, S. 1). Des Weiteren entstehen Aufwände in Bezug auf Betrieb und Wartung des Systems in den gemeinsam genutzten Technologieschichten nur einmal für alle Mandanten. Infolgedessen hängt der Grad der Einsparung vom Grad der gemeinsamen Nutzung und somit vom Typ des Mehrmandantensystems ab. In erster Linie betrifft dies Typ 1 und Typ 2, da hier alle Technologieschichten gesammelt für alle Benutzer oder Mandanten bereitgestellt werden. Bei den Multi-Instanz-Systemen sind demnach diese Vorteile vermeintlich erheblich geringer. Dennoch steigt die Popularität von Multi-Instanz-Systemen gemeinsam mit der von Virtualisierung und Cloud-Plattformen. Laut Bezemer und Zaidman (2010) sind Multi-Instanz Systeme für eine relativ geringe Anzahl von Mandanten besser geeignet. Begründet wird diese Aussage durch erhöhten Wartungsaufwand der einzelnen Instanzen aufgrund von Updates. Dieser Nachteil kann aber durch die Verwendung von Cluster-Management-Systemen und Container-Technologien, mithilfe derer ein Update gesammelt für mehrere Instanzen ausgerollt werden kann, teilweise entkräftet werden. Diese Aspekte werden in den Kapiteln 2.4.3 und 2.4.5 näher erörtert.

Der zweite Vorteil von Mehrmandantensystemen bezieht sich auf die effizientere Hardware-Ressourcennutzung. Als Zielgruppe dessen gelten besonders kleine und mittlere Unternehmen (KMU), da diese einerseits oft limitierte finanzielle Ressourcen haben und andererseits nicht die Leistung von exklusiv bereitgestellter Infrastruktur benötigen (Bezemer & Zaidman, 2010, S. 1). Über die geteilte Nutzung von mehreren Mandanten können Anbieter die Betriebs- und Wartungskosten senken und dadurch wiederum ihre Leistungen günstiger anbieten.

Bezemer und Zaidman (2010) zeigen für Mandantenfähigkeit auch Herausforderungen auf. Da die Hardware-Ressourcen von mehreren Mandanten geteilt werden, muss sichergestellt sein, dass jeder Mandant genügend Ressourcen zur Verfügung hat. Nimmt ein Mandant einen großen Teil der Ressourcen in Anspruch, hat dies unweigerlich eine Auswirkung auf die Performanz der

anderen Mandanten. In Multi-Instanz-Systemen besteht die Möglichkeit, einzelne Mandanten weitgehend zu isolieren und die verwendeten Ressourcen pro Mandanten zu beschränken. Dadurch wirkt sich eine hohe Auslastung eines einzelnen Mandanten in der Theorie nicht auf die Performanz der anderen aus. Dieser Lösungsansatz hat jedoch eine ineffizientere Ressourcennutzung des Gesamtsystems zur Folge, da verfügbare Ressourcen anderer, unausgelasteter Mandanten ungenutzt bleiben (Bezemer & Zaidman, 2010). Mithilfe von Elastizität kann dieser Ineffizienz entgegengewirkt werden, indem die verfügbaren Ressourcen einzelner Mandanten bei Bedarf erweitert oder reduziert werden.

Im Hinblick auf Security muss bei Mehrmandantensystemen besonders auf die Datensicherheit zwischen den Mandanten geachtet werden. Sicherheitslücken entstehen, wenn die Daten eines Mandanten für andere Mandanten zugänglich sind (Bezemer & Zaidman, 2010). Dies ist besonders für Mehrmandantensysteme wichtig, welche sich eine Datenbank teilen. In Mehrmandantensystemen vom Typ 4 mit getrennten Datenbanken wird dieses Risiko aufgrund besserer Isolierung der Daten pro Mandanten reduziert.

Des Weiteren ist Hochverfügbarkeit ein wichtiger Punkt in Mehrmandantensystemen, da Anpassungen und Wartungen einzelner Mandanten keine Auswirkung auf andere Mandanten haben dürfen (Bezemer & Zaidman, 2010). Je mehr Technologieschichten zwischen Mandanten geteilt werden, umso schwieriger ist es, diese Anforderung einzuhalten. Umgekehrt wird dies einfacher, wenn die Technologieschichten einzelner Mandanten unabhängig voneinander sind.

Als weitere Herausforderung von Mandantenfähigkeit nennen Bezemer und Zaidman (2010) Skalierbarkeit. Zum einen wird das durch die gemeinsame Verwendung einer Applikations- und Datenbankinstanz von mehreren Mandanten begründet, wodurch nur eine vertikale Skalierung ermöglicht wird, jedoch keine horizontale. Diese Annahme trifft für ein Mehrmandantensystem vom Typ 1 und 2 sowie in Bezug auf die Datenbankinstanz auch auf Typ 3 zu. Allerdings ist dies nicht der Fall in einem Mehrmandantensystem vom Typ 4. Hier sind Applikation und Datenbank der Mandanten unabhängig voneinander und somit auch skalierbar. Pro Mandanten ist dadurch die Skalierbarkeit gleichzustellen mit der eines Einzelmandantensystems vom Typ 5, da die Möglichkeit besteht, einen Mandanten auf einem exklusiven Server bereitzustellen. Auf das Gesamtsystem betrachtet ergibt sich sogar eine Verbesserung der Skalierbarkeit, da in so einem System eine horizontale Skalierung über mehrere Applikations- und Datenbankserver möglich ist. In Summe wirkt sich die Mehrmandantenfähigkeit von Typ 4 also positiv auf die Skalierbarkeit eines Systems aus. Weiters deklarieren Bezemer und Zaidman (2010) Skalierbarkeit als Herausforderung, da Mandanten aus verschiedenen Ländern in einem System abgebildet werden können und sich dadurch unterschiedliche Anforderungen ergeben. Als Beispiel wird die Vorgabe der EU genannt, dass elektronische Rechnungen, welche innerhalb der EU erstellt werden, auch in der EU gespeichert werden müssen (Ganek & Corbi, 2003). Auch dieser Aspekt stellt für die Typen 1 bis 3 ein Problem dar, bei denen das System an einen einzigen physischen Host gebunden ist. Bei einem Mehrmandantensystem vom Typ 4 oder 5 ist mit der horizontalen Skalierung ebenso eine geografische Verteilung der Applikations- und Datenbankserver möglich. Cloud-Anbieter ermöglichen dafür die Bereitstellung von Servern oder Servergruppen in verschiedenen Regionen. Bei besonderen geografischen Anforderungen können entsprechende

Bereitstellungsregeln auf die jeweiligen Instanzen angewendet werden, um diese Ansprüche zu erfüllen.

2.4 Container-Technologien

Virtualisierung in der Informationstechnologie wurde bereits in den 1970er Jahren erstmalig verwendet, um mehrere unabhängige Instanzen von Betriebssystemen gleichzeitig auf derselben Maschine zu betreiben. Durch das Aufkommen von Cloud-Umgebungen wurde diese Technologie erneut aufgegriffen und weiterentwickelt. Die Ziele dabei waren unter anderem bessere Portabilität, Effizienz und Zuverlässigkeit. Durch das Betreiben mehrerer Betriebssysteme auf demselben physischen Server wurde außerdem eine bessere Ressourcennutzung erreicht. (Marinescu, 2018)

Ein weiterer nennenswerter Vorteil von Virtualisierung ist die Erstellung von benutzerdefinierten Systemumgebungen. Dadurch kann die virtualisierte Umgebung unabhängig vom Hostsystem konfiguriert und an die jeweiligen Anforderungen angepasst werden. In weiterer Folge können das Betriebssystem sowie alle nötigen Abhängigkeiten und Konfigurationen in einem Image gekapselt werden. Auf diese Weise kann die virtualisierte Umgebung in ihrem Ursprungszustand beliebig oft wiederhergestellt werden. (Alfonso et al., 2017)

Mittlerweile ist Virtualisierung in der Informationstechnologie, und vor allem im Bereich Cloud-Computing, allgegenwärtig. Nach der ursprünglichen Form über virtuelle Maschinen haben sich Container-Technologien als leichtgewichtige Alternative entwickelt.

2.4.1 Virtuelle Maschinen und Container

Die Virtualisierung von virtuellen Maschinen benötigt zusätzliche Managementsoftware in Form eines Hypervisors, der den gleichzeitigen Betrieb mehrerer Betriebssysteme verwaltet (Marinescu, 2018). Zudem wird auf jeder virtuellen Maschine ein vollständiges Betriebssystem inklusive aller zugehörigen Funktionen und Dienste ausgeführt, welche zwar für die laufenden Anwendungen benötigt werden, jedoch nicht mehrfach notwendig wären, wenn diese zentral zur Verfügung stünden. Da die Applikationen auf einige dieser Funktionen und Dienste aber nur innerhalb der eigenen virtuellen Maschine zugreifen können, ergibt sich die Notwendigkeit der mehrfachen Ausführung. Infolgedessen bringen sowohl Hypervisor als auch die eigenständigen Betriebssysteme pro virtuelle Maschine einen zusätzlichen Overhead mit sich.

Während virtuelle Maschinen Hardware-Virtualisierung nutzen, basieren Container-Technologien auf Betriebssystem-Virtualisierung. Das heißt, bei virtuellen Maschinen grenzt der Hypervisor einen Teil der Hardware-Ressourcen des Hostsystems virtuell ab und simuliert diesen als eigenständige Hardware-Plattform für die virtuelle Maschine. Die Betriebssystem-Virtualisierung für Container wendet dasselbe Prinzip auf einer Ebene höher an. Das Betriebssystem stellt für den Container einen isolierten Prozessraum zur Verfügung und simuliert dem Container, er wäre exklusiv auf dem Betriebssystem installiert. Dementsprechend gibt es bei Container-Technologien keinen Hypervisor sowie nur ein Host-Betriebssystem, auf welchem die Container-

Plattform ausgeführt wird. Die Container nutzen direkt das Host-Betriebssystem und beinhalten selbst nur die Applikation und zugehörige Bibliotheken und Services, die der jeweilige Container für die Ausführung benötigt. Wie in Abbildung 2-7 dargestellt, haben Container dadurch einen geringeren Overhead als virtuelle Maschinen und profitieren dennoch von denselben Vorteilen wie Portabilität, Unabhängigkeit und Isolation. Diese Eigenschaften haben sich vor allem für Cloud-Anwendungen als besonders nützlich herausgestellt und in Kombination mit Cluster-Management-Systemen eröffnen sich weitere vielseitige Möglichkeiten. Darum haben Container-Technologien weitgehende Akzeptanz im Cloud-Computing Umfeld erreicht. (Marinescu, 2018; Zeng et al., 2017)

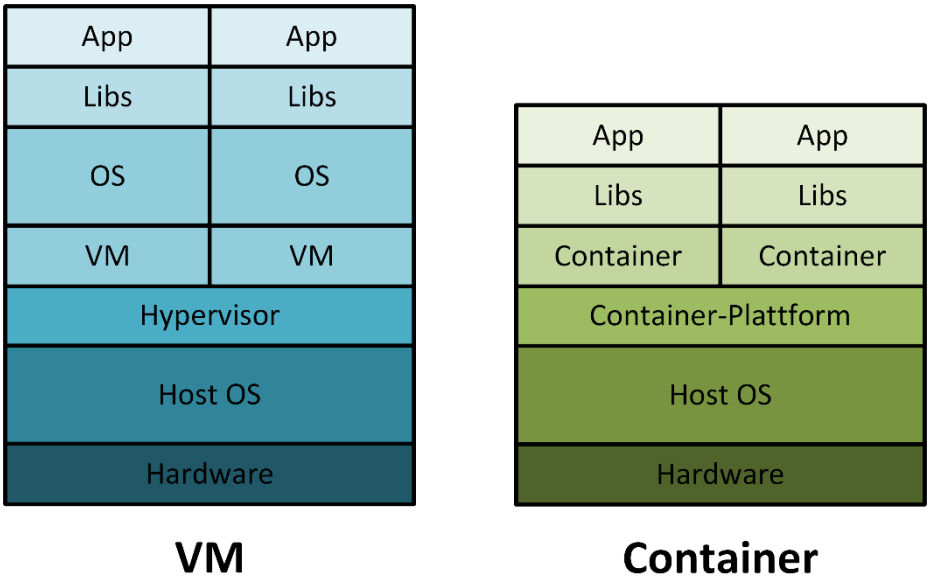


Abbildung 2-7 Vergleich zwischen virtueller Maschine und Container (in Anlehnung an Marinescu, 2018 und Alfonso, Calatrava und Moltó, 2017)

2.4.2 Eigenschaften von Containern

Hinsichtlich der in Kapitel 2.4.1 genannten Aspekte ergeben sich weitere konkrete Vorteile für Container, wie die erhöhte Produktivität in Bezug auf Implementierung und Betrieb. Aufgrund der Isolation und Unabhängigkeit einzelner Container können die jeweils passendsten Programmiersprachen, Bibliotheken und Werkzeuge verwendet werden. Zudem müssen Systeme nicht mehrfach in verschiedenen Umgebungen eingerichtet werden. Jeder Container beinhaltet alle für die Ausführung benötigten Komponenten, Laufzeitumgebungen und Bibliotheken. Demnach kann ein Container ohne weitere Konfiguration auf einem beliebigen Server ausgeführt werden, sofern die jeweilige Container-Plattform vorhanden ist. Die Tatsache, dass Container einfach und unabhängig gestartet, gestoppt und migriert werden können, trägt zur einfachen Replikation bei, welche sich in weiterer Folge positiv auf die Skalierbarkeit und somit auch auf die Elastizität eines Systems auswirkt. (Marinescu, 2018) Diese Vorteile von Containern decken sich mit den Vorteilen von Microservices. Aufgrund dieser Parallelen werden Container-Technologien häufig in Kombination mit diesem Architekturansatz genutzt werden.

Des Weiteren bieten sich Container auch ohne einer Microservice-Architektur an, um Mandanten innerhalb eines Mehrmandantensystems abzubilden. Jeder Mandant kann, in sich mit einer monolithischen Architektur, innerhalb eines Containers ausgeführt werden. Damit wird sichergestellt, dass sich jeder Mandant gleich verhält und dennoch isoliert von anderen ausgeführt wird. Wie unter Kapitel 2.3 beschrieben, ergeben sich diese Möglichkeiten aufgrund der Unabhängigkeit zwischen den Mandanten. Container sind das zugehörige Werkzeug, um diese theoretischen Vorteile in der Architektur praktisch umzusetzen.

2.4.3 Elastizität mit Containern

Virtualisierung im Allgemeinen hat positive Auswirkungen auf Elastizität, welche unter anderem durch die Abstraktion der Infrastruktur gegenüber den höherliegenden Schichten begründet werden. Dadurch werden logische Einheiten von physischen Ressourcen entkoppelt und voneinander unabhängig. Die Elastizität steigt wiederum aufgrund höherer Flexibilität für Bereitstellung, Freigabe und Portabilität der logischen Einheiten. (Marinescu, 2018)

Speziell Container-Technologien haben noch weitere Vorteile für Elastizität. In Kapitel 2.2.6 werden Voraussetzungen und Begünstigungen für Elastizität behandelt, wovon einige im Einklang mit Container-Technologien stehen und durch deren Einsatz umgesetzt oder gefördert werden können. Über Container werden Systemkomponenten und deren Replikate mit all ihren Abhängigkeiten in gekapselter Form auf Basis eine Container-Images zur Verfügung gestellt. Aufgrund dieser Funktionsweise ist jeder Container eine abgeschlossene, unabhängige Einheit, wodurch Container sich besonders gut für Replikation, Verteilung und Parallelisierung eignen. Konzepte wie lose Koppelung und Zustandslosigkeit liegen in der Natur von Containern und bieten sich somit auch für deren Umsetzung an.

2.4.4 Mehrmandantensysteme mit Containern

Wie in Kapitel 2.3.1 dargelegt, kommt es bei der Auswahl des Typen für ein Mehrmandantensystem zu einem Kompromiss zwischen Kosten und Skalierbarkeit. Bei Typ 1 gibt es nur ein großes, gemeinsames System für alle Anwender, sodass auch die Kosten für Bereitstellung und Wartung nur einmalig auftreten. Andererseits ist in so einem System die Skalierbarkeit auf die vertikale Skalierung des jeweiligen Hostsystems begrenzt. Dieselbe Einschränkung trifft auch auf Typ 2 zu. Die Kosten sind im Vergleich zu Typ 1 jedoch hinsichtlich der Wartung von getrennten Konfigurationen pro Mandanten höher. Allerdings werden weiterhin Kosteneinsparungen aufgrund der gemeinsamen Nutzung der Applikationsinstanz und Datenbank begünstigt. Mit Typ 3 Systemen werden zusätzliche Kosten bei der Bereitstellung der Applikation in Kauf genommen, zu Gunsten der horizontalen Skalierbarkeit der Applikationsschicht. Die gemeinsame Datenbank aller Mandanten ist dabei aber weiterhin an einen Server gebunden. Bei Typ 4 erfolgt derselbe Kompromiss zwischen Kosten und Skalierbarkeit in Bezug auf die Datenbankschicht. In einem Typ 5 System werden alle Technologie-Schichten pro Mandanten unabhängig bereitgestellt, wodurch die Skalierbarkeit des Gesamtsystems über die Mandanten hinweg profitiert. Allerdings ergeben sich dadurch auch die

höheren Kosten für Bereitstellung, Betrieb und Wartung einer separaten Infrastruktur pro Mandanten. Daraus abgeleitet stellt einerseits Typ 1 ein System mit niedrigen Kosten und schlechter Skalierbarkeit dar, und andererseits Typ 5 ein System mit hohen Kosten und guter Skalierbarkeit.

Dieses Dilemma kann durch die Verwendung von Container-Technologien gelöst werden. Erreicht wird das durch die Funktionsweise von Containern, nach der Container-Instanzen immer auf Basis eines Container-Images erstellt werden. Diese Container-Images können in einer zentralen Container-Registry bereitgestellt und ab diesem Zeitpunkt beliebig für die Instanziierung von neuen Containern genutzt werden. Zum einen wird dadurch sichergestellt, dass alle neuen Bereitstellungen automatisch die aktuelle Version verwenden, zum anderen müssen neue Versionen jeweils nur einmal in der Container-Registry aktualisiert werden. Nach dem Neustart eines Containers kann automatisch auf die aktualisierte Version zugegriffen und so ein Update über alle Container-Instanzen ausgerollt werden.

In Bezug auf Mehrmandantensysteme können also die Versionen einer Applikation in der zentralen Container-Registry organisiert werden. Neue Mandanten starten automatisch eine Applikationsinstanz mit der aktuellsten Version und bestehende Mandanten können relativ einfach auf neue Versionen aktualisiert werden. Diese Funktionsweise ist nicht nur auf die Applikations-, sondern ebenso auf die Datenbankschicht anwendbar. Auf diese Weise ist eine einfache Bereitstellung einer Datenbankinstanz für neue Mandanten möglich. Des Weiteren können neue Versionen von Datenbanksystemen unkompliziert auf alle Mandanten ausgerollt werden.

In Hinblick auf die Bereitstellung von Datenbanken als Container muss allerdings der geforderte Zeithorizont der zu speichernden Daten beachtet werden. Sofern es sich um volatile Daten handelt, welche beim Herunterfahren der Container-Instanz verworfen werden können, ist eine Speicherung der Daten direkt im Container unproblematisch. Dies könnte beispielsweise in einem System der Fall sein, welches seine Daten bei jedem Neustart aus einem führenden, externen System bezieht. Besteht jedoch die Anforderung der dauerhaften Datenspeicherung, ist eine Trennung der Datenbank-Instanz und der Daten selbst erforderlich. In so einem Fall können die Daten pro Mandanten in einem Verzeichnis auf einem zentralen Fileserver abgelegt werden. Die jeweiligen Verzeichnisse der Mandanten werden in den zugehörigen Container-Instanzen der Datenbank als Laufwerk eingebunden.

Durch diese Verwendung von Containern wird der Kompromiss zwischen Kosten und Skalierbarkeit relativiert, da separate Technologieschichten pro Mandanten, ohne der Notwendigkeit von mehrfacher Bereitstellung, Wartung und Aktualisierungen, möglich sind. So kann die erhöhte Skalierbarkeit von Typ 4 mit den reduzierten Kosten des Typs 2 erreicht werden.

Erweiternd zu der Nutzung von Containern für ein Mehrmandantensystem ist eine Kombination mit virtuellen Maschinen wie in Abbildung 2-8 möglich. So können beispielsweise eigene virtuelle Maschinen pro Mandanten verwendet werden, auf denen wiederum die Applikationen für diesen Mandanten in Form von Containern gestartet werden. Auf diese Weise wird eine höhere Isolierung zwischen den Mandanten und somit eine höhere Sicherheitsabgrenzung erreicht (Alfonso et al., 2017). Dieser Ansatz führt implizit auch zu einer klaren Ressourcenabgrenzung

zwischen den einzelnen Mandanten. Wie in Kapitel 2.3.2 erörtert, hat so einerseits die hohe Auslastung eines einzelnen Mandanten korrekterweise keine Auswirkungen auf andere, andererseits sinkt dadurch die Effizienz der Ressourcennutzung.

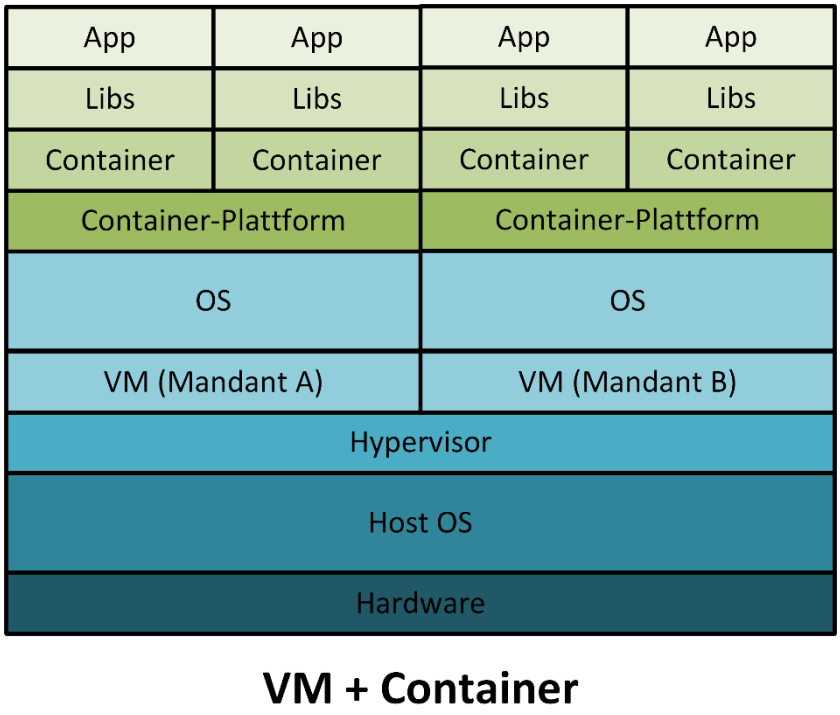


Abbildung 2-8 Kombination von virtuellen Maschinen mit Containern (in Anlehnung an Alfonso et al., 2017)

2.4.5 Hochverfügbarkeit und Ausfallsicherheit mit Containern

Durch die Verwendung von Containern wird es möglich, einzelne Komponenten des Systems zu replizieren und über verschiedene Server, Datencenter und geografische Standorte zu verteilen. Wenn diese Eigenschaften genutzt werden, ermöglichen Container dadurch eine höhere Verfügbarkeit des Gesamtsystems und sichern gleichzeitig gegen den Ausfall einzelner logischer oder physischer Komponenten ab. Ein weiterer wichtiger Aspekt ist die Weiterentwicklung und Aktualisierung eines Systems. Monolithische Systeme, deren Komponenten nicht unabhängig voneinander ausgeführt und bereitgestellt werden können, erfordern für Aktualisierungen das Stoppen und erneute Bereitstellen des gesamten Systems. Durch die hohe Unabhängigkeit und Replizierbarkeit von Containern kann dieses Problem gelöst werden. Die einzelnen Instanzen einer replizierbaren Komponente können parallel mit unterschiedlichen Versionen gestartet werden. Es wird also die neue Version einer Komponente zur Verfügung gestellt, während die Instanz mit der alten Version noch weiterläuft. Im Load-Balancing erfolgt keine Unterscheidung der Instanzen, wodurch beide zu gleichen Teilen eingehende Anfragen erhalten. Sobald die Instanz mit der alten Version nicht mehr benötigt wird, kann diese beim Load-Balancer abgemeldet oder direkt heruntergefahren werden. In beiden Fällen werden neue Requests nur noch auf die Instanz mit der aktualisierten Applikationsversion weitergeleitet. Auf diese Weise sind Aktualisierungen ohne Ausfallzeiten möglich.

Unter anderem wird dies erreicht, weil eine Applikation in einem Container gemeinsam mit all ihren Abhängigkeiten bereitgestellt wird. Dadurch können zwei Container-Instanzen beispielsweise unterschiedliche Versionen derselben Bibliothek verwenden, ohne sich gegenseitig zu beeinflussen (Dragoni et al., 2017). Allerdings muss bei dieser Praxis der parallele Betrieb unterschiedlicher Versionen auch in der Applikation und Datenhaltung berücksichtigt werden. Dazu ist es notwendig die Abwärtskompatibilität sowie die Einhaltung aller definierten Schnittstellen zu anderen Komponenten sicherzustellen.

2.5 Cluster-Management

Aufgrund der fortschreitenden Digitalisierung und Vernetzung sehen sich moderne Software-Systeme mit immer höheren Auslastungen konfrontiert. In der Vergangenheit wurden extreme Anforderungen an Rechenleistung mit Supercomputern abgedeckt. Allerdings gehen die hohen Auslastungen von manchen Systemen über die Grenzen der vertikalen Skalierung hinaus und erfordern daher verteilte Systeme mit ausgeprägter horizontaler Skalierbarkeit. Wie in Kapitel 2.4 verdeutlicht, bietet die Virtualisierung über virtuelle Maschinen oder Container-Technologien gute Voraussetzungen für diese Anforderungen. Diese Entwicklungen haben zu einem starken Aufstieg von Cloud-Anwendungen geführt und damit auch von privaten Cloud-Anbietern wie Amazon Web Services, Microsoft Azure oder Google Cloud Plattform. (Alfonso et al., 2017; Sadashiv & Kumar, 2011)

2.5.1 Cluster-Computing, Grid-Computing und Cloud-Computing

Die Verarbeitung von Daten und Anfragen in solchen verteilten Systemen wird auch als Cluster-Computing bezeichnet. Ein Cluster ist nach Sadashiv und Kumar (2011) eine Sammlung von parallelisierten oder verteilten Rechnern, welche über ein leistungsstarkes Netzwerk verbunden sind. Diese Rechner verarbeiten zusammen sehr rechen- oder datenintensive Aufgaben, welche von einzelnen Rechnern nicht zu bewältigen wären. Cluster werden meist genutzt, um Konzepte wie Hochverfügbarkeit oder Load-Balancing umzusetzen. Ersteres wird dabei durch redundante Knoten sichergestellt, welche bei einem Ausfall eines Knotens die Aufgaben übernehmen und somit einen Single-Point-of-Failure eliminieren. Dies passiert in Zusammenarbeit mit einem Load-Balancer, der eingehende Anfragen nur an die verfügbaren Knoten weiterleitet. Nach außen wirkt ein Cluster wie eine einzige Recheneinheit, die einzelnen Komponenten sowie die interne Komplexität werden gegenüber anderen Systemen gekapselt (Sadashiv & Kumar, 2011).

Sadashiv und Kumar (2011) unterscheiden Cluster-Computing explizit von den verwandten Begriffen Grid-Computing und Cloud-Computing. Ein Computing-Grid ist ein System, welches untereinander unabhängige Ressourcen in Abstimmung mit deren Verfügbarkeiten für die Bearbeitung einer einzelnen großen Aufgabe koordiniert. Dazu gehören Auswahl, Verteilung und Aggregation von Teilaufgaben auf meist geografisch separierte Ressourcen. Ermöglicht wird dies durch die Autonomie der einzelnen Teilaufgaben, sodass die beteiligten Ressourcen keine Kommunikation untereinander benötigen. In einem Computing-Grid können sowohl einzelne

Rechner als auch große cluster-basierte Systeme involviert sein (Chetty & Buyya, 2002; Sadashiv & Kumar, 2011).

Cloud-Computing wird von Buyya et al. (2009) als paralleles, verteiltes System mit mehreren, miteinander verbundenen, virtuellen Rechnern bezeichnet. Die Ressourcen in so einem System werden dynamisch unter der Einhaltung bestimmter SLA zur Verfügung gestellt. Dementsprechend sind sich Cluster-Computing und Cloud-Computing ähnlich, jedoch erweitert Cloud-Computing den Ansatz um die Aspekte der Virtualisierung und der dynamischen, bedarfsorientierten Bereitstellung. Damit bietet das Cloud-Computing Modell einen einfachen, schnellen und bedarfsbasierten Zugang zu verschiedenen Ressourcen wie Rechenleistung, Arbeitsspeicher, Festplattenspeicher oder Anwendungen. Je nachdem, auf welcher Ebene die Ressourcen für den Nutzer zur Verfügung gestellt werden, unterscheidet man zwischen Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) oder Software-as-a-Service (SaaS).

2.5.2 Anforderungen an Cluster-Management-Systeme

Cloud-Computing hat für Unternehmen, Forschung und Endnutzer einen Zugang zu vermeintlich unbegrenzten, frei konfigurierbaren Ressourcen ermöglicht. Damit diese Ressourcen effizient genutzt werden können, erfordert es umfangreiche Koordination und Überwachung. (Alfonso et al., 2017)

Wie in Kapitel 2.5.1 aufgezeigt, sind Cluster-Computing und Cloud-Computing Formen eines verteilten Systems, welches nach Marinescu (2018) eine Sammlung von Rechnern ist, die über ein Netzwerk und eine Verteilungssoftware verbunden sind. Eine derartige Verteilungssoftware wird in diesem Kontext als Middleware bezeichnet und ermöglicht den Rechnern die Koordination ihrer Aktivitäten sowie das Teilen von Ressourcen. In einem clusterbasierten oder cloudbasierten System kann diese Middleware auch als Cluster-Manager oder Cluster-Management-System verstanden werden.

Des Weiteren wird eine Reihe von Anforderungen an diese Middleware und somit an Cluster-Manager gestellt (Marinescu, 2018):

Zugriffstransparenz: Auf lokale und entfernte Informationen soll auf die gleiche Weise zugegriffen werden können.

Speicherorttransparenz: Auf Informationen wird zugegriffen, ohne deren genauen Speicherort zu kennen.

Gleichzeitigkeitstransparenz: Gleichzeitig ausgeführte Prozesse teilen dieselben Informationen, ohne sich gegenseitig zu beeinflussen.

Replikationstransparenz: Im System werden mehrere Instanzen ausgeführt, über die Benutzer und andere Anwendungen keine Kenntnis haben.

Ausfallstransparenz: Ausfälle einzelner Instanzen sind für Benutzer und andere Anwendungen nicht bemerkbar.

Migrationstransparenz: Informationen werden im System verschoben, ohne dass es eine Auswirkung auf die Operationen hat, die darauf ausgeführt werden.

Performanztransparenz: Das System passt sich an die Auslastung sowie QoS-Anforderungen an.

Skalierungstransparenz: Das System und die Anwendung können ohne Strukturänderungen und ohne Auswirkungen auf Benutzer oder andere Anwendungen skaliert werden.

2.5.3 Cluster-Management-Systeme und Elastizität

Auf Basis dieser Anforderungen ist erkennbar, dass die Funktionen eines Cluster-Managers über die von herkömmlichen Load-Balancer oder Job-Scheduler hinausgehen. Die beiden Anforderungen Performanz- und Skalierungstransparenz fordern die Überwachung der verfügbaren und genutzten Kapazitäten sowie die damit verbundene bedarfsorientierte Bereitstellung, Verwaltung und Freigabe dieser Kapazitäten auf Infrastruktur- und Softwareebene. Somit kann über einen Cluster-Manager Elastizität in einem Computing-Cluster umgesetzt werden. Voraussetzung dafür ist zum einen ein Monitoring-System, welches den Cluster-Manager mit den notwendigen Überwachungsdaten versorgt, um Entscheidungen zur Bereitstellung oder Freigabe von Kapazitäten zur ermöglichen. Entsprechend den Ausführungen in Kapitel 2.2 erfolgt dies auf der Basis von Entscheidungsprozessen und Elastizitätsdimensionen. Zum anderen erfordert es einen Ressource-Pool, auf den bei steigenden Kapazitätsanforderungen zurückgegriffen werden kann. (Marinescu, 2018)

Für die Skalierung in einem Computing-Cluster muss neben der horizontalen und vertikalen Skalierung noch eine weitere Unterscheidung zwischen der Skalierung auf Infrastruktur- und Software-Ebene getroffen werden. Die Skalierung auf Infrastruktur-Ebene bezieht sich auf die physischen Ressourcen im Cluster, welche im Rahmen von Cloud-Computing in der Regel über virtuelle Maschinen abgebildet werden. Dabei sind sowohl vertikale als auch horizontale Skalierung möglich. Vertikale Skalierung bedeutet in diesem Kontext entweder das Migrieren einer virtuellen Maschine auf eine leistungsstärkere physische Maschine oder die Erhöhung der zugewiesenen Ressourcen auf derselben physischen Maschine. Die häufiger verwendete horizontale Skalierung auf Infrastruktur-Ebene ist das Bereitstellen von zusätzlichen virtuellen Maschinen, wobei es irrelevant ist, auf welcher physischen Maschine diese Bereitstellung erfolgt. (Marinescu, 2018)

Dahingegen bezieht sich die Skalierung auf Software-Ebene auf das Bereitstellen oder Stoppen von Applikationsinstanzen. Dies erfordert zum einen die Replizierbarkeit der zu skalierenden Komponente sowie die Anwendung von Load-Balancing, um die zusätzlich bereitgestellten Instanzen auch entsprechend nutzen zu können. Das Load-Balancing muss dabei in der Lage sein, automatisch auf die sich ändernden Applikationsinstanzen zu reagieren. Im Gegensatz zur Skalierung auf Infrastruktur-Ebene ist die Skalierung auf Software-Ebene nur horizontal möglich. Dies ist damit begründet, dass in der Software nur eine logische Skalierung, jedoch keine Anpassung der zugrundeliegenden physischen Ressourcen vorgenommen werden kann.

3 TESTVORBEREITUNG

In der Testvorbereitung werden alle Rahmenbedingungen für die Durchführung der Tests definiert. Dazu gehören die Auswahl der verwendeten Technologien und Werkzeuge sowie die Definition der verwendeten Metriken und wie diese gemessen werden. Entsprechend den Anforderungen der Forschungsfrage werden Metriken sowohl für die Messung der Performanz als auch der Kosten festgelegt. Des Weiteren wird der Aufbau des statischen und elastischen Testsystems definiert. Über die Testpläne wird festgelegt, welche Szenarien im Rahmen der Tests abgebildet werden. Die Testpläne sind für beide Systeme ident. Abschließend wird in der Testvorbereitung die Hypothese für die Durchführung der Tests definiert und begründet.

3.1 Verwendete Technologien und Werkzeuge

Die im Rahmen dieser Arbeit verwendeten Technologien sind weitgehend dieselben, die auch in der vorangegangenen Arbeit verwendet wurden (Winkler, 2019). Die Testapplikation ist eine Node.js Anwendung, welche im Hintergrund eine MongoDB-Instanz für die Datenspeicherung verwendet. Eine nähere Beschreibung der Testapplikation erfolgt in Kapitel 3.6.1. Die Testapplikation wird in Docker-Containern ausgeführt. Für die Erstellung des Docker-Images wird die Node.js Applikation gemeinsam mit der MongoDB-Instanz in einem Docker-Image gebündelt, welches ein Alpine Linux Image als Basis verwendet. Die so erstellten Docker-Images werden auf Docker-Hub hochgeladen, in einem Kubernetes-Cluster bereitgestellt und entsprechend skaliert. Die darunterliegende Cloud-Plattform, auf der Kubernetes ausgeführt wird, ist Microsoft Azure. Deshalb erfolgt die Bereitstellung über den in Microsoft Azure integrierten Azure Kubernetes Service (AKS).

Des Weiteren wird der in Microsoft Azure integrierte Azure-Monitor verwendet, um die CPU-Auslastungen der Kubernetes-Cluster und der darin enthaltenen virtuellen Maschinen zu messen. Ebenso misst der Azure-Monitor die Anzahl der bereitgestellten virtuellen Maschinen und deren Status. Für die Ausführung der Lasttests, sowie die damit verbundene Messung der Performanz in Form von Antwortzeiten, wird Apache JMeter verwendet. Zur Verarbeitung der aus dem Azure-Monitor und JMeter gewonnen Messwerte, sowie zur Berechnung der benötigten Metriken, wird RStudio herangezogen.

3.2 Metriken für Performanz

Im Rahmen der durchgeführten Tests soll die Performanz der zu vergleichenden Systeme gemessen werden. Der Vergleich der Systeme erfolgt in erster Linie aufgrund der durchschnittlichen Antwortzeit $\bar{r}$. Zusätzlich werden die maximale Antwortzeit sowie die 99%- und 95%-Quantile erhoben, um weitere Aussagen über die Messdaten treffen zu können.

Für den Vergleich der Ergebnisse zwischen den Systemen wird pro Auslastungskurve $w \in W$ der prozentuale Unterschied der Performanz zwischen dem Vergleichssystem S_1 und dem Referenzsystem S_0 berechnet.

$$P_{S_1,\ w_x} = \frac{\bar{r}_{S_1,\ w_x}}{\bar{r}_{S_0,\ w_x}}$$

Aus den prozentualen Unterschieden $P_{S_1,\ w_x}$ wird anschließend P_{S_1} als das geometrische Mittel berechnet, welches als zentraler Vergleichswert der beiden Systeme in Bezug auf Performanz dient.

$$P_{S_1} = \sqrt[n]{\prod_{x=1}^{n}(P_{S_1,\ w_x})}$$

3.3 Metriken für Kosten

Weinman (2011) hat in seiner Arbeit wichtige Grundaspekte für die Berechnung von Elastizität beschrieben, welche in weiterer Folge für die Messung und Berechnungen in anderen weiterführenden Arbeiten aufgegriffen wurden. Einführend wird davon ausgegangen, dass jede Organisation versucht, die Nachfrage genau mit dem Angebot zu decken, wobei die Funktion der Nachfrage im Verlauf der Zeit mit $D(t)$ ausgedrückt wird und das Angebot in Form von verfügbaren Ressourcen mit $R(t)$. Für die überversorgten sowie die unterversorgten Zustände wird eine Kostengewichtung mit c_d und c_r festgelegt. Darauf basierend erstellte Weinman (2011) eine Formel zur Berechnung der zusätzlichen Kosten aufgrund von Über- und Unterversorgung.

$$L = \int_{t1}^{t2} [D(t) - R(t)] * c_d \mid D(t) > R(t) + \int_{t1}^{t2} [R(t) - D(t)] * c_r \mid R(t) > D(t)$$

Optimale Kapazität ist somit als $D(t) = R(t)$ definiert, wobei Weinman (2011) treffend beschreibt, dass sich die Kurven dieser beiden Funktionen in der realen Welt eher zufällig überschneiden, als dass diese dauerhaft deckungsgleich verlaufen. Die optimale Kapazität nach Weinman (2011) und dessen Unerreichbarkeit in der Praxis deckt sich mit der optimalen Elastizität in Kapitel 2.2.3.

Islam et al. (2012) bauen auf die Arbeit von Weinman (2011) auf und definieren Kosten für Über- und Unterversorgung sowie Metriken, wie diese gemessen werden können. Dadurch ergibt sich ein Modell, welches basierend auf diesen Kosten eine Bewertung der Elastizität eines Systems ermöglicht. Je geringer die Kosten für ein System sind, umso besser ist die Elastizität. Ein System ohne Kosten für Über- und Unterversorgung würde der optimalen Elastizität gleichkommen und ist somit unrealistisch.

3.3.1 Überversorgung

Für die Messung der Ressourcen wird zwischen benötigten Ressourcen $D(t)$, verfügbaren Ressourcen $R(t)$ und verrechneten Ressourcen $M(t)$ unterschieden. Benötigte Ressourcen sind jene, die das System zu einer bestimmten Zeit benötigt. Die Unterscheidung zwischen verfügbaren und verrechneten Ressourcen ist auf die Verrechnungsweise vieler Cloud-Anbieter zurückzuführen, nach der Ressourcen nicht unbedingt verfügbar sein müssen, um verrechnet zu werden. Dies kann beispielsweise die Bereitstellungszeit einer Ressource betreffen. Die Verrechnung einer Cloud-Ressource beginnt meist mit der Anfrage, diese bereitzustellen. Bis zu dem Zeitpunkt, zu dem eine Ressource tatsächlich für das System nutzbar ist, wird diese bereits verrechnet, obwohl sie noch nicht genutzt werden kann. Ähnliche Situationen können sich bei der Freigabe von Ressourcen ergeben, wenn die Verrechnung einer Ressource auf größeren Zeiteinheiten basiert und beispielsweise für jede begonnene Stunde erfolgt. In so einem Fall ist eine freigegebene Ressource nicht mehr für das System verfügbar, allerdings wird diese bis zum Ende der angebrochenen Stunde verrechnet. Als maßgebliche Metrik verwenden Islam et al. (2012) die verrechneten Ressourcen, sodass die Überversorgung $O(t)$ als die Differenz zwischen den verrechneten Ressourcen $M(t)$ und den benötigten Ressourcen $D(t)$ errechnet wird, wenn die verfügbaren Ressourcen $R(t)$ höher sind als die benötigten Ressourcen $D(t)$.

$$O(t) = M(t) - D(t) \mid R(t) > D(t)$$

Zusätzlich wird der Fall berücksichtigt, dass Ressourcen verrechnet werden, aber nicht verfügbar sind. Dies ist allerdings nur dann relevant, wenn auch die benötigten Ressourcen $D(t)$ höher sind als die verfügbaren Ressourcen $R(t)$. Somit kann die Formel wie folgt erweitert werden:

$$O(t) = [M(t) - D(t) \mid R(t) > D(t)] + [M(t) - R(t) \mid D(t) \geq R(t) \; und \; M(t) > R(t)]$$

Diese Überversorgung wird mit einem Kostenfaktor c_o für die jeweilige Ressourceneinheit multipliziert. Dadurch ergeben sich die Kosten für Überversorgung in Geldeinheiten pro Zeiteinheit. Um einen Vergleichswert mit anderen Systemen zu ermitteln, werden die durchschnittlichen Kosten C_o über die Dauer der Messung errechnet.

$$C_o(t) = O(t) * c_o$$

$$C_o = \frac{\sum C_o(t)}{t_e - t_s}$$

Diese Berechnung kann für mehrere Ressource-Typen erfolgen. Die gesamten Kosten für Überversorgung ergeben sich dann aus der Summe der einzelnen Kosten pro Ressource-Typ. (Islam et al., 2012)

3.3.2 Genutzte Ressourcen

Als Gegenstück zu den Kosten der Überversorgung können auch die Kosten der tatsächlich genutzten Ressourcen berechnet werden. Diese ergeben sich aus der Multiplikation der benötigten Ressourcen $D(t)$ mit dem Kostenfaktor für die verwendeten Ressourcen c_d, wobei dieser im Regelfall dem der Überversorgung c_o entsprechen sollte. Als Vergleichswert dienen die durchschnittlichen Kosten pro Minute C_d.

$$C_d(t) = D(t) * c_d$$

$$C_d = \frac{\sum C_d(t)}{t_e - t_s}$$

Bei gleichen Kostenfaktoren ergeben sich in weiterer Folge die Kosten der verrechneten Ressourcen aus der Summe der Kosten der genutzten Ressourcen und der Überversorgung.

$$C_m(t) = M(t) * c_d$$

$$C_m(t) = C_d(t) + C_o(t) \mid c_d = c_o$$

3.3.3 Unterversorgung

Für die Berechnung von Unterversorgung werden vorab definierte QoS-Ziele Q in Bezug auf die Performanz des Systems verwendet. Die Funktion $U(t)$ definiert die Menge der Anfragen, welche ein festgelegtes QoS-Ziel $q \in Q$ zum Zeitpunkt t nicht erreichen. Über die Funktion $U_{opt}(t)$ wird die Menge der unzufriedenstellenden Anfragen definiert, die in einem optimalen System mit unbegrenzten Ressourcen auftreten würden. Somit dient $U_{opt}(t)$ als Baseline. Des Weiteren wird über die Funktion $f()$ die Zuordnung eines monetären Kostenfaktors vorgenommen. Dementsprechend werden die Kosten der Unterversorgung $C_u(t)$ durch die nachstehende Formel berechnet, welche die Kosten der Unterversorgung in Geldeinheiten pro Zeiteinheit ausdrückt. (Islam et al., 2012)

$$C_u(t) = f(\,U(t)\,) - f(\,U_{opt}(t))$$

$$C_u = \frac{\sum C_u(t)}{t_e - t_s}$$

Werden mehrere QoS-Ziele in die Berechnung einbezogen, dann werden die Kosten für jedes $q \in Q$ summiert und als gesammelte Funktion ausgedrückt.

3.3.4 Ressourcenabweichung und Elastizität

Aus der Summe der Kosten für Überversorgung $C_o(t)$ und Unterversorgung $C_u(t)$ werden die Kosten für die Ressourcenabweichung $C_{o+u}(t)$ errechnet. Anschließend wird die Summe durch die Dauer des Zeitintervalls $[t_e - t_s]$ dividiert, um die durchschnittlichen Kosten pro Zeiteinheit zu erhalten.

$$C_{o+u}(t) = C_o(t) + C_u(t)$$

$$C_{o+u} = \frac{\sum C_{o+u}(t)}{t_e - t_s}$$

Die dadurch errechneten Kosten der Ressourcenabweichung C_{o+u} beschreiben die Elastizität eines Systems aufgrund der Messung für eine bestimmte Auslastungskurve $w_1(t)$. Abhängig von dieser ergeben sich in einem System unterschiedliche Zustände für Über- und Unterversorgung. Auf Auslastungskurven mit langsamen und gleichmäßigen Änderungen kann das elastische Verhalten eines Systems besser reagieren als auf unerwartete Sprünge. Das bedeutet C_{o+u} bezieht sich nur auf eine spezielle Messung, kann jedoch nicht als allgemeingültiger Wert für Elastizität in einem System angesehen werden. Um einen aussagekräftigeren Wert für die Elastizität eines Systems zu erhalten, müssen mehrere Messungen mit unterschiedlichen Auslastungskurven durchgeführt werden.

Islam et al. (2012) beschreiben die Berechnung einer gesammelten Metrik für Elastizität als das geometrische Mittel der Verhältnisse der Ressourcenabweichungskosten C_{o+u} aus den Messungen der verschiedenen Auslastungskurven $w \in W$ zwischen zwei zu vergleichenden Systemen. Das heißt, die Berechnung einer gesammelten Metrik für Elastizität erfolgt aufgrund des Vergleiches eines Vergleichssystems S_1 gegen ein Referenzsystem S_0 und den zugehörigen Messungen der Ressourcenabweichungskosten beider Systeme $C_{o+u.\ S_0}$ und $C_{o+u.\ S_1}$. Dazu werden zunächst die Kosten beider zu vergleichender Systeme aus den verschiedenen Auslastungskurven $w \in W$ in Beziehung gesetzt. Daraus ergeben sich die verhältnismäßigen Unterschiede in der Elastizität beider Systeme in Bezug auf eine bestimmte Auslastung w_x. (Islam et al., 2012)

$$E_{S_1,\ w_x} = \frac{C_{o+u.\ S_1,\ w_x}}{C_{o+u.\ S_0,\ w_x}}$$

Schlussendlich wird das geometrische Mittel aus diesen Verhältnissen errechnet, um eine konsolidierte Metrik für die Elastizität eines Vergleichssystems S_1 gegenüber einem Referenzsystem S_0 auszudrücken. (Islam et al., 2012)

$$E_{S_1} = \sqrt[n]{\prod_{x=1}^{n} (E_{S_1,\ w_x})}$$

3.3.5 Gesamtkosten

Die Elastizität eines Systems wird auf Basis der Ressourcenabweichungskosten zweier zu vergleichender Systeme errechnet. Diese Kosten entsprechen jedoch nicht den Gesamtkosten des Systems, da hierbei die Kosten der tatsächlich genutzten Ressourcen nicht berücksichtigt werden. Diese sind jedoch relevant, da die Gesamtkosten nicht nur durch die Ressourcenabweichungen, sondern auch durch die genutzten Ressourcen definiert werden. Demnach können die Gesamtkosten C(t) als die Summe der Kosten für genutzte Ressourcen $C_d(t)$, Überversorgung $C_o(t)$ und Unterversorgung $C_u(t)$ errechnet werden. Abschließend wird

ein einzelner Vergleichswert berechnet, der die durchschnittlichen Gesamtkosten pro Minute ausdrückt.

$$C(t) = C_d(t) + C_{o+u}(t)$$

$$C(t) = C_d(t) + C_o(t) + C_u(t)$$

$$C = \frac{\sum C(t)}{t_e - t_s}$$

Um die Gesamtkosten zweier Systeme über mehrere verschiedene Auslastungskurven zu vergleichen, wird auch hier das geometrische Mittel errechnet. Dadurch lässt sich der durchschnittliche prozentuelle Kostenunterschied T_{S_1} von S_1 gegenüber dem Referenzsystem S_0 ausdrücken.

$$T_{S_1,\ w_x} = \frac{C_{.S_1,\ w_x}}{C_{S_0,\ w_x}}$$

$$T_{S_1} = \sqrt[n]{\prod_{x=1}^{n} (T_{S_1,\ w_x})}$$

3.4 Messung

In den Kapiteln 3.2 und 3.3 werden bereits die für die durchgeführten Tests maßgeblichen Metriken vorgestellt. Da jedoch nicht nur die Berechnung der Metriken, sondern ebenso die Art ihrer Messung essenziell ist, werden im nachfolgenden die verwendeten Messmethoden erläutert. Im Speziellen wird dabei auf die Erhebungsmethoden von Antwortzeit sowie genutzte, verfügbare, verrechnete und benötigte Ressourcen, und daraus abgeleitet Über- und Unterversorgung, eingegangen.

3.4.1 Performanz

Die Messung der Performanz erfolgt als die Antwortzeit in Millisekunden. Es erfolgt keine separate Erhebung der Antwortzeiten pro Art der Anfrage. Demnach wird die durchschnittliche Antwortzeit $\bar{r}$ aus den Antwortzeiten aller ausgeführten Anfragen eines Tests errechnet. Die Berechnung erfolgt auf Minutenbasis, sodass der Verlauf der durchschnittlichen Antwortzeit pro Minute im Test verfolgt werden kann. Auf die gleiche Weise werden die maximale Antwortzeit sowie die 99%- und 95%-Quantile errechnet.

3.4.2 Überversorgung

Wie in Kapitel 3.3.1 beschrieben, wird Überversorgung auf Basis der folgenden Werte berechnet:

- verfügbare Ressourcen: $R(t)$
- verrechnete Ressourcen: $M(t)$
- benötigte Ressourcen: $D(t)$
- Kostenfaktor: c_o

Ressourcen werden im Rahmen der Tests in Form der vCPUs (virtuelle CPU) im System berücksichtigt. Die Messung der Auslastung erfolgt in Millicores, wobei 1000 Millicores die Kapazität einer vCPU repräsentieren.

Dementsprechend werden die verfügbaren Ressourcen $R(t)$ im System über die Anzahl der verfügbaren vCPUs zum Zeitpunkt t multipliziert mit 1000 definiert. Als verfügbare vCPUs gelten jene, die zu einer virtuellen Maschine mit dem Status „Bereit“ gehören. In weiterer Folge werden für die Messung der verrechneten Ressourcen $M(t)$ auch die vCPUs jener virtuellen Maschinen berücksichtigt, welche den Status „Nicht Bereit“ haben.

Für die Messung der benötigten Ressourcen $D(t)$ nutzen Islam et al. (2012) eine Approximation in Form der durchschnittlichen tatsächlichen CPU-Auslastung über alle bereiten virtuellen Maschinen. Im Rahmen dieser Arbeit wird dieselbe Approximation verwendet, obgleich diese, wie in Kapitel 3.4.3 erläutert, nicht optimal ist.

Alle Messwerte für die verfügbaren Ressourcen $R(t)$, die verrechneten Ressourcen $M(t)$ und die benötigten Ressourcen $D(t)$ haben die Einheit Millicores/Minute. Um die Überversorgung im System auf Basis dieser Messwerte zu errechnen, werden die benötigten Ressourcen von den verrechneten Ressourcen subtrahiert.

In der Beschreibung der Berechnung der Überversorgung in Kapitel 3.3.1 wird der Fall berücksichtigt, dass Ressourcen verrechnet werden, obwohl diese aktuell nicht verfügbar sind. Dieser Fall ist jedoch nur relevant, sofern die benötigten Ressourcen $D(t)$ über den verfügbaren Ressourcen $R(t)$ liegen. Aufgrund der Approximation für die Messung der benötigten Ressourcen kann dieser Fall nicht eintreten und muss somit nicht berücksichtigt werden. Dementsprechend kann die Überversorgung durch die einfache Subtraktion der benötigten Millicores pro Minute von den verrechneten Millicores pro Minute errechnet werden. Als Ergebnis beschreibt $O(t)$ die überversorgten Millicores im System zum Zeitpunkt t.

$$O(t) = M(t) - D(t)$$

Der Kostenfaktor c_o legt einen monetären Wert fest, mit dem ein überversorgter Millicore im System bewertet wird. In der vorliegenden Arbeit wird dieser mit dem Preis der verwendeten Microsoft Azure DS2v2 Maschinen festgesetzt, die mit zwei vCPUs

eine Kapazität von 2000 Millicores besitzen. Der Preis beträgt laut aktuellem Pricing 0,2505 € / Stunde. Um den Preis für einen Millicore in einer Minute zu berechnen, wird dieser Wert durch

60 und 2000 dividiert. Der sich dadurch ergebende Kostenfaktor für einen Millicore Überversorgung ist 0,000002087 € / Minute.

$$C_o(t) = O(t) * c_o$$

Derselbe Kostensatz von 0,000002087 € pro Millicore und Minute wird für die Berechnung der Kosten der genutzten Ressourcen $C_d(t)$ verwendet.

3.4.3 Approximation der benötigten Ressourcen

Die Approximation der benötigten Ressourcen mithilfe der tatsächlichen CPU-Auslastung besitzt einige Nachteile. Einer davon ist die Deckelung der benötigten Ressourcen mit den verfügbaren Ressourcen. Das bedeutet, dass die Kurve der benötigten Ressourcen nie über jene der verfügbaren Ressourcen steigen kann. Das entspricht jedoch nicht der Realität, da die benötigten Ressourcen auch über den verfügbaren Ressourcen liegen können, vor allem bei Zuständen der Unterversorgung. Dieser Umstand stellt laut Islam et al. (2012) kein Problem für die Messung von Elastizität dar, weil die benötigten Ressourcen ohnehin nur für die Messung der Überversorgung herangezogen werden. In Zuständen der Unterversorgung werden für die Messung die Kosten aufgrund der Überschreitung von QoS-Zielen herangezogen.

Die genannte Begründung von Islam et al. (2012) ist nachvollziehbar, allerdings nicht ganz vollständig, denn es ergeben sich dadurch auch Ungenauigkeiten in der Messung der Überversorgung. Diese begründen sich darauf, dass die durchschnittliche tatsächlich gemessene CPU-Auslastung in einem Cluster in der Praxis nur selten 100% erreicht, auch wenn das System bereits einige QoS-Ziele nicht einhalten kann und damit als unterversorgt gilt. So erreicht das System häufig schon die Grenzen seiner Kapazitäten, obwohl die CPU-Auslastung noch in einem Bereich zwischen 95% und 100% liegt. Aufgrund dieses Verhaltens sinken mit dieser Form der Messung die Kosten der Überversorgung nur selten auf null, auch wenn das System bereits an seine Grenzen stößt und die benötigten Ressourcen in Wirklichkeit höher liegen als die tatsächlich genutzten. Dadurch ergibt sich die Diskrepanz, dass die Kosten für Überversorgung meist nicht auf null sinken, obwohl gleichzeitig eine Unterversorgung gemessen wird.

Ein zweiter Nachteil, der von Islam et al. (2012) genannt wird, ist die mögliche Verzerrung von Auslastungsspitzen. So kann es passieren, dass sich die Abarbeitung vieler gleichzeitig eintreffender Anfragen verteilt und sich die Kurve der genutzten und benötigten Ressourcen nach rechts abflacht.

Des Weiteren beschreiben Islam et al. (2012) als zusätzlichen Nachteil die hohe Schwankung in der Performanz von bereitgestellten virtuellen Maschinen, wonach die gemessene CPU-Auslastung für dieselbe Workload zwischen 350% und 450% schwankt. Demnach würden die benötigten Ressourcen auf unterschiedlichen Maschinen sehr verschieden sein, sodass bei der mehrfachen Ausführung derselben Tests die Ergebnisse nicht nachgestellt werden könnten. Die Vergleichbarkeit und damit die Qualität derselben würden dadurch stark verringert. Ob diese Schwankungen der Performanz auch auf die im Rahmen dieser Arbeit verwendeten virtuellen Maschinen zutreffen, ist nicht voll und ganz auszuschließen, allerdings deuten die in Kapitel 4.1.3 vorgestellten Ergebnisse darauf hin, dass diesbezüglich keine Problematik vorliegt.

Obwohl diese Form der Messung mit den oben genannten Nachteilen nicht optimal ist, wird mangels einer besseren Messmethode dieselbe verwendet. Im Kapitel 3.5 wird ein Ansatz für eine alternative Methode zur Bestimmung der benötigten Ressourcen diskutiert, der sich allerdings nicht für die weitere Verwendung eignet.

3.4.4 Unterversorgung

Für das in Kapitel 3.3.3 beschriebene Vorgehen zur Berechnung der Unterversorgung müssen folgende Werte definiert oder gemessen werden:

- QoS-Ziel
- Unterversorgung: $U(t)$
- optimale Unterversorgung: $U_{opt}(t)$
- Kostenfunktion: $f()$

Islam et al. (2012) verwenden die QoS-Aspekte aus der Arbeit von Nah (2004) und definieren das QoS-Ziel der Performanz mit zwei Sekunden. Im Rahmen dieser Arbeit wird ein anderer Ansatz für die Festlegung des QoS-Zieles verwendet. Die Grenze der Antwortzeit für Unterversorgung wird als die durchschnittliche Antwortzeit über alle zu vergleichenden Systeme definiert. Auf diese Weise ist keine externe Definition des QoS-Zieles notwendig, sondern dieses wird aufgrund der durchschnittlichen Performanz der zu vergleichenden Systeme festgelegt. Der Vorteil dabei besteht darin, dass kein Vorwissen über die Performanz der gegenständlichen Systeme notwendig ist. Dies eliminiert die Notwendigkeit von Prä-Tests zur Erhebung einer sinnvollen QoS-Grenze, wenn solche Vorkenntnisse zu einem System nicht vorhanden sind. Beispielsweise könnte eine zu hohe QoS-Grenze in Systemen mit sehr kurzen Antwortzeiten dazu führen, dass in keinem der zu vergleichenden Systeme eine Unterversorgung gemessen wird. In so einem Fall wäre auch kein sinnvoller Vergleich basierend auf der Unterversorgung möglich.

Die Berechnung der QoS-Grenze q erfolgt in diesem Zuge aufgrund eines ungewichteten Durchschnittes der durchschnittlichen Antwortzeiten $\bar{r}$ aller zu vergleichenden Systeme mit Anzahl n. Würde man eine Gewichtung hinsichtlich der Anzahl der Anfragen verwenden, so würde sich die QoS-Grenze zum Durchschnittswert des Systems verschieben, welches die höhere Anzahl an gemessenen Anfragen in den zugrundeliegenden Messdaten besitzt. Bei einem starken Ungleichgewicht hätten Systeme mit wenigen Anfragen in den Messdaten nur geringen Einfluss auf die QoS-Grenze, sodass diese nur für stark gewichtete Systeme repräsentativ wäre. Da die Performanz aller zu vergleichenden Systemen aber zu gleichen Teilen berücksichtigt und unabhängig von der Größe der Messdatensätze sein sollen, wird ein ungewichteter Durchschnitt verwendet.

$$q = \frac{\sum_0^n \bar{r}}{n}$$

Des Weiteren würde über die externe Festlegung der QoS-Grenze ein gewisser Grad an Willkürlichkeit in die Tests eingebracht. Dadurch können sich gezielte Vorteile oder Nachteile für

bestimmte Systeme ergeben, beispielsweise wenn ein System eine bessere allgemeine Performanz gegenüber anderen hat, jedoch das 95%-Quantil höher liegt als bei den verglichenen Systemen. Das Risiko solcher versteckten Vorteile und Nachteile wird über das angewandte Vorgehen reduziert. Dies geschieht, indem Systeme mit einer besseren allgemeinen Performanz implizit eine geringere Unterversorgung haben, da die durchschnittliche Antwortzeit des performanteren Systems im Gegensatz zu dem weniger performanten System unterhalb der QoS-Grenze liegt. Bei der Festlegung und auch der Berechnung der QoS-Grenze muss beachtet werden, dass sich die Kurve der Unterversorgung immer weiter an die Kurve der Antwortzeiten angleicht, je geringer die QoS-Grenze gesetzt wird. Ist die QoS-Grenze null, so entspricht die Kurve der Unterversorgung der Kurve der Antwortzeiten. Da dies aber nicht sinnvoll ist, sollte darauf geachtet werden, dass die QoS-Grenze weder bei externer Definition noch bei der Berechnung aufgrund der durchschnittlichen Antwortzeiten zu niedrig ist.

Die Funktion $U(t)$ wird von Islam et al. (2012) als Funktion definiert, die zum Zeitpunkt t die Anzahl der Anfragen beschreibt, welche über der QoS-Grenze liegen. Im Rahmen dieser Arbeit wird für $U(t)$ nicht die Anzahl der Anfragen gemessen, sondern die Summe der Millisekunden, welche zum Zeitpunkt t über die QoS-Grenze fallen, dividiert durch die QoS-Grenze. Dafür werden zunächst alle Messwerte der im Test durchgeführten Anfragen gefiltert und nur jene berücksichtigt, welche über der QoS-Grenze liegen. Von der Antwortzeit r dieser unzufriedenstellenden Anfragen wird die QoS-Grenze q subtrahiert. Dadurch ergibt sich die Unterversorgung u in Millisekunden zu jeder einzelnen unzufriedenstellenden Anfrage $n \in N$.

$$u_n = r_n - q \mid r > q$$

In weiterer Folge werden die überschrittenen Millisekunden u_n der unzufriedenstellenden Anfragen auf Basis der Minuten in der Testlaufzeit aggregiert. Dadurch ergibt sich die Summe der die QoS-Grenze überschrittenen Millisekunden pro Minute $U_{ms}(t)$.

Abschließend wird die Summe der überschrittenen Millisekunden pro Minute $U_{ms}(t)$ durch die QoS-Grenze q dividiert. Dadurch wird die Unterversorgung auf Basis der QoS-Grenze relativiert. Das heißt, jedes Vielfache der QoS-Grenze, die eine Anfrage benötigt, wird als ein Strafpunkt für Unterversorgung bewertet. Wenn beispielsweise eine Anfrage die doppelte Dauer der QoS-Grenze benötigt, erhöht sich die gemessene Unterversorgung um 1, bei der dreifachen Dauer um 2, usw. Die Werte für die Unterversorgung werden dabei nicht diskret berechnet, sodass jede Überschreitung der QoS-Grenze die Unterversorgung mit dem entsprechenden Anteil erhöht.

$$U(t) = \frac{U_{ms}(t)}{q}$$

In der Arbeit von Islam et al. (2012) wird die optimale Unterversorgung $U_{opt}(t)$ als Baseline der Unterversorgung beschrieben, die sogar in einem optimalen System mit unbegrenzten Ressourcen auftreten würde. In der Durchführung der Fallstudie von Islam et al. (2012) wird die Verwendung solch einer Baseline-Funktion $U_{opt}(t)$ nicht mehr erwähnt. Auch im Rahmen der vorliegenden Arbeit wird auf die Verwendung einer Baseline-Funktion $U_{opt}(t)$ verzichtet. Dadurch kann die Formel für die Unterversorgung wie folgt reduziert werden:

$$C_u(t) = f(\,U(t)\,)$$

Die Kostenfunktion $f()$ ist abhängig vom Kontext und Anwendungsfall der jeweiligen Tests und kann demnach unterschiedlich für verschiedene Organisationen sein (Islam et al., 2012). Ein Ansatz ist eine lineare Kostenfunktion, welche eine bestimmte Gewichtung pro überschrittener Millisekunde definiert. Beispielsweise könnte dafür ein Vielfaches des Kostenfaktors für die Überversorgung des Systems herangezogen werden, sodass die Kosten für eine überschrittene Millisekunde doppelt so hoch gewichtet werden als die Überversorgung um einen Millicore.

Ein weiterer möglicher Ansatz für die Handhabung der Kostenfunktion ist eine exponentielle Steigerung der Kosten abhängig von der Überschreitung der QoS-Grenze, sodass leichte Überschreitungen der QoS-Grenze nur minimal berücksichtigt und schwere Überschreitungen mit höheren Kosten belegt werden. Dieses Verhalten ist auch sinnvoll in Kombination mit der Berechnung der QoS-Grenze mithilfe der durchschnittlichen Antwortzeiten über alle Testsysteme.

Im Rahmen dieser Arbeit wird eine konstante Kostenfunktion angenommen, die dem Kostenfaktor für Überversorgung c_o mit 0,000002087 € entspricht, sodass c_o gleich c_u ist. Der Kostenfaktor bezieht sich bei der Unterversorgung allerdings nicht auf einen Millicore, sondern auf eine Millisekunde.

$$f(\,U(t)\,) = U(t) * c_u$$

$$C_u(t) = f(\,U(t)\,) = \;\; U(t) * c_u$$

Durch die gleichen Kostenfaktoren für Überversorgung, Unterversorgung und genutzte Ressourcen können diese drei Metriken aufgrund der Kosten direkt miteinander verglichen werden. Bei praktischen Anwendungen dieser Methode muss im jeweiligen Kontext evaluiert werden, welcher Kostenfaktor oder welche Kostenfunktion sinnvoll sind.

3.4.5 Vergleichbarkeit der Messwerte

Bei der Berechnung der Überversorgung $O(t)$ und der Unterversorgung $U(t)$ muss beachtet werden, dass diese nicht direkt miteinander vergleichbar sind, da es sich dabei um zwei unterschiedliche Einheiten handelt. Die Überversorgung wird in Millicores/Minute ausgedrückt und die Unterversorgung in Millisekunden/Minute. Damit eine Vergleichbarkeit und damit Vereinbarkeit der beiden Metriken möglich sind, werden diese mithilfe entsprechender Kostenfaktoren oder Kostenfunktionen in Kosten umgerechnet, welche monetäre Werte darstellen. Auf diese Weise ist die weitere Berechnung einer einheitlichen Metrik für Elastizität möglich, welche ebenso als monetärer Wert ausgedrückt wird.

3.4.6 Granularität der Messungen

Im Zuge der Messungen der benötigten Metriken ergeben sich Diskrepanzen aufgrund der Granularität. Während JMeter Ergebnisse auf Basis des Unix-Zeitstempel angibt, liefert Azure-Monitor die Ergebnisse nur auf Minutenbasis.

Die CPU-Auslastung der Testsysteme ist demnach auch nur in durchschnittlichen Werten pro Minute messbar. Für die Messung von spontanen Auslastungsspitzen und kurzfristigen

Belastungen wären feinere Messintervalle vorteilhaft. Aufgrund dieser groben Daten von Azure-Monitor sind aber jegliche Angaben in den Ergebnissen sowie alle Berechnungen nur auf Minutenbasis möglich.

Aufgrund dieses Unterschiedes in der Granularität der Messungen müssen die Messdaten aus JMeter mit einer zusätzlichen Spalte angereichert werden, welche den Unix-Zeitstempel auf volle Minuten rundet. Auf Basis dieser zusätzlichen Spalte können die Messwerte zu den jeweiligen Minuten aggregiert werden. Auf diese Weise ist eine Zuordnung der JMeter Messdaten zu den Azure-Monitor Messdaten möglich. Diese wird unter anderem im Zuge der Berechnungen für die Unterversorgung benötigt, wie in Kapitel 3.4.4 beschrieben. Außerdem können die JMeter Messdaten mithilfe dieser Spalte mit weiteren Daten angereichert werden, wie beispielsweise Anzahl der Anfragen, Durchschnittswerte oder Quantile zu den Antwortzeiten auf Minutenbasis.

3.5 Alternative Methode zur Bestimmung der benötigten Ressourcen

Aufgrund der genannten Probleme mit der Messung der benötigten Ressourcen, wäre eine alternative Erhebungsform sinnvoll, welche die benötigten von den verfügbaren Ressourcen entkoppelt und die Möglichkeit gibt, eine von den verfügbaren Ressourcen unabhängige Funktion $D(t)$ für die benötigten Ressourcen zu definieren.

Ein möglicher Ansatz dazu ist die empirische Ermittlung einer Funktion, welche die veränderte Auslastung pro Workload-Einheit beschreibt. Das kann beispielsweise die CPU-Auslastung pro Anfrage sein. In der Praxis ist die zusätzliche Auslastung einer einzelnen Anfrage allerdings meist sehr klein und nicht sinnvoll messbar. Stattdessen kann als größere Workload-Einheit ein virtueller Benutzer definiert werden. Jeder virtuelle Benutzer führt einen zuvor definierten Testplan mit unterschiedlichen Arten von Anfragen aus. Zwischen jeder Anfrage ist ein konstanter Sleep-Timer von fünf Millisekunden eingebaut. Auf diese Weise ist der Test nicht von der Performanz der Maschine abhängig, auf der die Testsoftware ausgeführt wird.

In weiterer Folge werden Testsysteme bereitgestellt, welche jeweils zwischen einer und bis zu fünf virtuellen Maschinen beinhalten. Die virtuellen Maschinen für die Testsysteme werden aus einem Node-Pool von Azure DS2v2 Maschinen mit jeweils 2 vCPUs und 7 GB RAM bereitgestellt. Die Test-Software wird auf einer Azure DS4v2 Maschine mit 8 vCPUs und 28 GB RAM ausgeführt.

Auf jedem der Testsysteme werden hintereinander unterschiedliche Workload-Stufen ausgeführt. Ein zusätzlicher virtueller Benutzer entspricht dabei einer Steigerungsstufe. Jede Workload-Stufe wird für eine Dauer von drei Minuten gehalten. Die durchschnittliche CPU-Auslastung in der zweiten dieser drei Belastungsminuten wird als Auslastung für die gegebene Workload herangezogen, damit sichergestellt ist, dass eine volle Minute durchgehende Belastung in den Messwerten abgebildet ist. Aus diesen Messergebnissen wird die Differenz der CPU-Auslastung für jeden zusätzlichen virtuellen Benutzer pro virtueller Maschine errechnet.

3.5.1 Messungen der CPU-Auslastung

1 virtuelle Maschine

virtuelle Benutzer	virtuelle Benutzer/VM	CPU-Auslastung	Differenz für zus. Benutzer/VM
0	0	77,00	
1	1	406,56	329,56
2	2	687,99	281,43
3	3	937,07	249,08
4	4	1165,17	228,10
5	5	1364,66	199,49
6	6	1518,38	153,73
7	7	1506,56	-11,82
8	8	1559,96	53,40
9	9	1662,30	102,34
10	10	1839,68	177,38
11	11	1812,72	-26,95
12	12	1905,28	92,55
13	13	1903,16	-2,12
14	14	1913,68	10,52
15	15	1946,79	33,12

Tabelle 3-1 Messwerte zur CPU-Auslastung bei steigender Benutzeranzahl pro virtueller Maschine in einem Testsystem mit einer virtuellen Maschine (Quelle: eigene Messwerte)

2 virtuelle Maschinen

virtuelle Benutzer	virtuelle Benutzer/VM	CPU-Auslastung	Differenz für zus. Benutzer/VM
0	0	80,00	
2	1	376,15	296,15
4	2	604,60	228,44
6	3	809,51	204,91
8	4	997,47	187,96
10	5	1161,77	164,30
12	6	1330,17	168,40
14	7	1465,24	135,07
16	8	1570,08	104,84
18	9	1578,63	8,55
20	10	1684,43	105,80
22	11	1739,86	55,43
24	12	1780,06	40,20
26	13	1826,45	46,39
28	14	1865,65	39,20
30	15	1857,74	-7,91

Tabelle 3-2 Messwerte zur CPU-Auslastung bei steigender Benutzeranzahl pro virtueller Maschine in einem Testsystem mit zwei virtuellen Maschinen (Quelle: eigene Messwerte)

3 virtuelle Maschinen

virtuelle Benutzer	virtuelle Benutzer/VM	CPU-Auslastung	Differenz für zus. Benutzer/VM
0	0	77,00	
3	1	371,41	294,41
6	2	608,53	237,12
9	3	783,71	175,18
12	4	994,96	211,25
15	5	1089,84	94,88
18	6	1313,98	224,13
21	7	1436,82	122,85
24	8	1413,85	-22,98
27	9	1565,78	151,94
30	10	1644,86	79,08
33	11	1696,93	52,07
36	12	1731,21	34,28
39	13	1752,00	20,79
42	14	1785,68	33,68
45	15	1701,05	-84,63

Tabelle 3-3 Messwerte zur CPU-Auslastung bei steigender Benutzeranzahl pro virtueller Maschine in einem Testsystem mit drei virtuellen Maschinen (Quelle: eigene Messwerte)

4 virtuelle Maschinen

virtuelle Benutzer	virtuelle Benutzer/VM	CPU-Auslastung	Differenz für zus. Benutzer/VM
0	0	72,00	
4	1	382,01	310,01
8	2	634,32	252,31
12	3	829,41	195,09
16	4	1022,39	192,98
20	5	1060,49	38,10
24	6	1248,42	187,93
28	7	1402,13	153,71
32	8	1479,58	77,46
36	9	1513,10	33,51
40	10	1550,17	37,07
44	11	1582,69	32,52
48	12	1650,17	67,48
52	13	1662,04	11,87
56	14	1767,86	105,82
60	15	1814,75	46,89

Tabelle 3-4 Messwerte zur CPU-Auslastung bei steigender Benutzeranzahl pro virtueller Maschine in einem Testsystem mit vier virtuellen Maschinen (Quelle: eigene Messwerte)

5 virtuelle Maschinen

virtuelle Benutzer	virtuelle Benutzer/VM	CPU-Auslastung	Differenz für zus. Benutzer/VM
0	0	75,00	
5	1	363,88	288,88
10	2	585,86	221,98
15	3	769,97	184,11
20	4	940,25	170,27
25	5	1116,66	176,41
30	6	1256,17	139,51
35	7	1346,15	89,99
40	8	1422,74	76,59
45	9	1435,35	12,61
50	10	1539,71	104,36
55	11	1568,11	28,40
60	12	1672,34	104,23
65	13	1660,32	-12,02
70	14	1693,90	33,57
75	15	1767,31	73,41

Tabelle 3-5 Messwerte zur CPU-Auslastung bei steigender Benutzeranzahl pro virtueller Maschine in einem Testsystem mit fünf virtuellen Maschinen (Quelle: eigene Messwerte)

Bei den Messwerten ist auffällig, dass die Auslastung im Bereich niedriger Benutzerzahlen vergleichsweise stark durch einen zusätzlichen Benutzer steigt. Bei höherer Benutzeranzahl flacht sich die Kurve immer mehr ab. Ab ungefähr zehn gleichzeitigen Benutzern pro virtueller Maschine ist nur noch ein schwacher Anstieg pro zusätzlichem Benutzer zu verzeichnen.

Des Weiteren ist erkennbar, dass die durchschnittliche Auslastung bei höherer Anzahl der virtuellen Maschinen tendenziell niedriger ist. In Abbildung 3-1 ist erkennbar, dass die Auslastung mit einer virtuellen Maschine durchschnittlich am höchsten ist im Vergleich zu den anderen Testsystemen, obwohl in allen Tests und Workload-Stufen dieselbe Anzahl an virtuellen Benutzern pro virtuelle Maschine verwendet wird. Im Testsystem mit zehn virtuellen Maschinen ist die durchschnittliche CPU-Auslastung deutlich unterhalb der anderen, deshalb werden die Messwerte für das Testsystem mit zehn virtuellen Maschinen in der weiteren Berechnung der Regressionsfunktion nicht berücksichtigt.

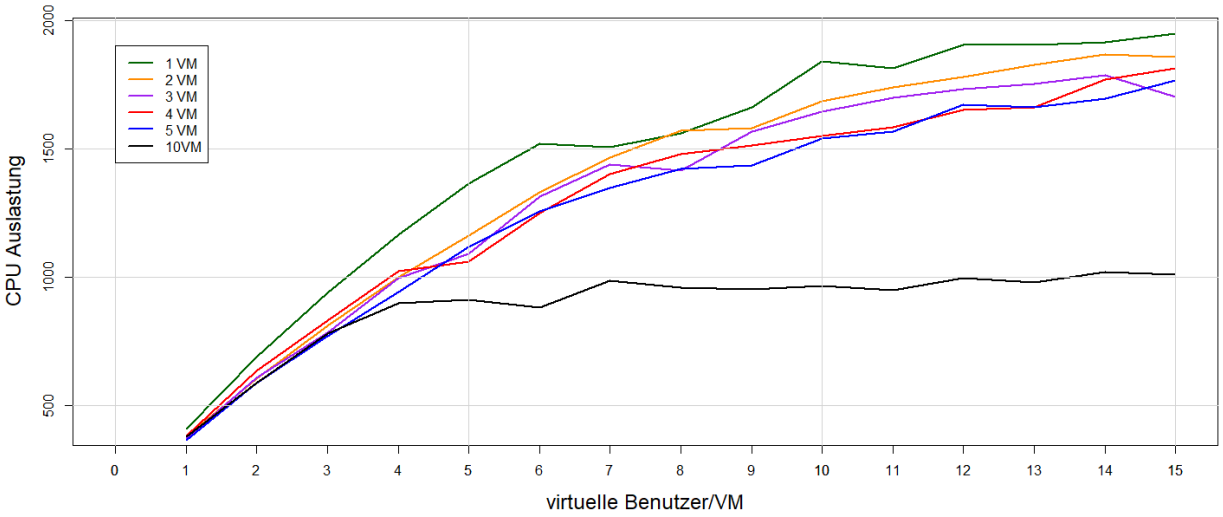


Abbildung 3-1 Verlauf der CPU-Auslastung auf steigender Anzahl der virtuellen Benutzer pro virtueller Maschine (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

3.5.2 Quadratische Regression

Basierend auf den Messwerten der Differenzen der CPU-Auslastung zwischen zwei Workload-Stufen für die Testsysteme mit einer bis fünf virtuellen Maschinen wird die weitere Berechnung vorgenommen. Dafür werden die Messwerte in einem linearen Regressionsmodell mit einer linearen und einer quadratischen Variable verwendet, um eine Regressionsfunktion zu erstellen. Die sich daraus ergebene Regressionsfunktion abhängig von der Benutzeranzahl lautet:

$$\Delta D(u) = -40{,}798\,u + 1.387u^2 + 327{,}777$$

Diese Regressionsfunktion ist in Abbildung 3-2 dargestellt und zeigt die zusätzliche CPU-Auslastung für jeden zusätzlichen Benutzer pro virtuelle Maschine.

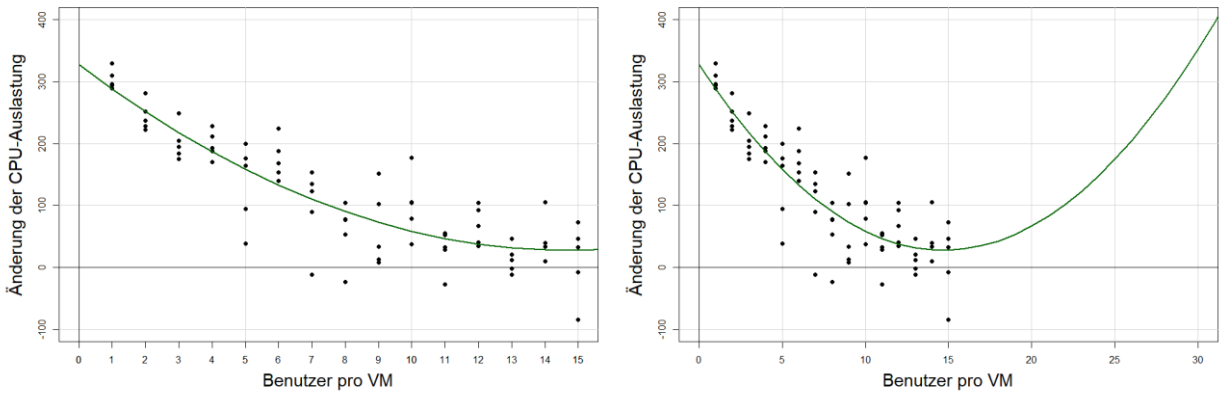


Abbildung 3-2 quadratische Regressionsfunktion über die Änderung der CPU-Millicores abhängig von der Anzahl der Benutzer / Anzahl der virtuellen Maschinen (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

Diese quadratische Regressionsfunktion hat jedoch Ungenauigkeiten und ist deshalb nicht optimal. Aufgrund der in Kapitel 3.5.1 beschriebenen Messwerte, in denen Testsysteme mit mehr virtuellen Maschinen tendenziell niedrigere CPU-Auslastungen haben, ergibt sich in der Regressionsfunktion eine Unterschätzung der benötigten Ressourcen für Testsysteme mit wenigen virtuellen Maschinen und eine Überschätzung der benötigten Ressourcen in Testsystemen mit höherer Anzahl an virtuellen Maschinen. Diese Unter- und Überschätzungen sind gut in Abbildung 3-3 erkennbar.

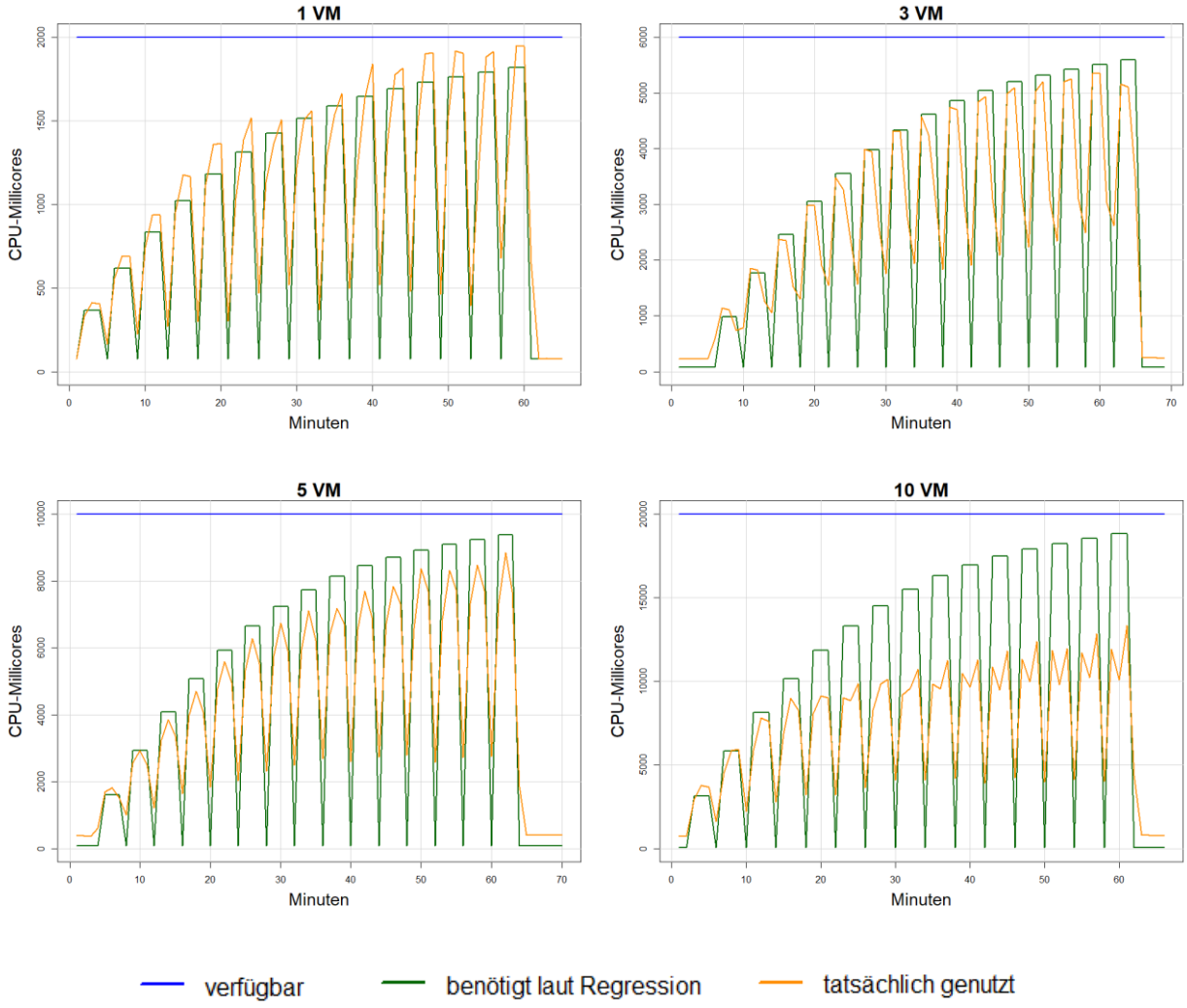


Abbildung 3-3 Vergleich von verfügbaren Ressourcen, laut quadratischer Regression benötigten Ressourcen und tatsächlich genutzten Ressourcen in den Testdurchführungen (Quelle: eigene Darstellung mit RStudio 1.1.383)

Das zweite und größere Problem der Regressionsfunktion ist ihr eingeschränkter Gültigkeitsbereich. Die quadratische Regressionsfunktion hat bereits bei 15 gleichzeitigen virtuellen Benutzern pro virtuelle Maschine ihren Tiefpunkt, danach steigt die Auswirkung auf die CPU-Auslastung mit der Benutzeranzahl exponentiell an. Dieses Verhalten würde vermutlich ab einem gewissen Punkt der Überlastung in einem System Sinn machen. Allerdings ist dieser Punkt noch nicht bei 15 virtuellen Benutzern pro virtueller Maschine erreicht. Wie in den Ergebnissen der Elastizitätstests in Kapitel 4 erkennbar ist, werden diese mit 50 bis 80 gleichzeitigen

Benutzern pro virtuelle Maschine durchgeführt. Bei 15 gleichzeitigen virtuellen Benutzern pro virtueller Maschine wird annähernd keine Unterversorgung festgestellt.

3.5.3 Logarithmische Regression

Als Alternative zur quadratischen Regressionsfunktion kann auch eine logarithmische Regressionsfunktion mit einer linearen und einer logarithmischen Variable verwendet werden. Die daraus resultierende Regressionsfunktion lautet:

$$\Delta D(u) = -4u - 88{,}87 * \log(u) + 313{,}38$$

Diese Regressionsfunktion wird in Abbildung 3-4 dargestellt und zeigt einen stetig sinkenden Verlauf der Auswirkung auf die CPU-Auslastung.

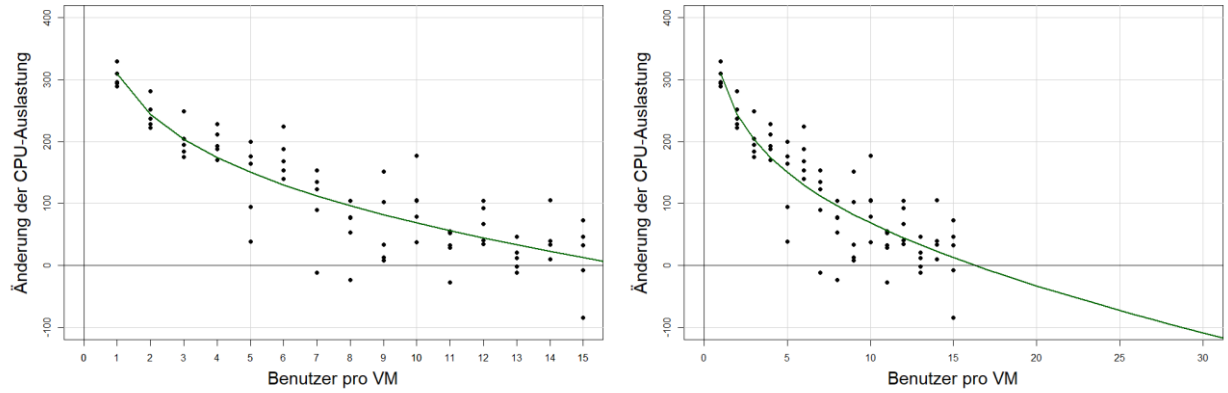
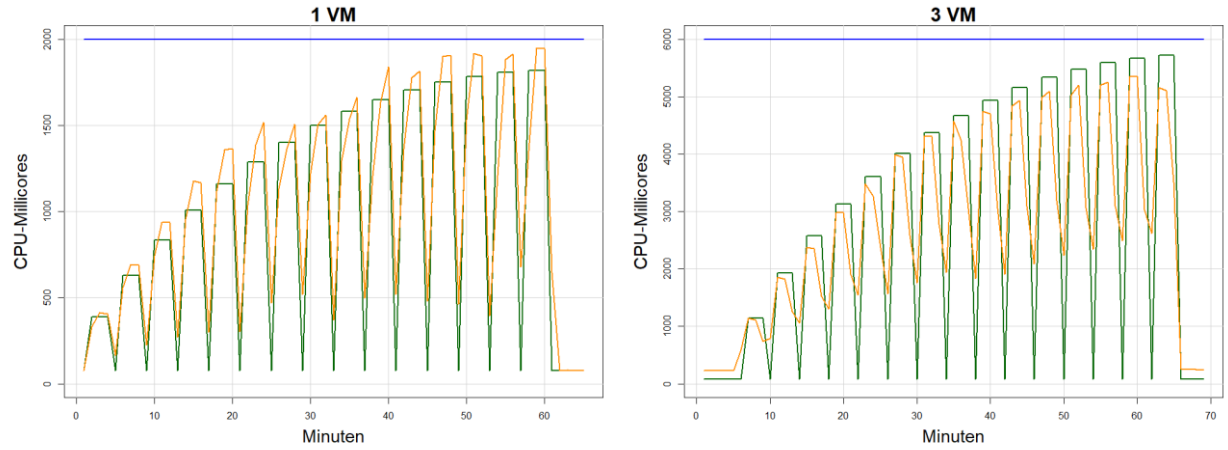


Abbildung 3-4 logarithmische Regressionsfunktion über die Änderung der CPU-Millicores abhängig von der Anzahl der Benutzer / Anzahl der virtuellen Maschinen (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

Insgesamt liefert die logarithmische Regressionsfunktion in den Vorhersagen der benötigten Ressourcen sehr ähnliche Ergebnisse wie jene der quadratischen Regressionsfunktion, jedoch sind diese marginal höher.



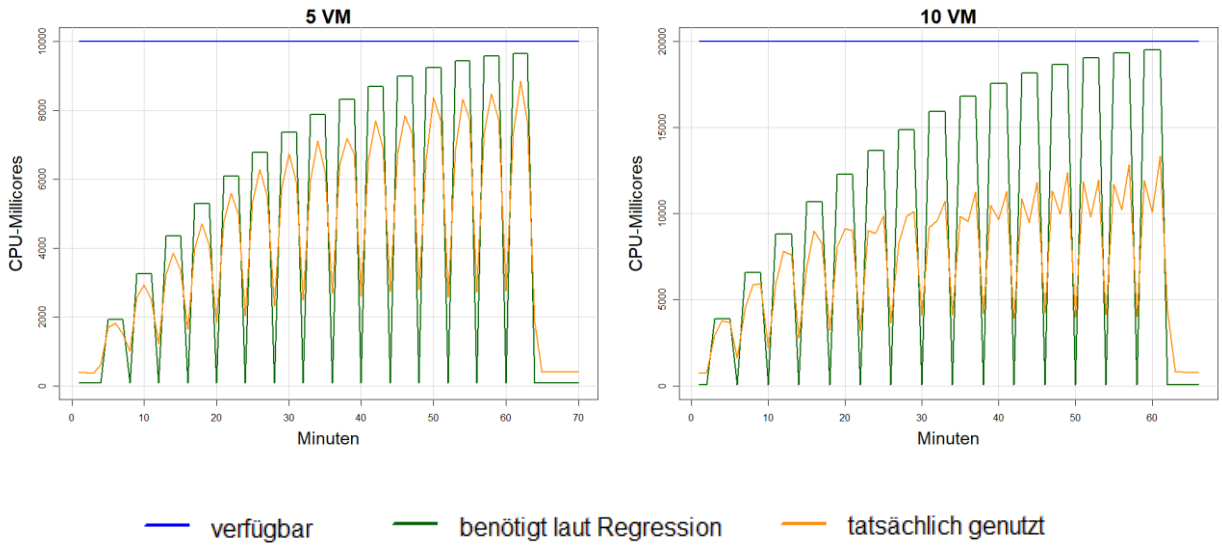


Abbildung 3-5 Vergleich der verfügbaren Ressourcen, laut logarithmischer Regression benötigten Ressourcen und tatsächlich genutzten Ressourcen in den Testdurchführungen (Quelle: eigene Darstellung mit RStudio 1.1.383)

Die logarithmische Regressionsfunktion besitzt dieselben grundlegenden Probleme wie die quadratische Regressionsfunktion. Einerseits werden die benötigten Ressourcen bei Testsystemen mit wenigen virtuellen Maschinen tendenziell unterschätzt, andererseits bei mehreren virtuellen Maschinen überschätzt. Die vorhergesagten benötigten Ressourcen liegen mit der logarithmischen Regressionsfunktion allgemein um ca. zwei bis vier Prozent höher als bei der quadratischen.

Ebenso ist der eingeschränkte Gültigkeitsbereich der logarithmischen Regressionsfunktion ein Problem. Wie in Abbildung 3-4 erkennbar, fällt die Änderung der CPU-Millicores bereits bei ungefähr 16 virtuellen Benutzern pro virtueller Maschine in den negativen Bereich. Dieser Verlauf der Funktion ist nicht sinnvoll, da eine Reduzierung der benötigten Ressourcen bei steigender Workload in der Realität nicht zu erwarten ist.

3.5.4 Bewertung

Aufgrund der bereits in Kapitel 3.5.2 und 3.5.3 genannten Probleme ist die Verwendbarkeit der beiden erstellen Regressionsfunktionen sehr eingeschränkt. Die wesentlichste Einschränkung stellt dabei der begrenzte Gültigkeitsbereich in Bezug auf die Benutzeranzahl dar. Dieses Problem ist darauf zurückzuführen, dass die CPU-Auslastung der Testsysteme bereits ab ungefähr zehn Benutzern pro virtueller Maschine einen relativ hohen Wert erreicht und danach nicht mehr wesentlich weiter steigt. Aufgrund dessen wurde bei der Durchführung dieser Tests für die Erstellung der Regressionsfunktionen davon ausgegangen, dass mit einer Anzahl von 15 virtuellen Benutzern pro virtuelle Maschine die Grenzen des Systems erreicht werden, und darüber bereits Zustände von Unterversorgung auftreten würden. In den späteren Ergebnissen der Elastizitätstests in Kapitel 4 zeigt sich jedoch, dass die Testsysteme in der Lage sind, viel höhere Workloads zu verarbeiten, ohne in einen Zustand der Unterversorgung zu kommen.

Um diesen alternativen Ansatz für die Berechnung der benötigten Ressourcen zu verwenden, werden präzisere Regressionsfunktionen benötigt. Diese können vermutlich erstellt werden, indem umfangreichere Tests zum Erhalt weitreichenderer Messwerte durchgeführt werden, welche mehrere Ausbaustufen des Testsystems sowie höhere Workloads abdecken. Des Weiteren kann die Regressionsfunktion um eine zusätzliche Variable für die Anzahl der virtuellen Maschinen im System erweitert werden. Damit können vermutlich die Überschätzungen und Unterschätzungen der benötigten Ressourcen bei unterschiedlichen Ausbaustufen ausgemerzt werden. Diese Ansätze zur Verbesserung der Regressionsfunktion werden im Rahmen der vorliegenden Arbeit allerdings nicht weiter verfolgt. Die Möglichkeiten für eine bessere Bestimmung der benötigten Ressourcen sind Gegenstand für zukünftige Arbeiten.

Außerdem ist im gleichen Zuge auch das Kosten-Nutzen-Verhältnis für die Erstellung dieser Regressionsfunktion zu hinterfragen, da die zeitlichen und finanziellen Aufwände für solch umfangreiche Tests nicht unerheblich sind. Des Weiteren ist zu beachten, dass eine erstellte Regressionsfunktion nur für das jeweilige System in unterschiedlichen Ausbaustufen gilt. Sollen die benötigten Ressourcen für ein anderes System bestimmt werden, muss eine neue Regressionsfunktion auf Basis von neuen Messwerten ermittelt werden. Dementsprechend ist dieser Ansatz nur sinnvoll, wenn eine mehrfache und längerfristige Nutzung der erarbeiteten Regressionsfunktion für dasselbe System absehbar ist.

Angesichts der aufgeführten Umstände wird die beschriebene alternative Methode für die Bestimmung der benötigten Ressourcen im Rahmen dieser Arbeit nicht verwendet. Stattdessen werden die benötigten Ressourcen über die tatsächlich genutzten Ressourcen approximiert, wie in Kapitel 3.4.2 beschrieben.

3.5.5 Erkenntnisse

Trotz der mangelnden Anwendbarkeit der erstellten Regressionsfunktionen geben die in diesem Kapitel beschriebenen Ergebnisse Aufschluss über das auffällige Verhalten der CPU-Auslastung in den Testsystemen bei steigender Workload. Daraus ist nämlich erkennbar, dass die Änderung der Workload bei sehr niedriger CPU-Auslastung eine große Auswirkung hat und bei höherer CPU-Auslastung die Auswirkungen immer kleiner werden.

Eine mögliche Begründung dafür ist der Overhead der im Testsystem genutzten Technologien. So würde bei der Verarbeitung der Anfragen eines einzelnen Benutzers ein großer Anteil der CPU-Auslastung auf den System-Overhead entfallen und ein vergleichsweise kleiner Teil aufgrund der tatsächlichen Benutzer-Anfragen entstehen. Wenn die Workload um weitere virtuelle Benutzer steigt, steigt zwar auch die Auslastung aufgrund der Benutzer-Anfragen, jedoch steigt die Auslastung aufgrund des System-Overhead nicht mehr wesentlich an.

Eine andere mögliche Begründung könnte das generelle Verhalten der verwendeten virtuellen Maschinen sein, sodass diese eher dazu neigen, nicht verwendete Ressourcen auch bereits bei niedriger Workload in Gebrauch zu nehmen. Dadurch steigt die durchschnittliche CPU-Auslastung relativ schnell auf Werte über 80 bis 90 Prozent, obwohl die tatsächliche Kapazität des Systems noch lange nicht erreicht ist. Infolgedessen führt eine weiter steigende Workload zu

einer effizienteren Nutzung der CPU-Ressourcen. Dadurch ist es möglich, dass die Testsysteme trotz bereits hoher CPU-Auslastung in der Lage sind, zusätzliche Workloads zu verarbeiten.

3.6 Testsysteme

Für die Durchführung der Tests werden zwei getrennte Testsysteme bereitgestellt. Das erste System ist statisch konfiguriert und hat keine Skalierungsfunktionen. Dieses statische System stellt das Referenzsystem S_0 dar. Das zweite Testsystem entspricht grundlegend dem statischen System, allerdings besitzt dieses elastische Eigenschaften und kann während der Laufzeit abhängig von der Auslastung skalieren. Das elastische System stellt das Vergleichssystem S_1 dar. Beide Systeme werden für jeden Test parallel in zwei eigenen, neu bereitgestellten Kubernetes-Clustern gestartet. Dadurch wird sichergestellt, dass diese vollkommen unabhängig voneinander laufen.

3.6.1 Testapplikation

Die in dieser Arbeit verwendete Testapplikation basiert auf jener, die auch in der vorangegangenen Arbeit verwendet wurde. Diese hat den Zweck, ein einfaches Webshop-System zu simulieren. Dazu beinhaltet die Testapplikation einen User-Service, einen Product-Service und einen Cart-Service. (Winkler, 2019)

Der User-Service bietet Endpunkte für das Registrieren, Einloggen und Abfragen eines Benutzers. Über den Product-Service sind Funktionen zum Anlegen, Bearbeiten, Abfragen und Suchen von Produkten verfügbar. Der Cart-Service bildet die Funktionen des Einkaufswagens eines Benutzers mit Service-Endpunkten zum Abfragen des Einkaufswagens eines Benutzers sowie für das Hinzufügen eines Produktes zum Einkaufswagen ab. In allen Funktionen der drei Services werden standardmäßige Prüfungen sowie Datenbankzugriffe auf die darunterliegende MongoDB-Instanz durchgeführt und entsprechende Antworten zurückgeliefert (Winkler, 2019). Im Vergleich zu der verwendeten Testapplikation in der vorangegangenen Arbeit wurde diese für die Verwendung in dieser Arbeit leicht adaptiert, um die Mandantenfähigkeit im Testsystem abbilden zu können.

3.6.2 Abbildung von Mandanten

Die Testapplikation wird gemeinsam mit der zugehörigen MongoDB in einem Docker-Image gebündelt, sodass die Bereitstellung mehrerer Instanzen der Testapplikation problemlos möglich ist. Das bedeutet jedoch auch, dass jede Applikationsinstanz auf Basis desselben Docker-Images erstellt wird und somit vollkommen identisch zu anderen Instanzen ist, unabhängig davon, welchem Mandanten diese zugehört.

Um nun diese identen Applikationsinstanzen zu Mandanten zuzuordnen, wird die Konfiguration des Kubernetes-Clusters verwendet. Innerhalb dieser Konfiguration besteht die Möglichkeit, sogenannte Deployments zu definieren. Jedes Deployment besteht aus einem oder mehreren

Pods. Ein Pod kann wiederum in diesem Kontext als eine Container-Instanz verstanden werden und stellt daher genau eine Instanz der Testapplikation inklusive MongoDB dar. In weiterer Folge wird für jeden Mandanten im Testsystem ein Deployment definiert. Alle Pods innerhalb desselben Deployments werden mit einem entsprechenden Label versehen, welches die Zugehörigkeit der Applikationsinstanz zu einem bestimmten Mandanten ausweist.

In den abgebildeten Testsystemen ist durch dieses Vorgehen eine getrennte Datenbasis pro Applikationsinstanz vorhanden, da jede Applikationsinstanz gemeinsam mit einer eigenen MongoDB-Instanz im selben Container ausgeführt wird und die Daten direkt in der jeweiligen Container-Instanz gespeichert wird. Dies ist nicht optimal, da in einer realen Anwendung im Regelfall alle Applikationsinstanzen eines Mandanten auf dieselbe Datenbasis zugreifen. Wie in Kapitel 2.4.4 beschrieben, wäre ein möglicher Ansatz die Speicherung der Daten pro Mandanten in einem Verzeichnis auf einem zentralen Fileserver. Dieses zentrale Verzeichnis kann pro Mandanten für jede Applikationsinstanz eingebunden werden, sodass jede MongoDB-Instanz desselben Mandanten auf die gleichen Daten zurückgreift. Dabei ist jedoch zu prüfen, ob es durch den simultanen Zugriff mehrerer MongoDB-Instanzen auf dieselben Daten zu Problemen kommen kann. Sollte dies der Fall sein, wäre die Implementierung eines vollständigen MongoDB-Clusters pro Mandanten notwendig. Im Rahmen dieser Arbeit wird als Approximation eine eigene MongoDB-Instanz mit separaten Daten innerhalb der jeweiligen Container-Instanz verwendet.

3.6.3 Testsystem Statisch

Im statischen Testsystem wird eine fest definierte Anzahl von virtuellen Maschinen sowie eine statische Anzahl von Applikationsinstanzen pro Mandanten bereitgestellt. Die eingehenden Anfragen werden von einem Ingress-Controller durch Sharding an die Load-Balancer der entsprechenden Mandanten weitergeleitet. Jeder Mandant hat einen eigenen internen Load-Balancer, welcher die Anfragen automatisch auf alle verfügbaren Container-Instanzen des jeweiligen Mandanten verteilt. Jeder Mandant wird über ein Deployment bereitgestellt. Innerhalb des Deployments sind alle Applikationsinstanzen dem jeweiligen Mandanten zugeordnet. Die Anzahl der Applikationsinstanzen pro Mandanten wird im statischen Testsystem immer auf drei Instanzen festgelegt. Dieser logische Aufbau des statischen Testsystems ist in Abbildung 3-6 dargestellt.

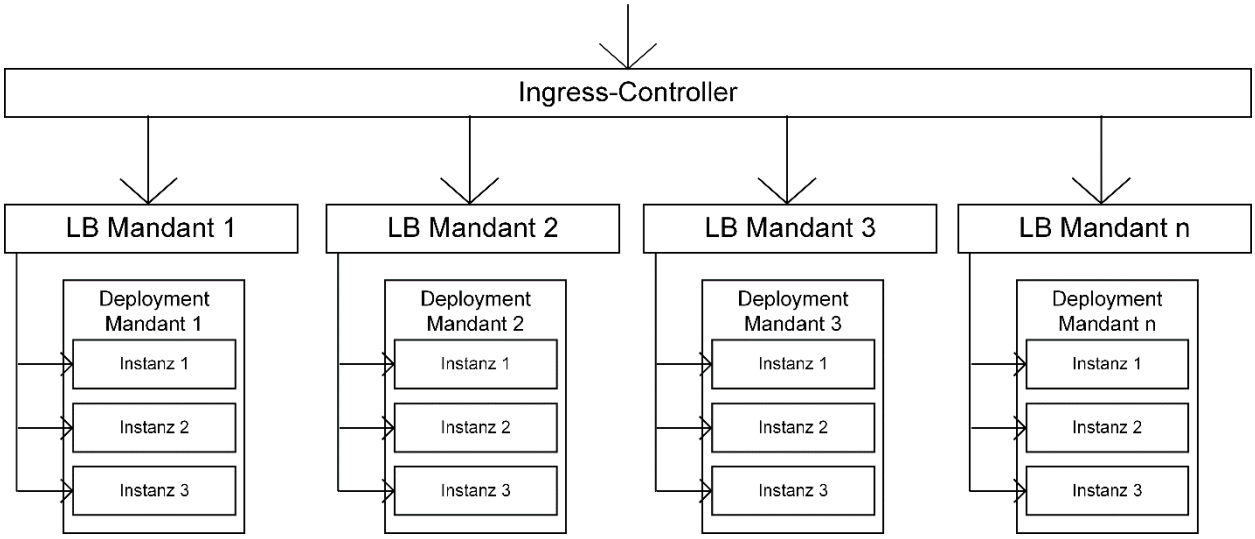


Abbildung 3-6 logischer Aufbau des statischen Testsystems (Quelle: eigene Darstellung)

Wie in Abbildung 3-7 abgebildet, erfolgt die Bereitstellung des Ingress-Controller und der Load-Balancer sowie weiterer Kubernetes-System-Services getrennt von den eigentlichen Applikationsinstanzen auf einer dedizierten virtuellen Maschine. Die Applikationsinstanzen der Mandanten werden innerhalb des separaten Mandanten-Node-Pools auf dafür vorgesehene virtuelle Maschinen bereitgestellt.

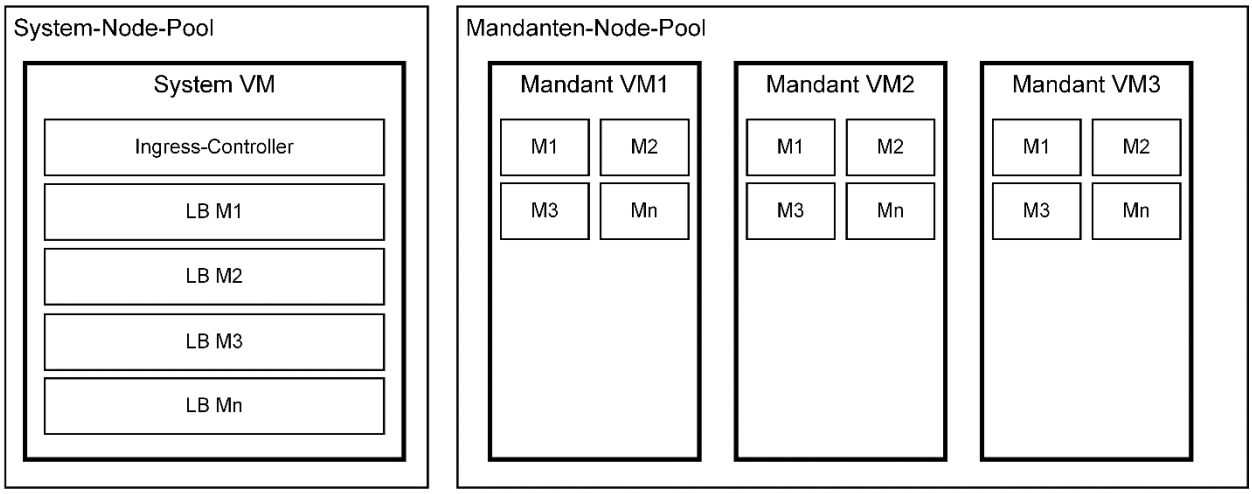


Abbildung 3-7 Verteilung der Services auf die virtuellen Maschinen im statischen Testsystem (Quelle: eigene Darstellung)

3.6.4 Testsystem Elastisch

Das elastische Testsystem ist grundlegend gleich aufgebaut wie das statische Testsystem. Es wird allerdings um die notwendigen Komponenten zur Abbildung des elastischen Verhaltens erweitert. Dies wird durch die Verwendung der Horizontal Pod Autoscaler (HPA) erreicht, welche die Anzahl der Applikationsinstanzen eines Mandanten auf Basis der durchschnittlichen CPU-Auslastung der bestehenden Applikationsinstanzen anpassen. Steigt die durchschnittliche CPU-Auslastung aller Applikationsinstanzen eines Mandanten über einen bestimmten Wert, werden zusätzliche Instanzen für diesen Mandanten gestartet. Sinkt hingegen die durchschnittliche

Auslastung der Applikationsinstanzen eines Mandanten, können die HPA die Anzahl der Applikationsinstanzen eines Mandanten bis auf eine Instanz reduzieren. Die Eingliederung der HPA in den logischen Aufbau des Testsystems ist in der Abbildung 3-8 dargestellt.

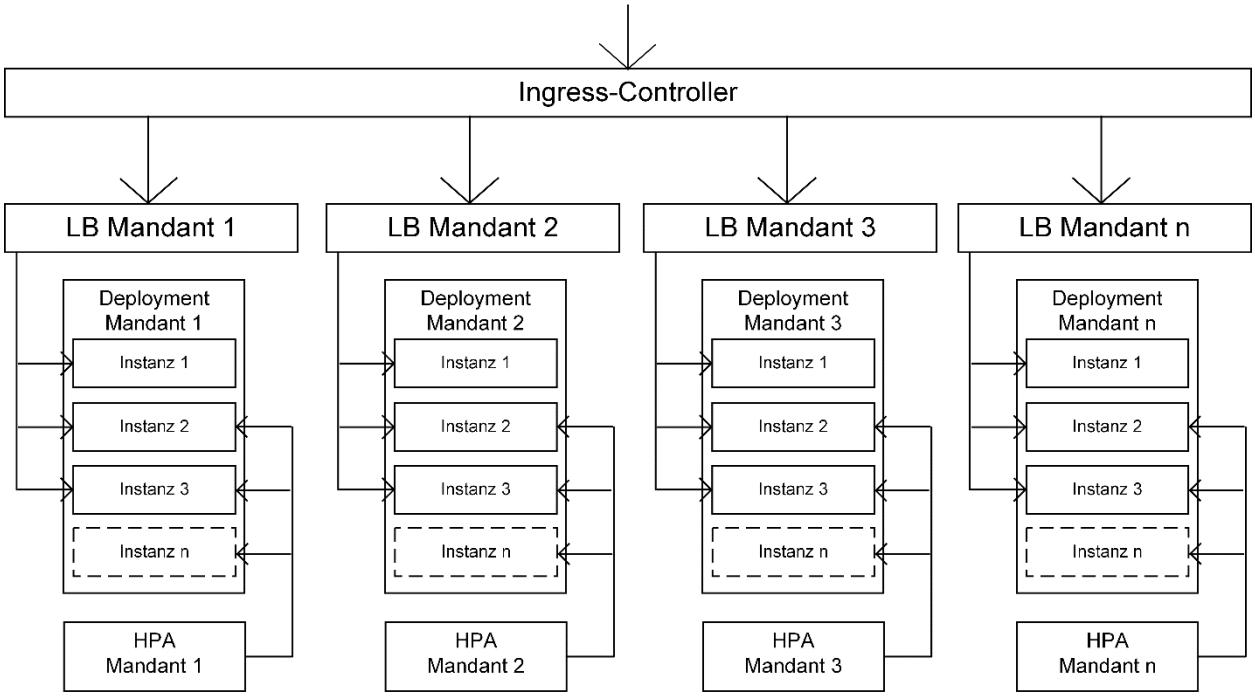


Abbildung 3-8 logischer Aufbau des elastischen Testsystems (Quelle: eigene Darstellung)

Wie im statischen Testsystem wird auch im elastischen Testsystem eine dedizierte virtuelle Maschine für die Kubernetes-System-Services sowie Ingress-Controller und Load-Balancer bereitgestellt. Zusätzlich werden auf dieser virtuellen Maschine im elastischen System auch die HPA bereitgestellt.

Des Weiteren wird der Mandanten-Node-Pool im elastischen System mit einer variablen Anzahl von virtuellen Maschinen konfiguriert. Dadurch kann der Mandanten-Node-Pool auf sich ändernde Ressourcenanforderungen reagieren und abhängig davon automatisch skalieren. Es gibt dabei keine feste Zuordnung von Mandanten zu virtuellen Maschinen. Die Auswahl der virtuellen Maschine für die Bereitstellung von neuen Applikationsinstanzen eines Mandanten trifft Kubernetes unter Berücksichtigung einer möglichst gleichmäßigen Verteilung.

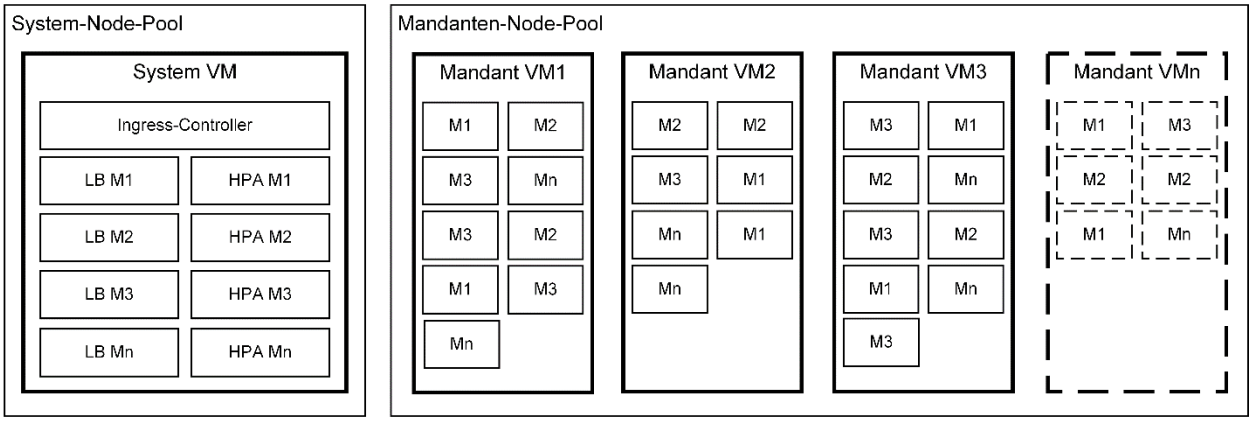


Abbildung 3-9 Verteilung der Services auf die virtuellen Maschinen im elastischen Testsystem (Quelle: eigene Darstellung)

3.6.5 Skalierungskonzept des elastischen Systems

Um die automatische Skalierung im elastischen Testsystem umzusetzen, muss ein entsprechendes Skalierungskonzept festgelegt werden, welches die Bedingungen und Schwellwerte für die Entscheidungsprozesse der Skalierung definiert. Des Weiteren muss, wie in Kapitel 2.5.3 beschrieben, zwischen der Skalierung auf Infrastruktur-Ebene und Software-Ebene unterschieden werden. Die technische Ausführung des vorgegebenen Konzeptes erfolgt im verwendeten Testsystem durch die HPA sowie durch dem Cluster-Autoscaler. Die HPA beziehen sich dabei auf die Skalierung auf Software-Ebene und der Cluster-Autoscaler auf Infrastruktur-Ebene.

Als Voraussetzung für die Konfiguration des elastischen Verhaltens müssen die Pods im Kubernetes-Cluster mit einer minimalen Ressourcenanforderung konfiguriert werden. Diese legt fest, wieviel Ressourcen auf einer virtuellen Maschine im Mandanten-Node-Pool verfügbar sein müssen, damit ein Pod auf dieser Maschine für die Ausführung eingeplant wird. Für die Pods der Applikationsinstanzen im Testsystem wird eine minimale Ressourcenanforderung von 100 Millicores und 128 MB RAM festgelegt. Dies bedeutet jedoch nicht, dass eine Instanz diese Menge an Ressourcen auch tatsächlich laufend in Anspruch nimmt, sondern dass sie für diese Instanz zugesichert sind. Beispielsweise werden 20 Applikationsinstanzen mit einer Ressourcenanforderung von je 100 Millicores auf einer virtuellen Maschine mit einer Kapazität von 2000 Millicores ausgeführt, jedoch liegt die durchschnittliche Auslastung der einzelnen Instanzen nur bei 50 Millicores. In so einem Fall hat die virtuelle Maschine 1000 Millicores an ungenutzten Kapazitäten, aber es werden auf dieser Maschine keine weiteren Instanzen mehr eingeplant. (Kubernetes, 2020a)

Im nächsten Schritt wird die Konfiguration der HPA vorgenommen, in der eine prozentuale Ziel-Auslastung der Applikationsinstanzen auf Basis der zuvor definierten Ressourcenanforderung festgelegt wird. Die Ziel-Auslastung wird mit 100% festgelegt, sodass die Ressourcenanforderung von 100 Millicores gleichzeitig dem Auslastungsziel entspricht. In weiterer Folge wird bei jeder Prüfung des HPA die Soll-Anzahl der Replikate anhand der aktuellen Anzahl der Replikate und des Verhältnisses zwischen tatsächlicher Auslastung und Ziel-Auslastung errechnet. (Kubernetes, 2020b)

$$\#ReplikateNeu = ceil\left[\#ReplikateAktuell * \left(\frac{tats\ddot{a}chlicheAuslastung}{ZielAuslastung}\right)\right]$$

In jeder Iteration des HPA wird auf diese Weise die neue Anzahl der Applikationsinstanzen des zugehörigen Mandanten ermittlet. Ergibt sich eine höhere Anzahl von Replikaten, werden die zusätzlichen Instanzen über den Kubernetes-Scheduler eingeplant. Bei einer Reduktion der Anzahl werden überflüssige Instanzen terminiert. (Kubernetes, 2020b)

Bei steigender Workload können früher oder später zusätzliche Applikationsinstanzen nicht mehr eingeplant werden, wenn alle im System verfügbaren Ressourcen genutzt oder durch Ressourcenanforderungen zugesichert sind. In so einem Fall werden die neuen Applikationsinstanzen vom Kubernetes-Scheduler mit dem Status „Pendent“ erstellt, aber auf keiner bestehenden virtuellen Maschine zur Ausführung eingeplant. Ab diesem Zeitpunkt ist die

Skalierung auf Infrastruktur-Ebene notwendig, um weitere physische Ressourcen im System bereitzustellen.

Dementsprechend läuft parallel zu den HPA der Cluster-Autoscaler und prüft laufend auf pendente Applikationsinstanzen, die nicht eingeplant werden können. Sind solche Instanzen vorhanden, werden vom Cluster-Autoscaler zusätzliche virtuelle Maschinen aus dem Mandanten-Node-Pool angefragt. Sobald diese neuen virtuellen Maschinen bereit sind, werden die pendenten Applikationsinstanzen darauf zur Ausführung eingeplant. (Microsoft Azure Documentation, 2020)

Um überflüssige virtuelle Maschinen aus dem Mandanten-Node-Pool wieder freizugeben, wird in der Cluster-Autoscaler Konfiguration ein Schwellwert der CPU-Auslastung zur Freigabe einer virtuellen Maschine gesetzt. Wird dieser Schwellwert von 70% unterschritten, so werden die übrigen Applikationsinstanzen nach Möglichkeit auf anderen Maschinen eingeplant. Sind keine Applikationsinstanzen mehr auf der virtuellen Maschine vorhanden und ist der Auslastungsschwellwert unterschritten, wird diese heruntergefahren. (Microsoft Azure Documentation, 2020)

Die Standard-Konfiguration des Cluster-Autoscaler wird im Zuge der Tests mit sensitiveren Werten angepasst, sodass die Reaktionszeiten verkürzt werden und sich ein spontaneres Skalierungsverhalten einstellt. Dies wird mit den Erkenntnissen von Islam et al. (2012) begründet, nach denen die gemessene Elastizität durch aggressives Hochskalieren und konservatives Freigeben von Ressourcen erzielt wird. Die Reaktionszeiten für die Freigabe von Ressourcen werden trotzdem auch zu einem Teil reduziert, um im kurzfristigen Rahmen der durchgeführten Tests auch die Freigabe von Ressourcen zu erreichen.

Aufgrund dieser Funktionsweise der Skalierung über HPA und Cluster-Autoscaler muss besonders auf die sinnvolle Auswahl der Ressourcenanforderungen und Skalierungsparameter geachtet werden. Dies ist hauptsächlich dadurch begründet, dass die Skalierungsentscheidungen auf Basis der durchschnittlichen CPU-Auslastung der Applikationsinstanzen eines Mandanten getroffen werden und nicht aufgrund der CPU-Auslastung einer virtuellen Maschine. Abhängig von der Ziel-Auslastung der HPA kann es dadurch dazu kommen, dass die CPU-Kapazitäten der virtuellen Maschine bereits ausgelastet sind, jedoch der Entscheidungsprozess des HPA noch keine Erhöhung der Anzahl der Applikationsinstanzen ergibt.

3.6.6 Trennung in System-Node-Pool und Mandanten-Node-Pool

Im Zuge der ersten durchgeführten Tests fiel auf, dass die Auslastung der ersten virtuellen Maschine im Testsystem immer höher war als jene der anderen virtuellen Maschinen. Nach genauerer Untersuchung wurde deutlich, dass dies auf die zusätzliche Auslastung aufgrund der Kubernetes System-Services, dem Ingress-Controller sowie der Load-Balancer zurückzuführen ist. Dadurch wurden die Messungen zur Auslastung aufgrund der Testapplikation verfälscht.

Um diesem Problem entgegenzuwirken, wird eine dedizierte virtuelle Maschine für die Kubernetes System-Services, den Ingress-Controller und die Load-Balancer bereitgestellt. Dazu werden zwei getrennte Node-Pools in Form eines System-Node-Pools und eines Mandanten-

Node-Pools bereitgestellt. Der System-Node-Pool hat eine fest definierte Größe von einer virtuellen Maschine im leistungsstärken Azure Format DS4v2 mit 8 vCPUs und 28 GB RAM. Der Mandanten-Node-Pool beinhaltet virtuelle Maschinen im Format DS2v2 mit 2 vCPUs und 7 GB RAM.

Um die Services und Mandant-Instanzen innerhalb der richtigen Node-Pools zu starten, wird mit Taints und Tolerations gearbeitet. Dabei wird der Mandanten-Node-Pool mit einem entsprechenden Taint konfiguriert und die Mandanten-Applikationsinstanzen mit einer zugehörigen Toleration. Dadurch können auf den virtuellen Maschinen im Mandanten-Node-Pool nur die Mandanten-Applikationsinstanzen gestartet werden. Alle System-Services sowie auch Ingress-Controller und Load-Balancer werden somit im System-Node-Pool gestartet. (Kubernetes, 2020c)

3.7 Testpläne

Für die Durchführung der Elastizitätstests werden verschiedene Testpläne erstellt. Das grundlegende Szenario ist in allen Testplänen dasselbe und basiert auf dem in der vorangegangenen Arbeit verwendeten Szenario (Winkler, 2019).

Aufbauend auf dem Testszenario werden die Testpläne mit unterschiedlichen Auslastungsmustern erstellt. Die variierende Auslastung wird dabei über unterschiedliche Anzahlen von gleichzeitigen virtuellen Benutzern abgebildet. Ein virtueller Benutzer entspricht einem Thread, der das definierte Testszenario wiederholt hintereinander ausführt. Dementsprechend wird die Workload im Rahmen dieser Arbeit über die Anzahl der gleichzeitigen virtuellen Benutzer definiert.

Die Ausführung der JMeter-Testpläne erfolgt auf zwei dedizierten Microsoft Azure DS4v2 Maschinen. Auf diese Weise werden die identen Testpläne auf zwei vollkommen getrennten, unabhängigen Testsystemen ausgeführt.

3.7.1 Testszenario

Dieses Szenario simuliert, wie in Abbildung 3-10 dargestellt, zwei Benutzergruppen. Eine dieser Benutzergruppen sind Administratoren, welche sich im System registrieren, einloggen und anschließend Produkte bearbeiten. Über Schleifen im Szenario werden Wiederholungen der einzelnen Schritte simuliert. So wird davon ausgegangen, dass sich der Administrator einmal registriert, zehnmal anmeldet und dabei jeweils drei Produkte bearbeitet. (Winkler, 2019)

In ähnlicher Weise läuft das Szenario für einen Webshop Kunden ab. Es erfolgt eine einmalige Registrierung. Anschließend werden über die Login-Schleife 20 Anmeldungen im Webshop simuliert, zu denen der Kunde jeweils zwei Produkte sucht, in den Warenkorb legt und den Warenkorb anzeigen lässt. (Winkler, 2019)

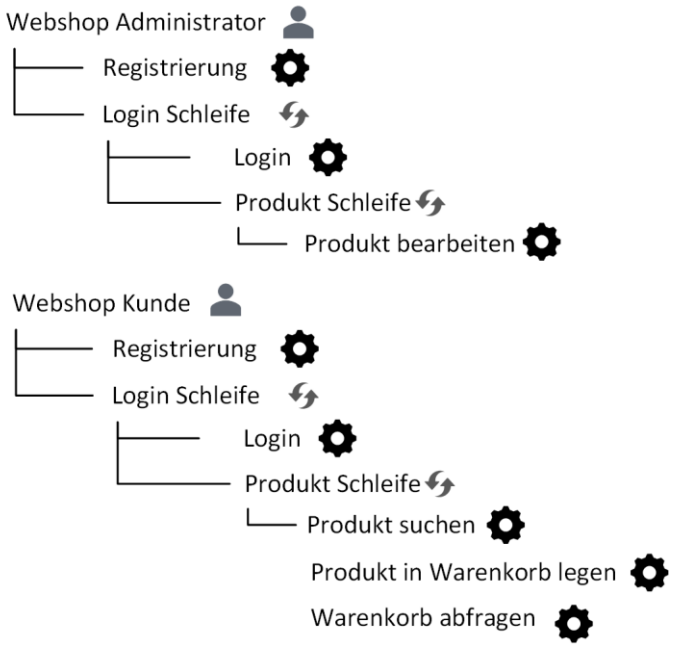


Abbildung 3-10 Testszenario (Quelle: Winkler, 2019)

Im Testszenario wird zwischen jeder ausgelösten Anfrage ein konstanter Timer von fünf Millisekunden eingebaut. Ohne diesen Timer würde JMeter das Testszenario wiederholt so schnell als möglich ausführen. Dadurch würde die Anzahl der gesendeten Anfragen von der Performanz der Maschine abhängen, auf dem JMeter ausgeführt wird. Durch die Verwendung des Timers werden die Testpläne von der Performanz der ausführenden Maschine entkoppelt.

3.7.2 Plateau Auslastung

Das erste Auslastungsmuster entspricht, wie in Abbildung 3-11 ersichtlich, einem Plateau, das zu Beginn über einen Zeitraum von 20 Minuten insgesamt 240 Threads startet. Diese Workload wird dann über eine Dauer von 30 Minuten gehalten. Anschließend werden wieder alle 240 Threads über eine Dauer von 20 Minuten gestoppt.

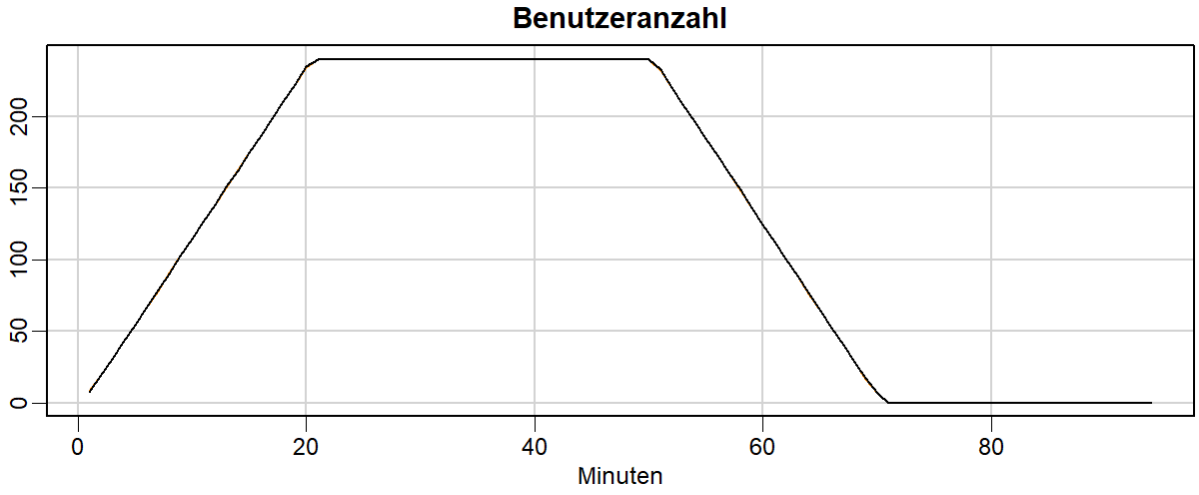


Abbildung 3-11 Auslastungsmuster Plateau mit 240 gleichzeitigen Benutzern (Quelle: eigene Darstellung RStudio Version 1.1.383)

3.7.3 Sinus Auslastung

Das zweite Auslastungsmuster stellt einen sinus-ähnlichen Verlauf mit steigender und fallender Workload dar, wie in Abbildung 3-12 dargestellt. Dabei werden zuerst innerhalb von zwei Minuten 15 Threads gestartet. Nach 15 Minuten wird die Anzahl der Threads innerhalb von zehn Minuten auf 150 erhöht und für eine Dauer von 15 Minuten gehalten. Anschließend werden die Threads innerhalb von fünf Minuten wieder auf 15 reduziert. Nach insgesamt 75 Minuten wiederholt sich dieser Vorgang noch einmal. Anschließend wird die Workload von 15 Threads noch für 47 Minuten gehalten bevor alle Threads gestoppt werden. Dasselbe Muster wird auch mit einer Benutzeranzahl von 250 verwendet, wie in Abbildung 3-13 dargestellt.

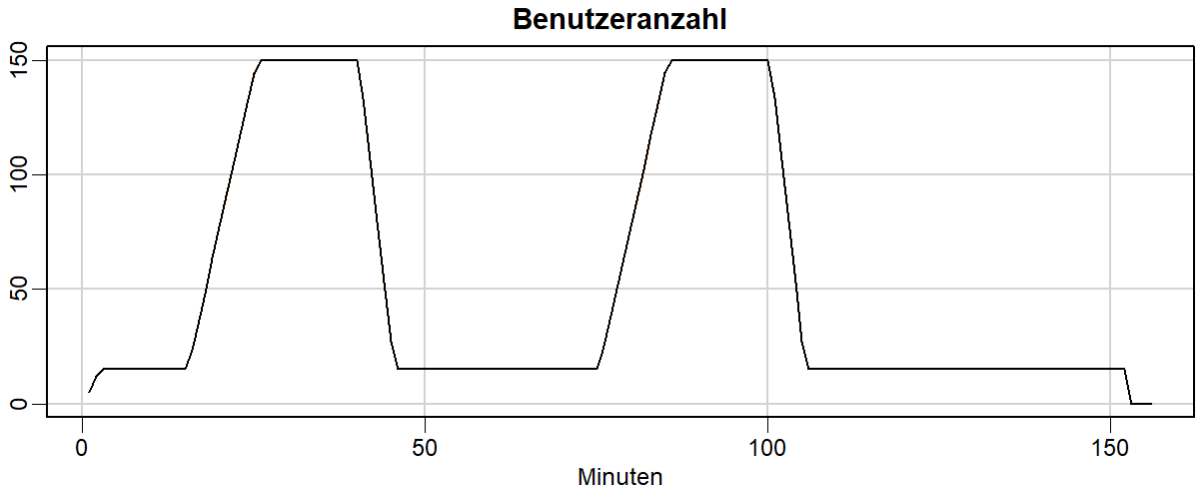


Abbildung 3-12 Auslastungsmuster sinus-ähnlich mit 150 gleichzeitigen Benutzern (Quelle: eigene Darstellung RStudio Version 1.1.383)

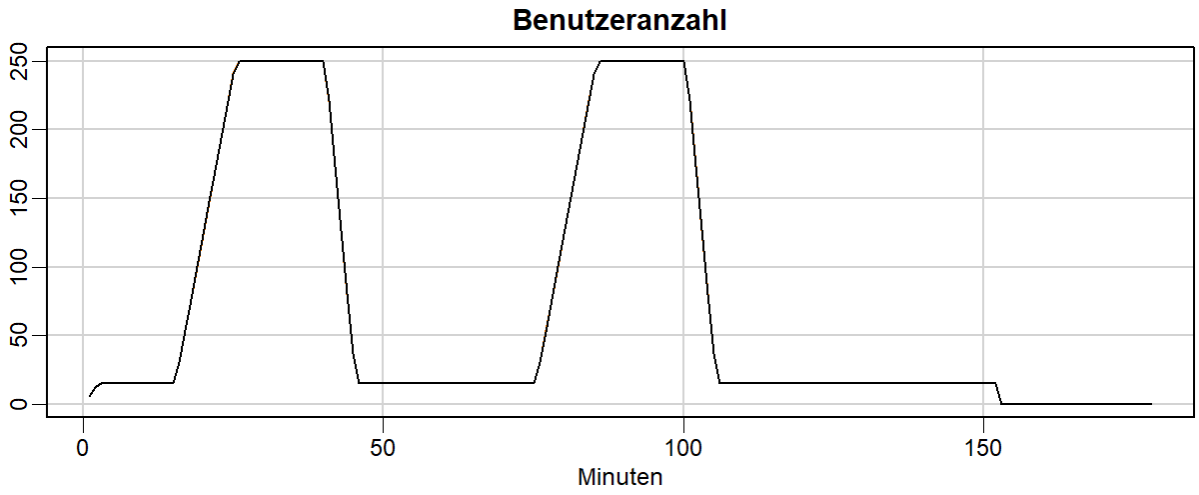


Abbildung 3-13 Auslastungsmuster sinus-ähnlich mit 250 gleichzeitigen Benutzern (Quelle: eigene Darstellung RStudio Version 1.1.383)

3.7.4 Auslastungsspitze und Ramp-Down

Als drittes Auslastungsmuster wird, wie in Abbildung 3-14 dargestellt, eine schnelle Auslastungsspitze mit anschließender Abflachung der Workload abgebildet. Dazu werden zu Beginn innerhalb von zwei Minuten 15 Threads gestartet. Nach 15 Minuten steigt die Workload rapide innerhalb von zehn Minuten auf 360 Threads an und wird für 15 Minuten gehalten. Anschließend werden die Threads über eine Dauer von 75 Minuten allmählich wieder gestoppt. Dasselbe Muster wird, wie in Abbildung 3-15 dargestellt, ein zweites Mal mit einer Benutzeranzahl von 600 verwendet.

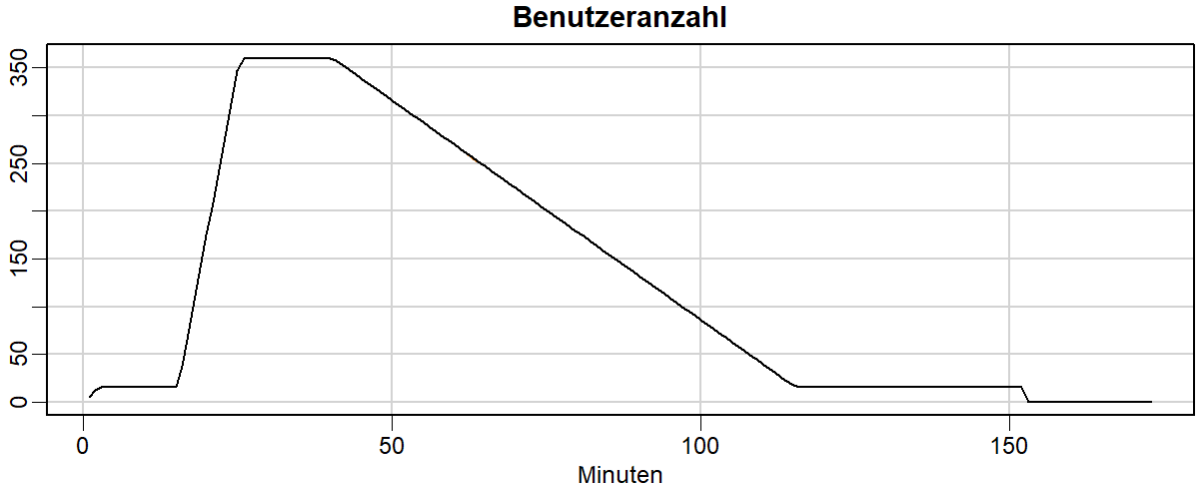


Abbildung 3-14 Auslastungsspitze mit 360 Benutzern und anschließendem Ramp-Down (Quelle: eigene Darstellung RStudio Version 1.1.383)

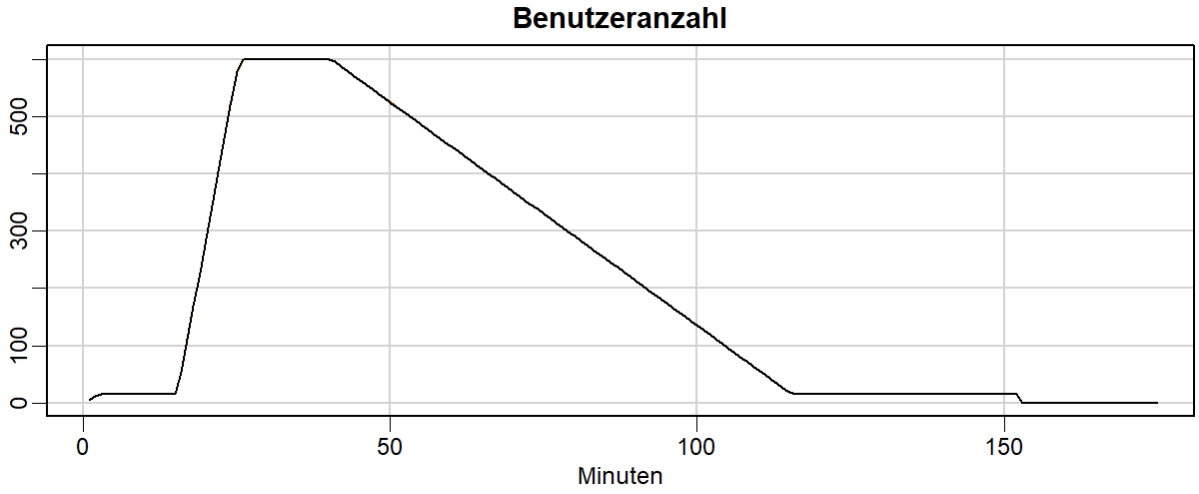


Abbildung 3-15 Auslastungsspitze mit 600 Benutzern und anschließendem Ramp-Down (Quelle: eigene Darstellung RStudio Version 1.1.383)

3.7.5 Auslastungsverteilung auf Mandanten

Für die Ausführung der Elastizitätstests im Rahmen dieser Arbeit wurde der Testplan aus der vorangegangenen Arbeit adaptiert, um die Auslastung auf die verfügbaren Mandanten im Testsystem zu verteilen. Über eine herkömmliche Zufallsvariable würde sich eine gleichmäßige Verteilung der Workload über alle Mandanten ergeben. So eine gleichmäßige Verteilung der Workload auf alle Mandanten ist allerdings nicht realistisch. In der Praxis unterscheiden sich die Anzahl der BenutzerInnen sowie dessen Frequenz über die Mandanten. Um höhere Praxisnähe zu erreichen, wird die Mandantenzugehörigkeit einer Iteration im Testplan auf Basis einer approximierten Gauß-Verteilung bestimmt. Bei jedem Durchlauf des Testszenarios wird ein neuer Mandant für die darin enthaltenen Anfragen ermittelt. Auf diese Weise wird eine stärkere Auslastung der mittleren Mandanten-IDs und eine schwächere Auslastung der höheren oder niedrigeren Mandanten-IDs erreicht. Im Testplan erfolgt dies über die Definition eines Parameters, welcher mithilfe eines eingebetteten Javascript-Codes für jede Iteration die approximierte, Gauß-verteilte Zufallsvariable berechnet (*JavaScript - Random with Gaussian Distribution*, 2020).

3.8 Hypothese

Wie im Kapitel 2.2 behandelt, können über Elastizität die verfügbaren Ressourcen eines Systems abhängig von der Ressourcenanforderung angepasst werden. Auf diese Weise sollen die verfügbaren Ressourcen eines Systems zu jeder Zeit möglichst genau den benötigten Ressourcen entsprechen. Aufgrund dieser Beschreibung von Elastizität ist davon auszugehen, dass sich durch elastische Eigenschaften Vorteile für die Performanz und Kosten eines Systems gegenüber einem statischen System ergeben.

Bei niedriger Auslastung des Systems können die verfügbaren Ressourcen des Systems reduziert werden. Dadurch reduzieren sich die ungenutzten Ressourcen des Systems und die laufenden Kosten können gesenkt werden. In einem statischen System hingegen bleiben die verfügbaren Ressourcen und damit die laufenden Betriebskosten gleich. Umgekehrt kann das System bei hoher Auslastung zusätzliche Ressourcen allokieren, um den erhöhten Anforderungen nachzukommen. Auf diese Weise soll die Performanz des Systems bei hoher Auslastung erhalten werden. Im Vergleich dazu ist das statische System nicht in der Lage, weitere Ressourcen zu nutzen, wodurch die Performanz bei hoher Auslastung sinkt. Basierend auf diesen Annahmen wird der Ausführung der Elastizitätstests folgende Hypothese zugrunde gelegt.

> *Die Performanz und Kosten eines mandantenfähigen, containerbasierten Webshop-Systems werden durch Elastizität verbessert.*

Dieselben Testpläne werden auf beiden Systemen ausgeführt, um die Performanz und Kosten dieser zu erheben. Über die Auswertung dieser Ergebnisse soll die obenstehende Hypothese bestätigt oder widerlegt werden.

4 TESTERGEBNISSE UND AUSWERTUNG

Bei der Durchführung der Tests werden die benötigten Metriken entsprechend den Ausführungen in Kapitel 3.4 gemessen. Auf Basis der erhobenen Messwerte werden die Auswertungen und die Konsolidierung der Ergebnisse entsprechend den Beschreibungen in Kapitel 3.2 und 3.3 durchgeführt. Aufgrund dieser Ergebnisse werden die Schlussfolgerungen in Bezug auf Performanz, Elastizität und Kosten der beiden Testsysteme gezogen.

Bei jedem durchgeführten Testlauf wird derselbe Testplan gleichzeitig in zwei vollkommen getrennten Testumgebungen ausgeführt. Für jeden Testlauf müssen die folgenden Parameter festgelegt werden:

- ein Testplan mit einer der in Kapitel 3.7 beschriebenen Auslastungskurven
- Workload als Anzahl der virtuellen Benutzer
- die Anzahl der Mandanten im statischen und elastischen Testsystem
- die Anzahl der virtuellen Maschinen im statischen Testsystem

In Tabelle 4-1 sind die ausgeführten Testläufe mit den gewählten Parametern zusammengefasst.

Testlauf Nr.	Auslastungsmuster	virtuelle Benutzer	Mandanten-anzahl	Anzahl VM statisch
Testlauf 1	Plateau	240	15	3
Testlauf 2	Plateau	240	25	5
Testlauf 3	Sinus	150	15	3
Testlauf 4	Sinus	250	25	5
Testlauf 5	Spitze und Ramp-Down	360	15	3
Testlauf 6	Spitze und Ramp-Down	600	25	5

Tabelle 4-1 Übersicht der durchgeführten Testläufe (Quelle: eigene Inhalte)

4.1 Ergebnisse

Für jeden Testlauf werden die gemessenen und berechneten Ergebnisse grafisch aufbereitet und in Liniendiagrammen dargestellt. Die abgebildeten Werte entsprechen den im Kapitel 3.2 und 3.3 beschriebenen Metriken. In jedem Liniendiagramm werden die Messwerte des statischen und des elastischen Systems verglichen.

Diagramm 1: Das erste Diagramm zeigt die Anfragen pro Sekunde und repräsentiert somit den Verlauf der Workload.

Diagramm 2: Im zweiten Diagramm werden die verfügbaren und die genutzten CPU-Ressourcen der beiden Systeme in Millicores dargestellt.

Diagramm 3: Das dritte Diagramm zeigt die durchschnittliche Antwortzeit $\bar{r}$ und erlaubt somit den Vergleich der Performanz der beiden Systeme.

Diagramme 4 bis 6: Die Diagramme 4 bis 6 zeigen die maximale Antwortzeit sowie die 99%- und 95%-Quantile.

Diagramme 7 bis 9: Die Diagramme 7 und 8 beinhalten die Metriken in Bezug auf Elastizität. Das sind Überversorgung $O(t)$ und Unterversorgung $U(t)$. In Diagramm 9 wird die Summe der Kosten für Über- und Unterversorgung $C_{o+u}(t)$ dargestellt. Auf separate Diagramme für die Kosten von Überversorgung $C_o(t)$ und Unterversorgung $C_u(t)$ wird verzichtet, da diese denselben Verlauf wie die Diagramme 7 und 8 auf einer anderen Skala zeigen.

Diagramm 10: Das Diagramm 10 zeigt die Kosten für die verwendeten CPU-Ressourcen $C_d(t)$.

Diagramm 11: Auf dem Diagramm 11 werden die Gesamtkosten $C(t)$ als die Summe der Kosten in den Diagrammen 9 und 10 dargestellt.

4.1.1 Testlauf 1: Plateau - 240 Benutzer - 15 Mandanten - 3 VM statisch

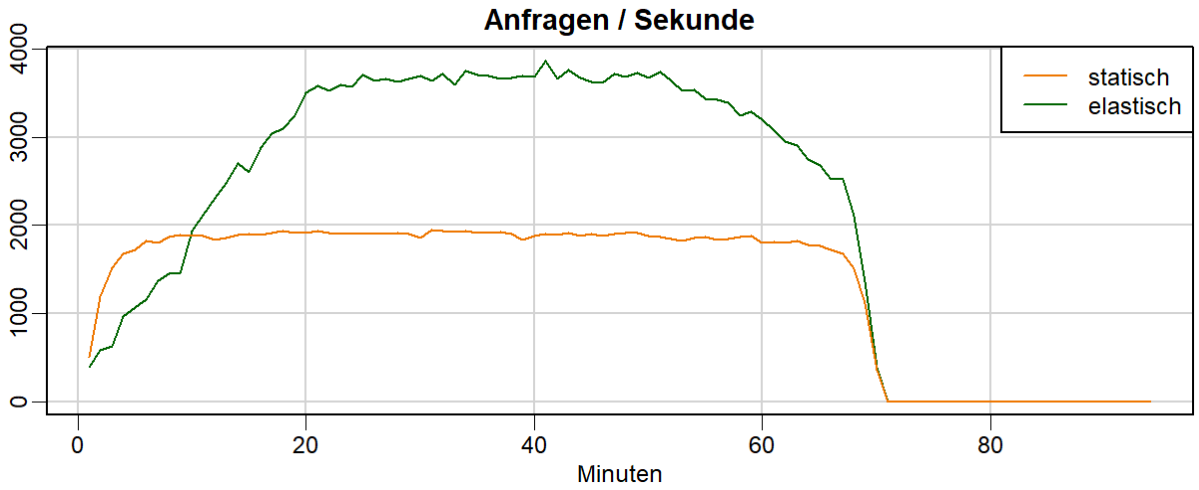


Abbildung 4-1 Plateau-Auslastung mit 240 virtuellen Benutzern und 15 Mandanten: Diagramm 1: Vergleich der Anfragen pro Sekunde (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

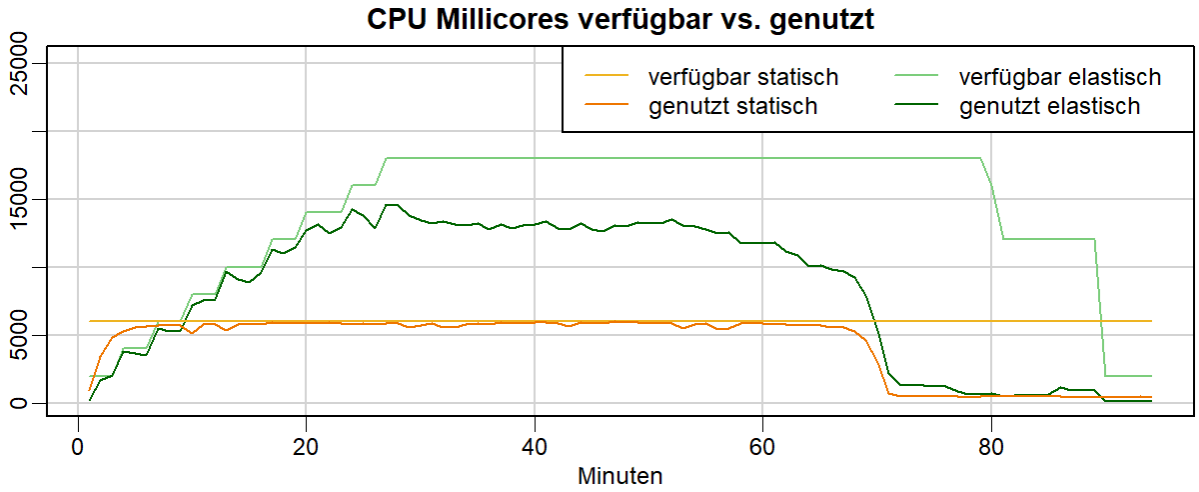


Abbildung 4-2 Plateau-Auslastung mit 240 virtuellen Benutzern und 15 Mandanten: Diagramm 2: verfügbare und genutzte CPU (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

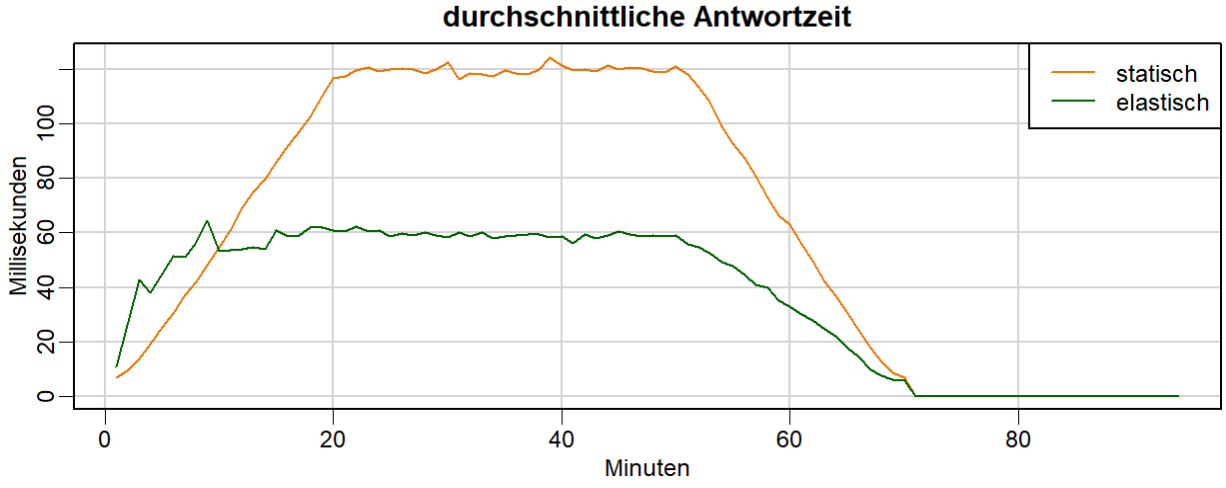


Abbildung 4-3 Plateau-Auslastung mit 240 virtuellen Benutzern und 15 Mandanten: Diagramm 3: Vergleich der durchschnittlichen Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

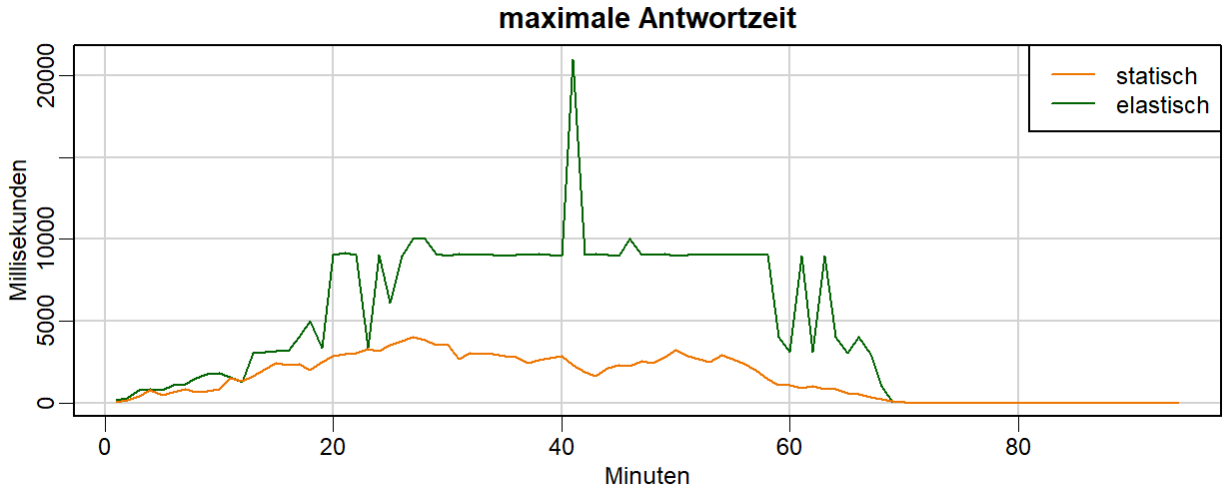


Abbildung 4-4 Plateau-Auslastung mit 240 virtuellen Benutzern und 15 Mandanten: Diagramm 4: Vergleich der maximalen Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

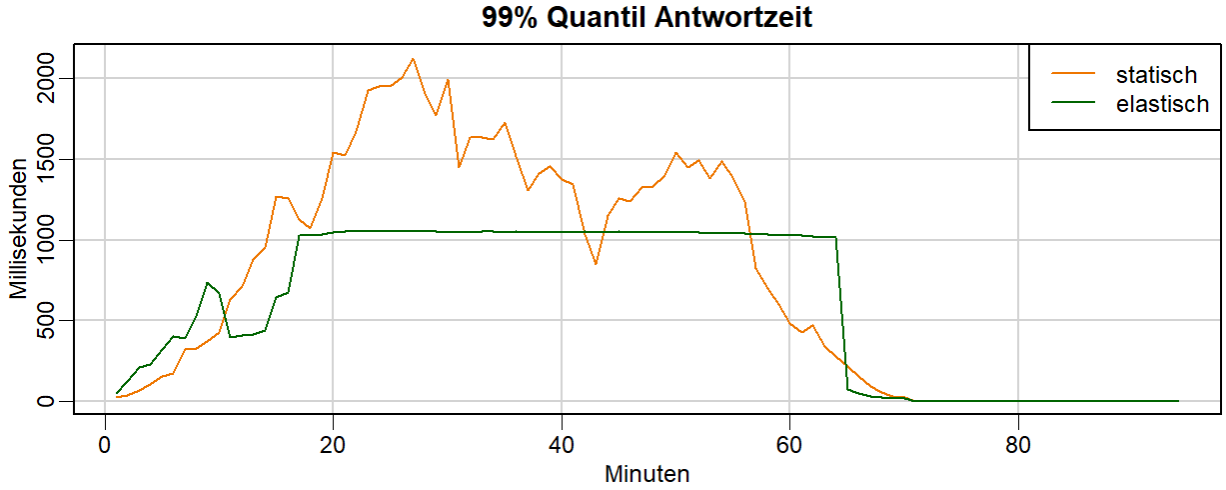


Abbildung 4-5 Plateau-Auslastung mit 240 virtuellen Benutzern und 15 Mandanten: Diagramm 5: Vergleich der 99% Quantile der Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

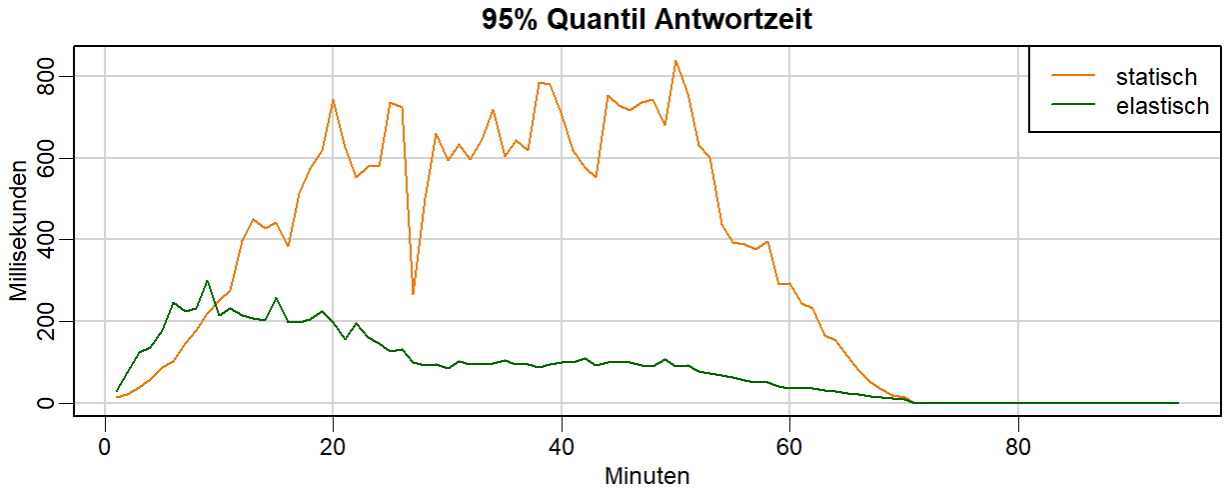


Abbildung 4-6 Plateau-Auslastung mit 240 virtuellen Benutzern und 15 Mandanten: Diagramm 6: Vergleich der 95% Quantile der Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

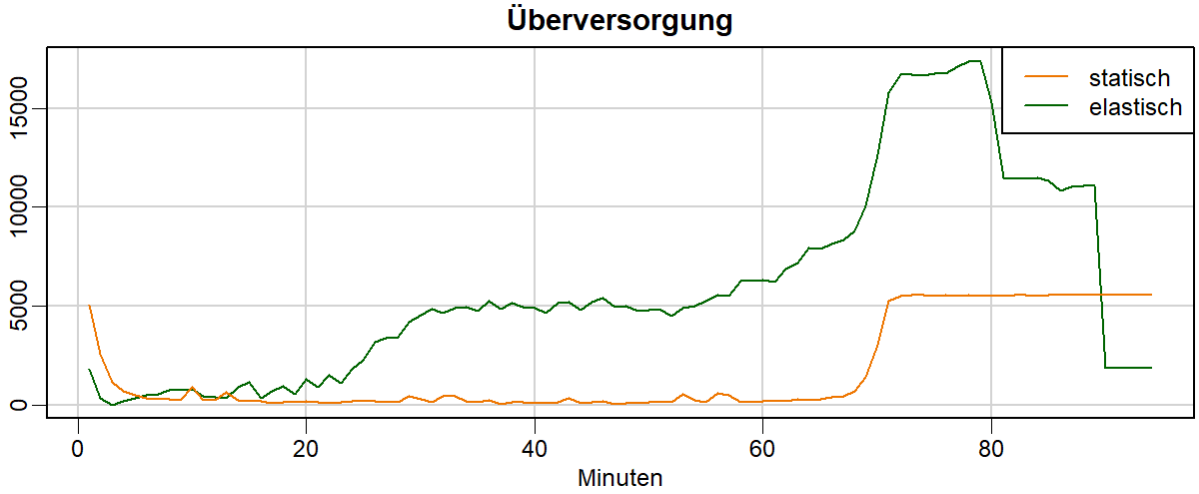


Abbildung 4-7 Plateau-Auslastung mit 240 virtuellen Benutzern und 15 Mandanten: Diagramm 7: Vergleich der Überversorgung (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

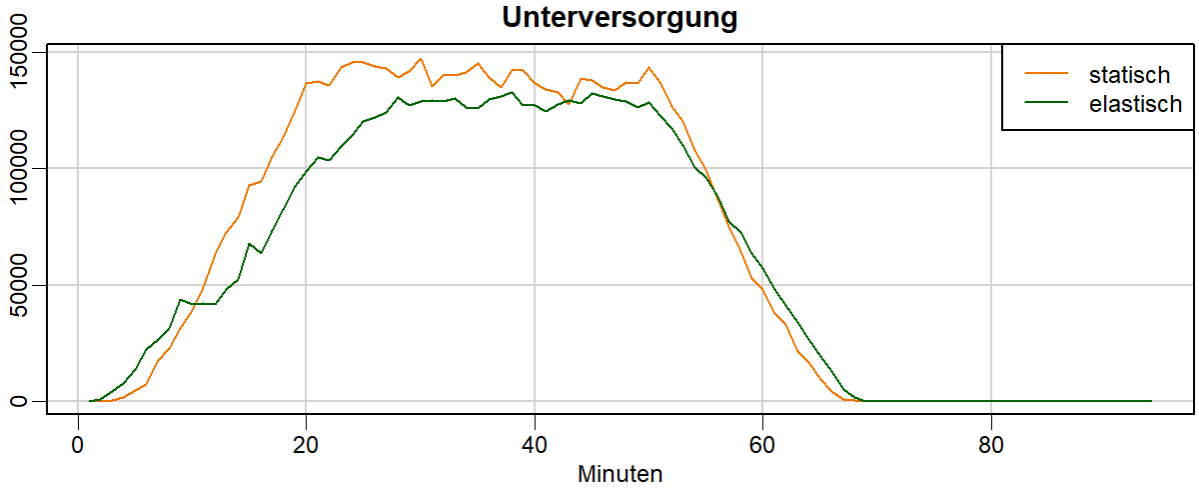


Abbildung 4-8 Plateau-Auslastung mit 240 virtuellen Benutzern und 15 Mandanten: Diagramm 8: Vergleich der Unterversorgung (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

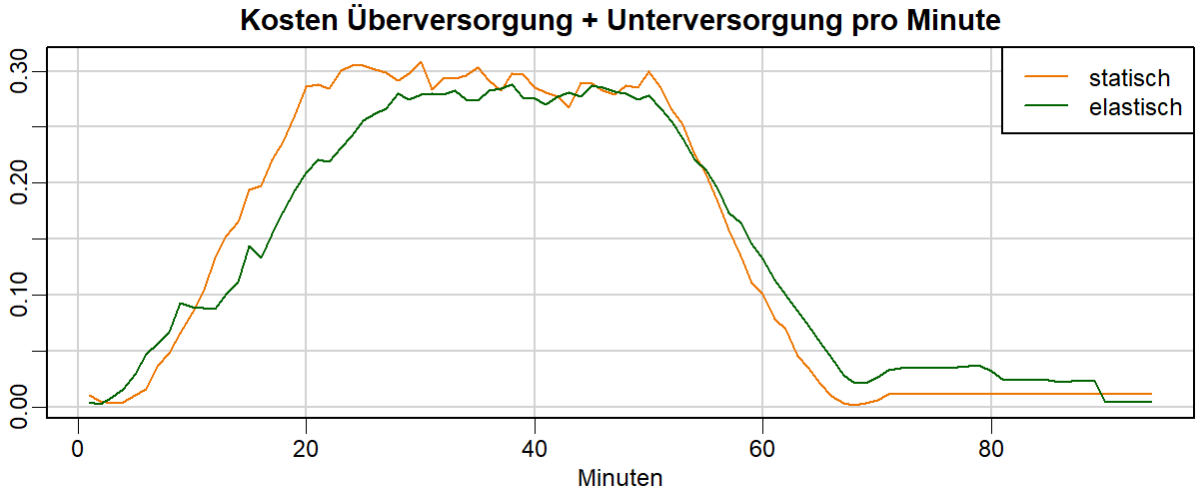


Abbildung 4-9 Plateau-Auslastung mit 240 virtuellen Benutzern und 15 Mandanten: Diagramm 9: Vergleich der Kosten für Überversorgung + Unterversorgung (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

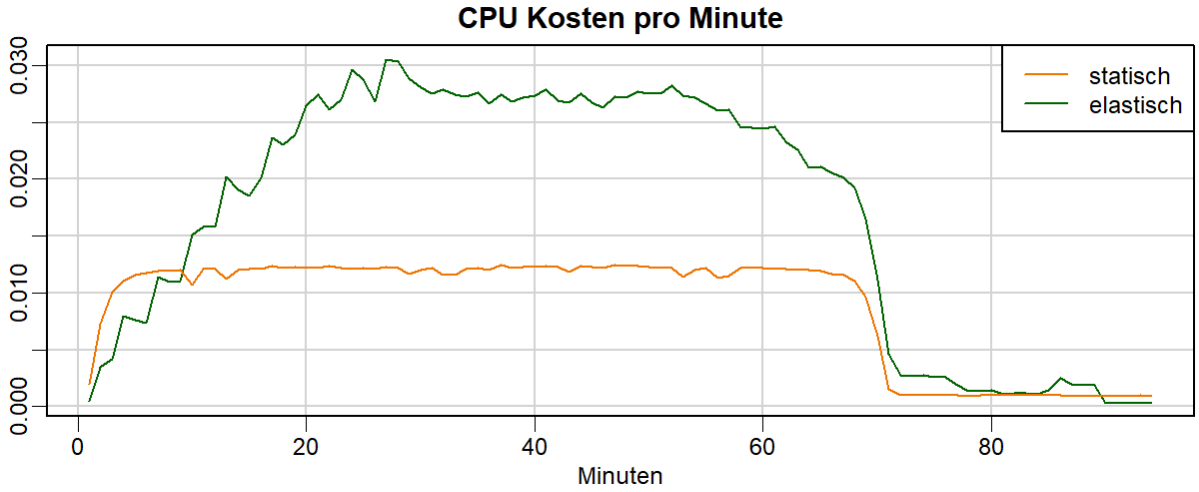


Abbildung 4-10 Plateau-Auslastung mit 240 virtuellen Benutzern und 15 Mandanten: Diagramm 10: Vergleich der Kosten der tatsächlich genutzten CPU-Kapazitäten (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

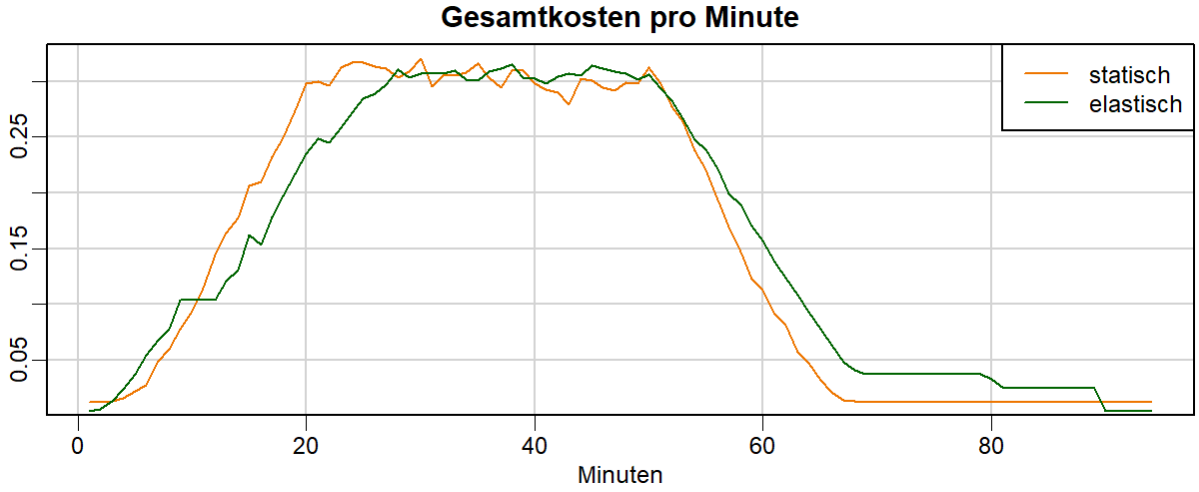


Abbildung 4-11 Plateau-Auslastung mit 240 virtuellen Benutzern und 15 Mandanten: Diagramm 11: Vergleich der Gesamtkosten (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

Das plateauförmige Auslastungsmuster ist in den Diagrammen von Testlauf 1 klar zu erkennen. Des Weiteren sind deutlich höhere CPU-Ressourcen im elastischen System, sowie kürzere durchschnittliche Antwortzeiten ersichtlich. Daraus ergibt sich auch eine höhere Anzahl an bearbeiteten Anfragen im elastischen System. In den Quantilen der Antwortzeit ist auffällig, dass die maximale Antwortzeit des elastischen Systems deutlich über der des statischen Systems, das 95%-Quantil jedoch bereits weit unterhalb liegt. In den Diagrammen von Über- und Unterversorgung sowie den Kosten dieser beiden ist klar eine dominante Rolle der Unterversorgungskosten zu erkennen. Auf die gleiche Weise ist auch die höhere Auswirkung der Über- und Unterversorgungskosten gegenüber den CPU Kosten in den Gesamtkosten deutlich ersichtlich. Diese und ähnliche Diagrammverläufe sind ebenso bei den folgenden Tests zu beobachten.

4.1.2 Testlauf 2: Plateau - 240 Benutzer - 25 Mandanten - 5 VM statisch

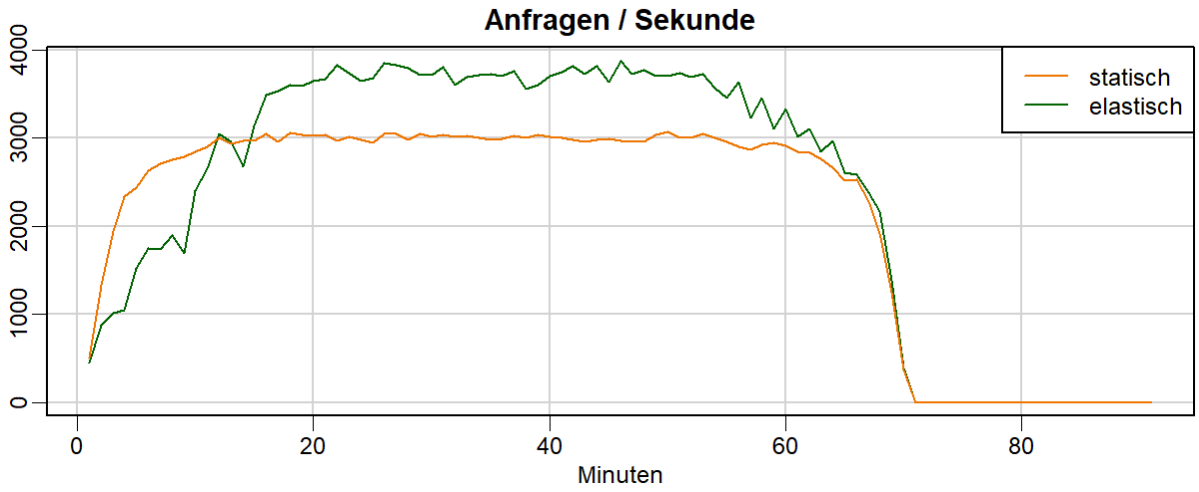


Abbildung 4-12 Plateau-Auslastung mit 240 virtuellen Benutzern und 25 Mandanten: Diagramm 1: Vergleich der Anfragen pro Sekunde (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

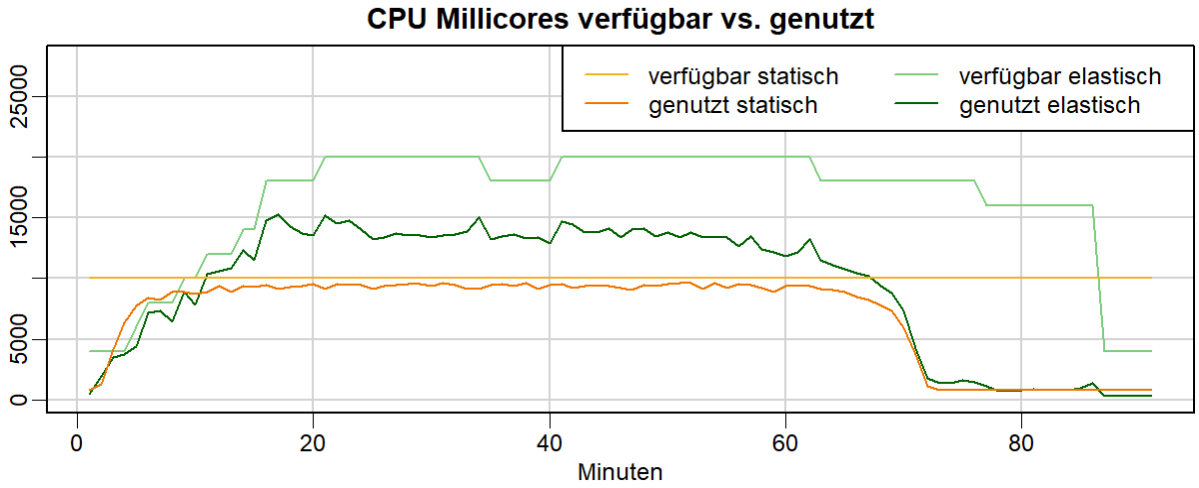


Abbildung 4-13 Plateau-Auslastung mit 240 virtuellen Benutzern und 25 Mandanten: Diagramm 2: verfügbare und genutzte CPU (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

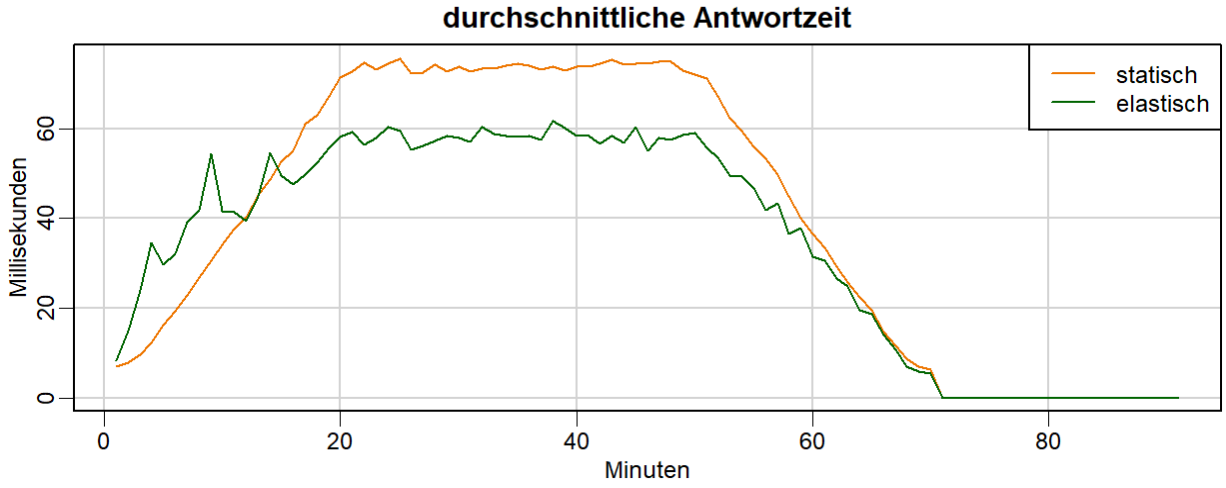


Abbildung 4-14 Plateau-Auslastung mit 240 virtuellen Benutzern und 25 Mandanten: Diagramm 3: Vergleich der durchschnittlichen Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

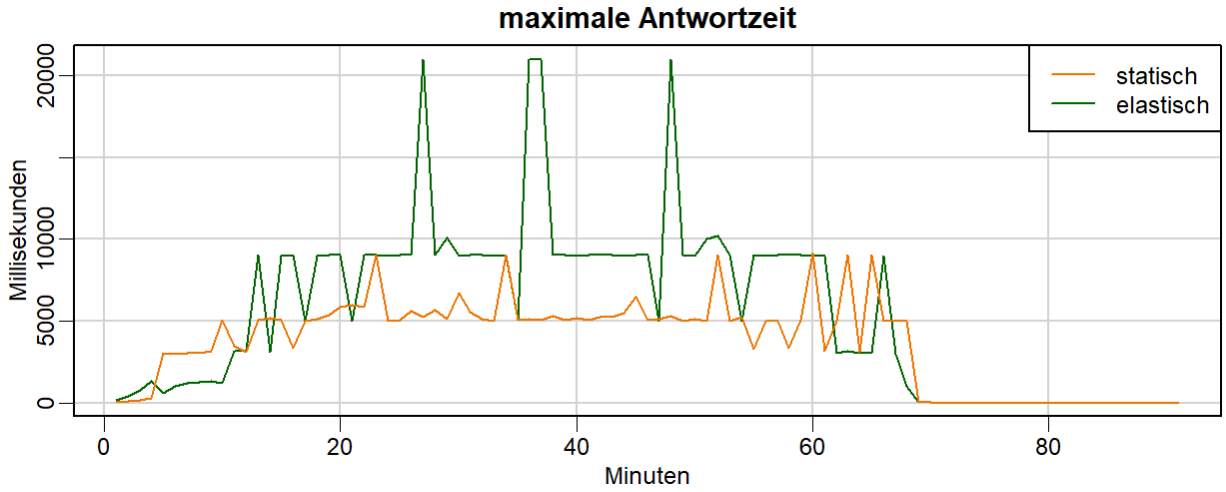


Abbildung 4-15 Plateau-Auslastung mit 240 virtuellen Benutzern und 25 Mandanten: Diagramm 4: Vergleich der maximalen Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

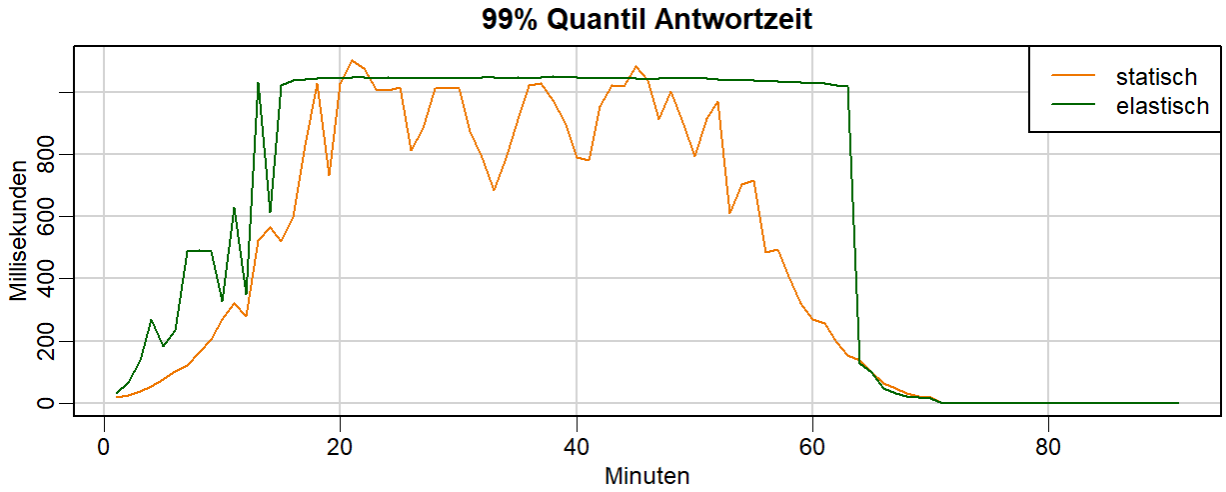


Abbildung 4-16 Plateau-Auslastung mit 240 virtuellen Benutzern und 25 Mandanten: Diagramm 5: Vergleich der 99% Quantile der Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

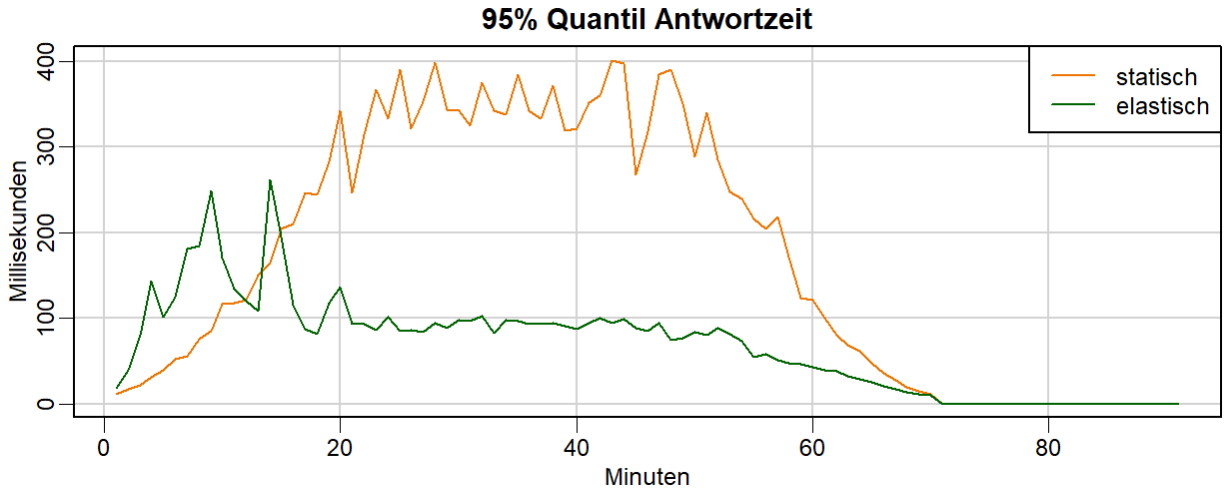


Abbildung 4-17 Plateau-Auslastung mit 240 virtuellen Benutzern und 25 Mandanten: Diagramm 6: Vergleich der 95% Quantile der Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

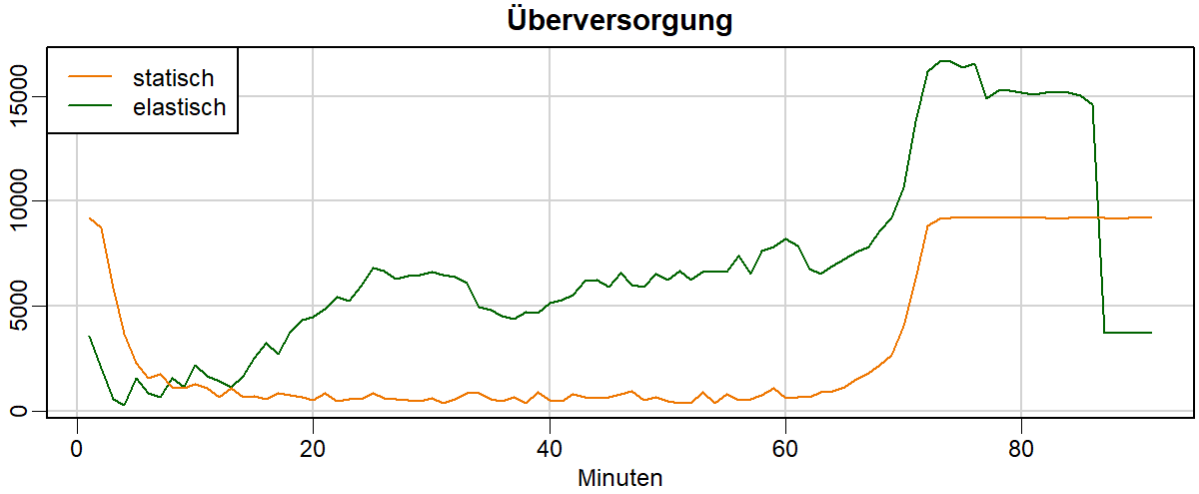


Abbildung 4-18 Plateau-Auslastung mit 240 virtuellen Benutzern und 25 Mandanten: Diagramm 7: Vergleich der Überversorgung (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

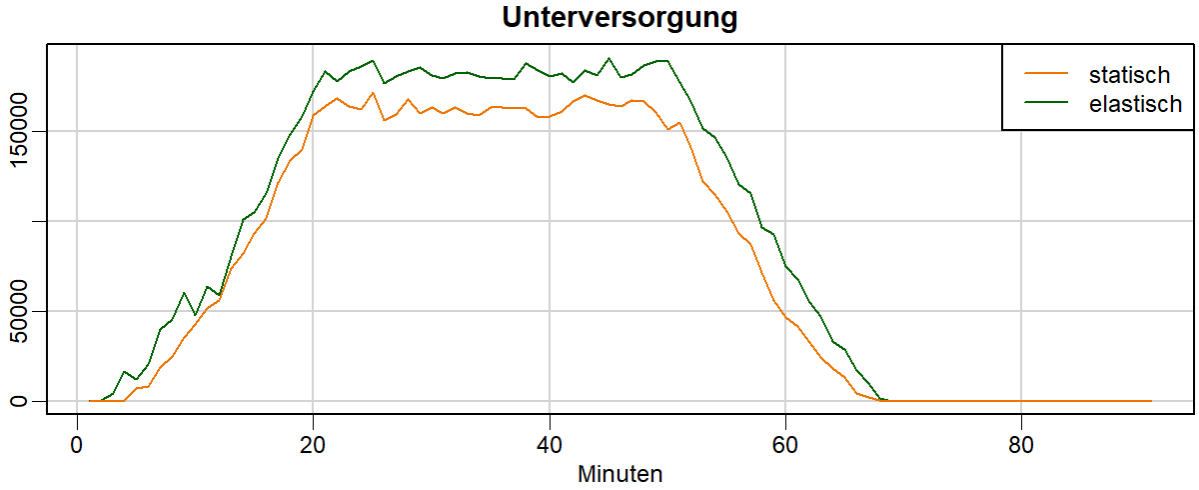


Abbildung 4-19 Plateau-Auslastung mit 240 virtuellen Benutzern und 25 Mandanten: Diagramm 8: Vergleich der Unterversorgung (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

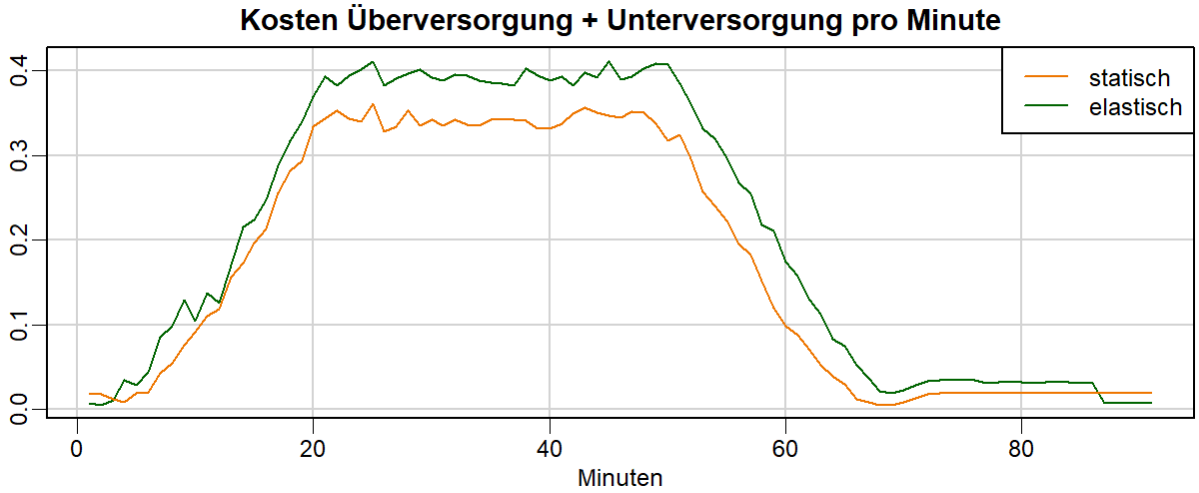


Abbildung 4-20 Plateau-Auslastung mit 240 virtuellen Benutzern und 25 Mandanten: Diagramm 9: Vergleich der Kosten für Überversorgung + Unterversorgung (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

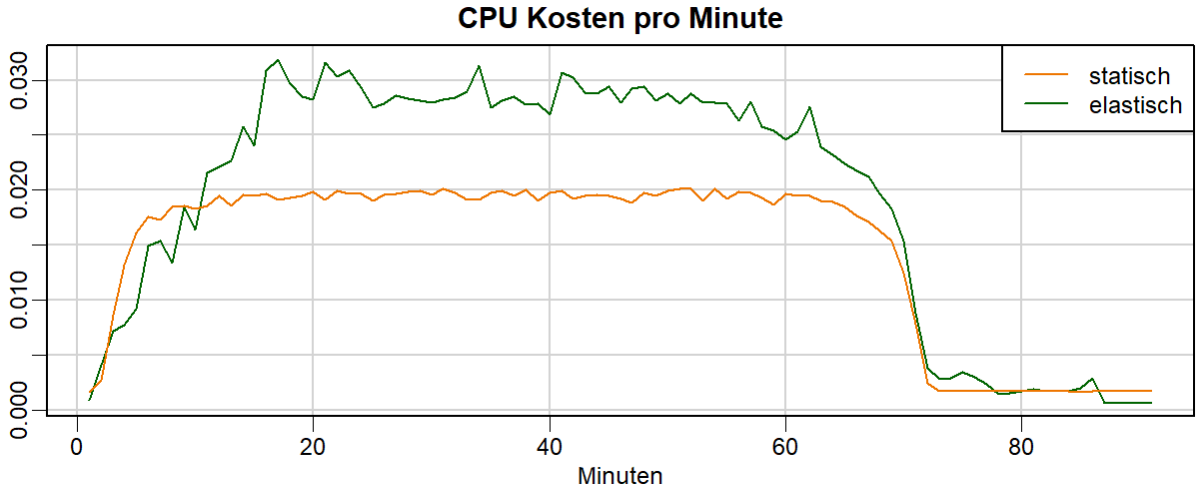


Abbildung 4-21 Plateau-Auslastung mit 240 virtuellen Benutzern und 25 Mandanten: Diagramm 10: Vergleich der Kosten der tatsächlich genutzten CPU-Kapazitäten (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

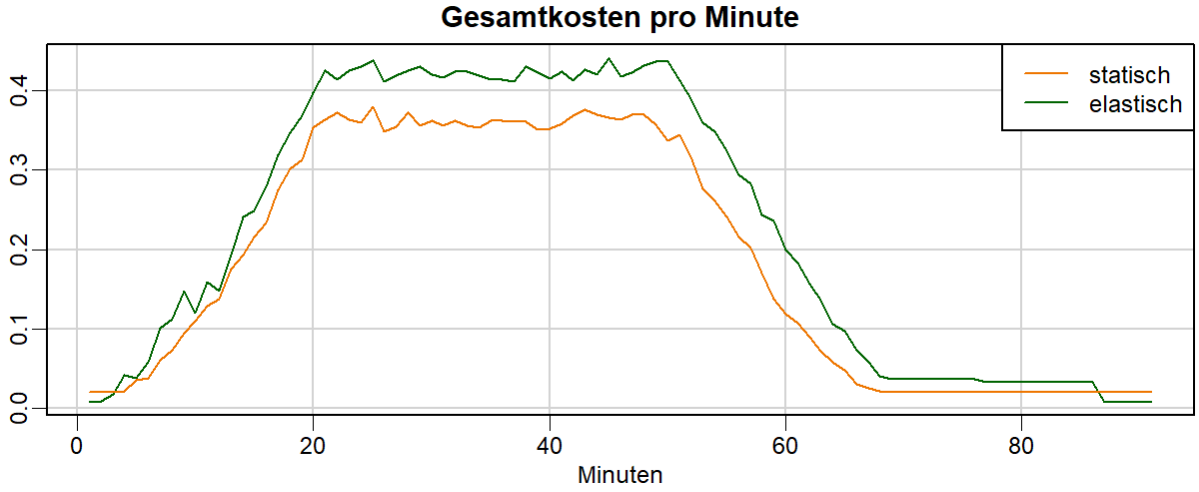


Abbildung 4-22 Plateau-Auslastung mit 240 virtuellen Benutzern und 25 Mandanten: Diagramm 11: Vergleich der Gesamtkosten (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

Aufgrund desselben Testplans sind die Rahmenbedingungen für das elastische System Testlauf 1 und 2 gleich. Auch die Resultate des elastischen Systems weisen im Vergleich zu Testlauf 1 keine wesentlichen Unterschiede auf. Dadurch wird das in 3.4.3 beschriebene Risiko von stark schwankender Performanz der virtuellen Maschinen in unterschiedlichen Tests ein Stück weit entkräftet. Das statische System hatte im Testlauf 2 allerdings fünf virtuelle Maschinen anstatt der drei virtuellen Maschinen im Testlauf 1. Aufgrund dieser Parameter sind in den Ergebnissen von Testlauf 1 und Testlauf 2 deutliche Parallelen zu erkennen. Unter anderem ist wie in Testlauf 1 gut ersichtlich, dass die durchschnittliche Antwortzeit im elastischen System vorerst stark ansteigt, die Kurve danach allerdings mit der Bereitstellung von zusätzlichen CPU-Ressourcen abflacht und somit unter jener des statischen Systems verbleibt. Unterdessen haben sich im statischen System sowohl die durchschnittliche Antwortzeit sowie die 99%- und 95%-Quantile gegen Testlauf 1 deutlich verkürzt. Aufgrund dieser verkürzten Antwortzeiten haben sich auch die Kosten für Unterversorgung merklich verringert, wodurch die Gesamtkosten nun, anders als in Testlauf 1, im gesamten Testverlauf unter denen des elastischen Systems liegen.

4.1.3 Testlauf 3: Sinus - 150 Benutzer - 15 Mandanten - 3 VM statisch

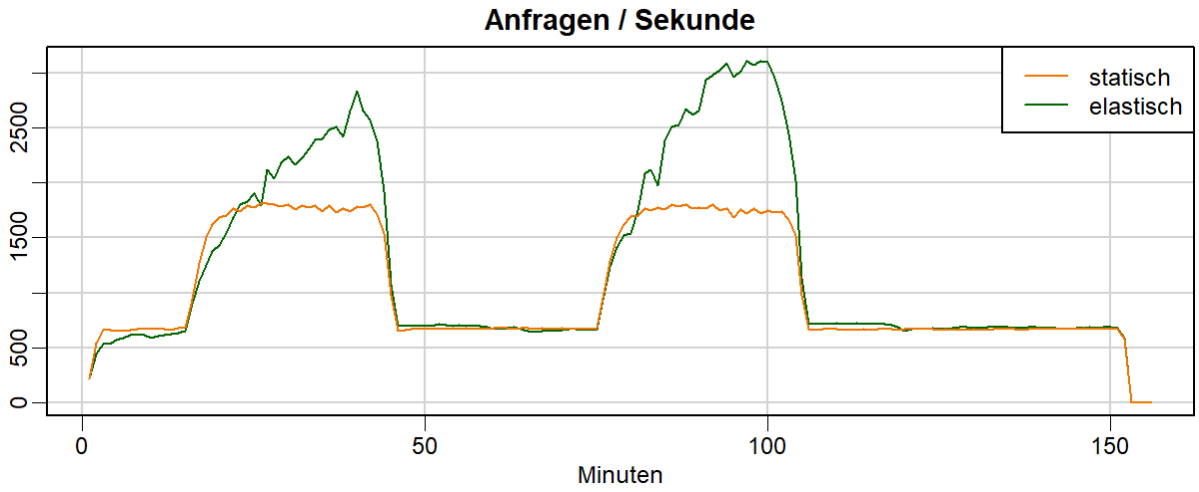


Abbildung 4-23 Sinus-Auslastung mit 150 virtuellen Benutzern und 15 Mandanten: Diagramm 1: Vergleich der Anfragen pro Sekunde (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

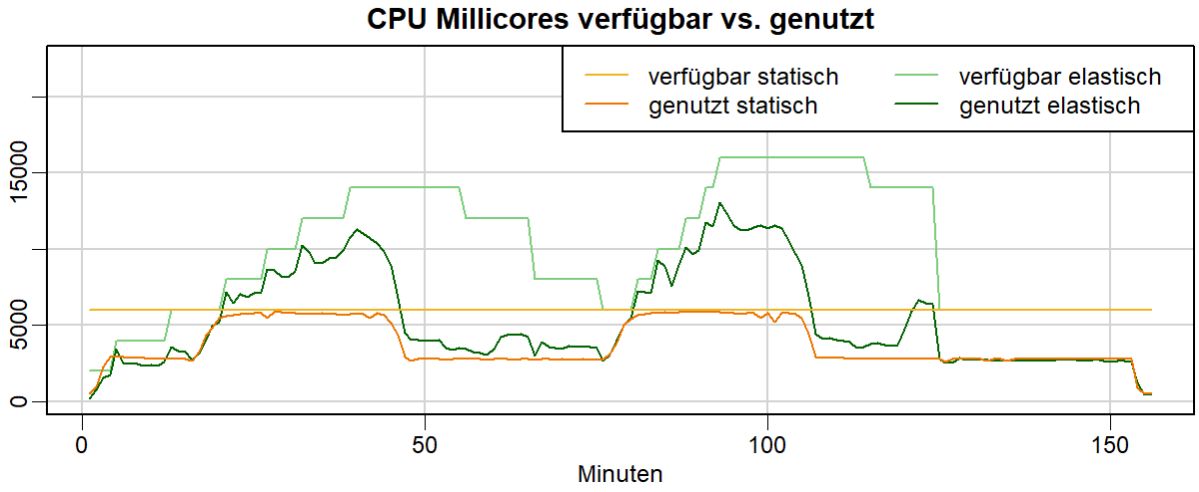


Abbildung 4-24 Sinus-Auslastung mit 150 virtuellen Benutzern und 15 Mandanten: Diagramm 2: verfügbare und genutzte CPU (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

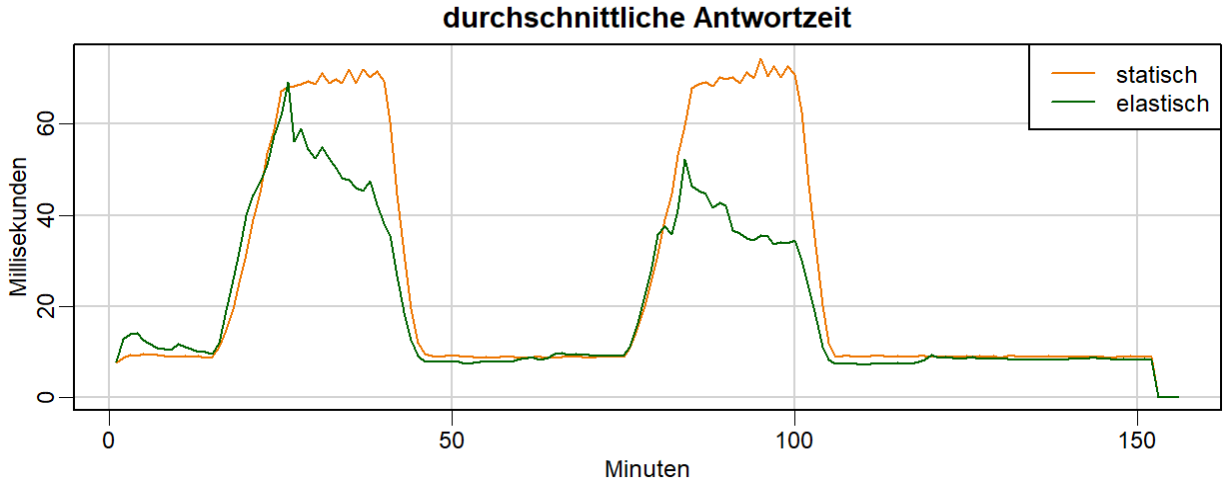


Abbildung 4-25 Sinus-Auslastung mit 150 virtuellen Benutzern und 15 Mandanten: Diagramm 3: Vergleich der durchschnittlichen Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

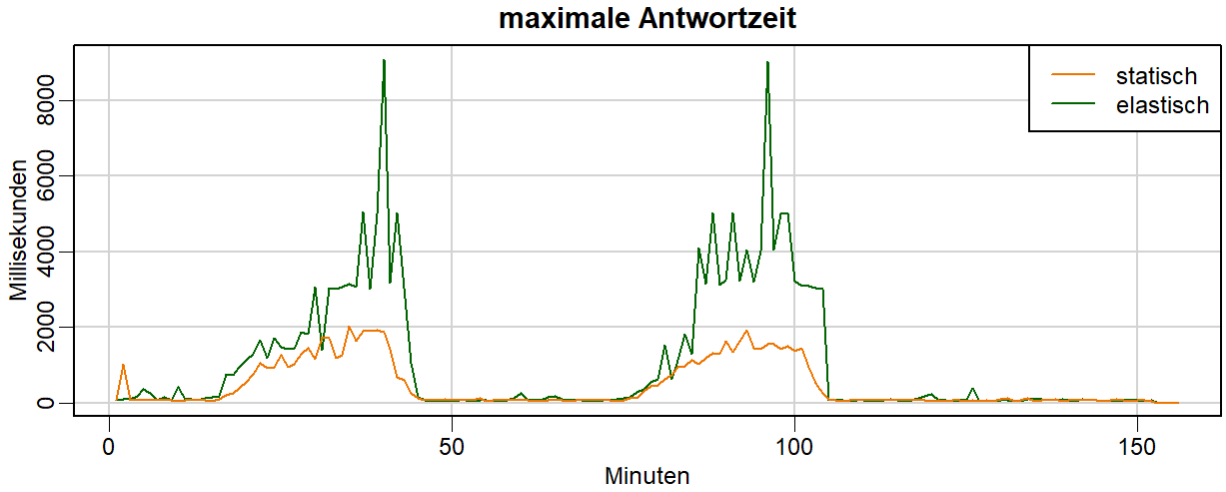


Abbildung 4-26 Sinus-Auslastung mit 150 virtuellen Benutzern und 15 Mandanten: Diagramm 4: Vergleich der maximalen Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

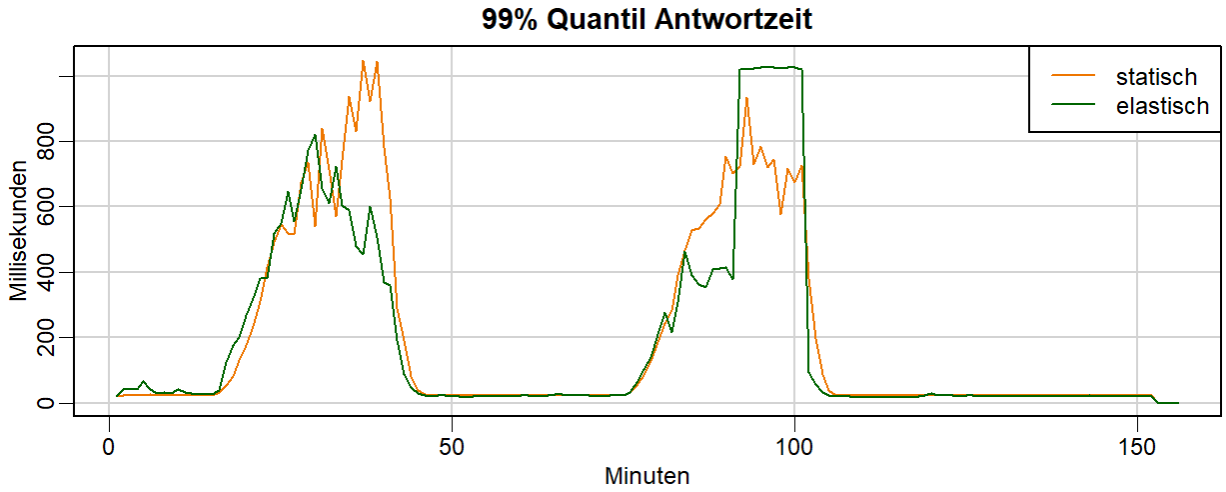


Abbildung 4-27 Sinus-Auslastung mit 150 virtuellen Benutzern und 15 Mandanten: Diagramm 5: Vergleich der 99% Quantile der Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

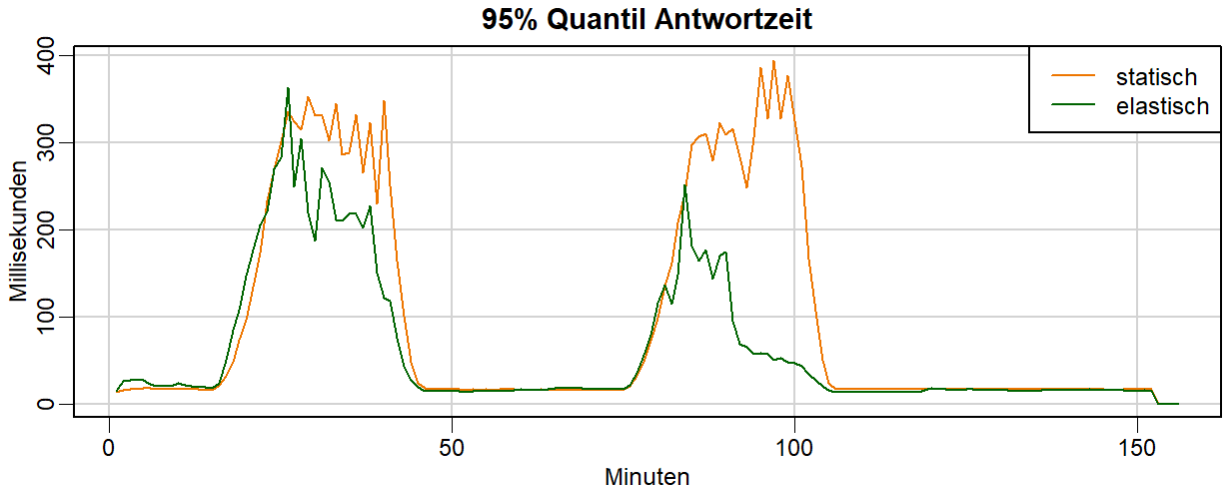


Abbildung 4-28 Sinus-Auslastung mit 150 virtuellen Benutzern und 15 Mandanten: Diagramm 6: Vergleich der 95% Quantile der Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

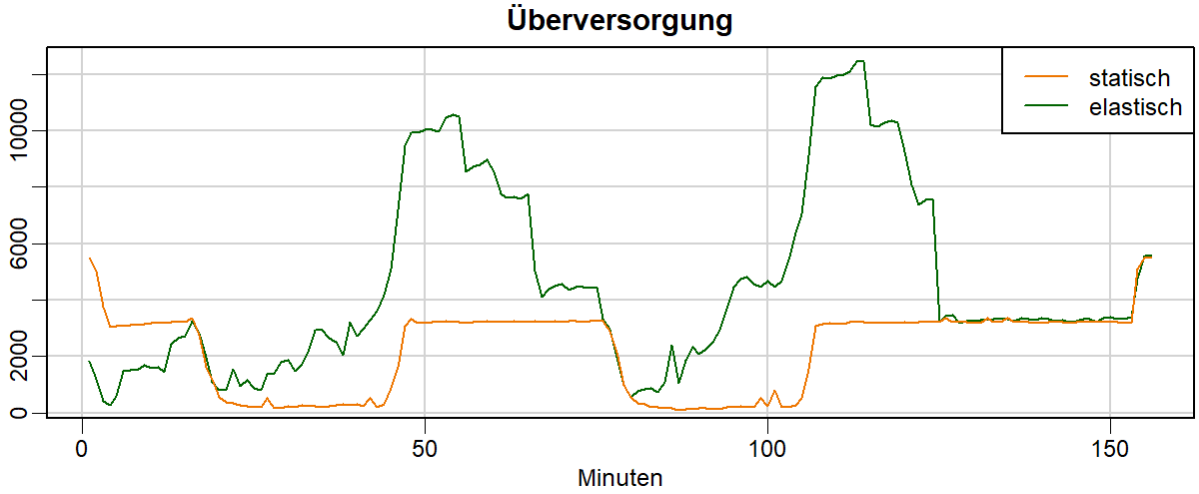


Abbildung 4-29 Sinus-Auslastung mit 150 virtuellen Benutzern und 15 Mandanten: Diagramm 7: Vergleich der Überversorgung (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

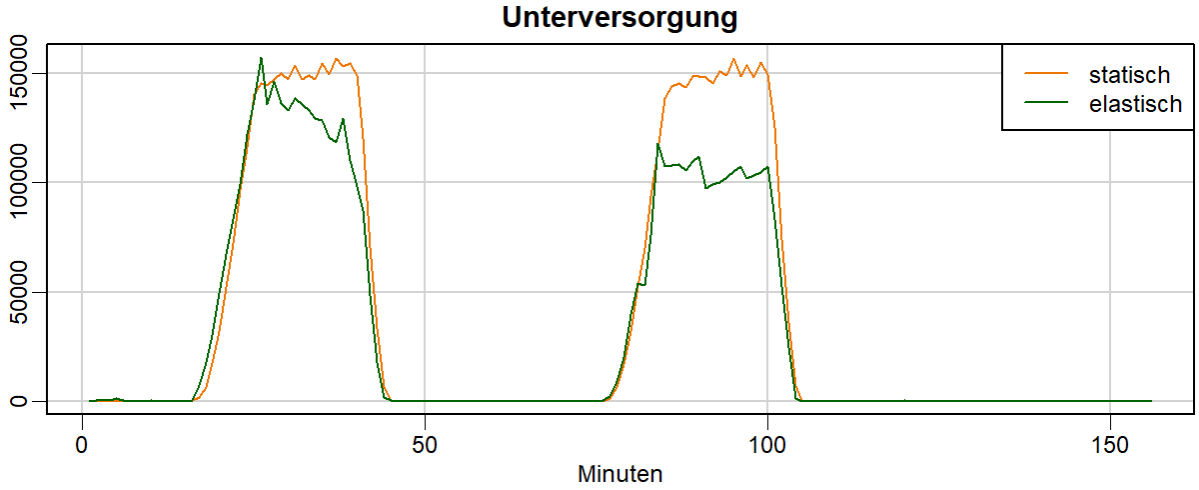


Abbildung 4-30 Sinus-Auslastung mit 150 virtuellen Benutzern und 15 Mandanten: Diagramm 8: Vergleich der Unterversorgung (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

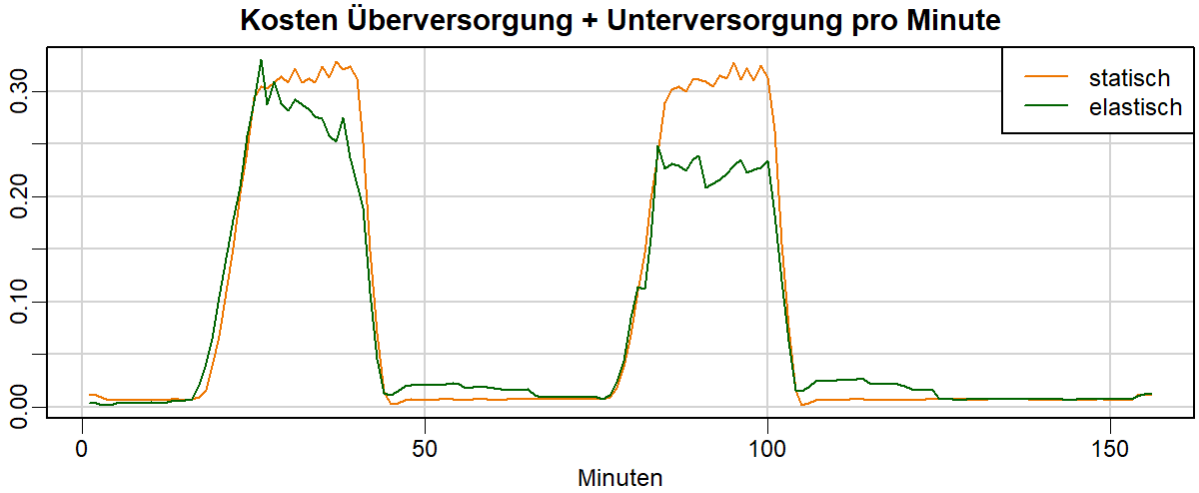


Abbildung 4-31 Sinus-Auslastung mit 150 virtuellen Benutzern und 15 Mandanten: Diagramm 9: Vergleich der Kosten für Überversorgung + Unterversorgung (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

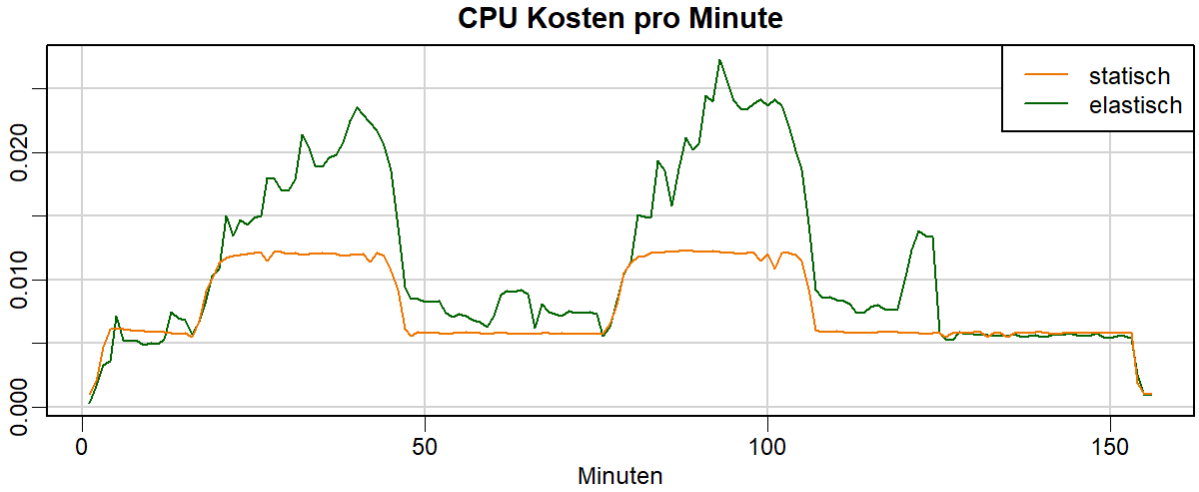


Abbildung 4-32 Sinus-Auslastung mit 150 virtuellen Benutzern und 15 Mandanten: Diagramm 10: Vergleich der Kosten der tatsächlich genutzten CPU-Kapazitäten (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

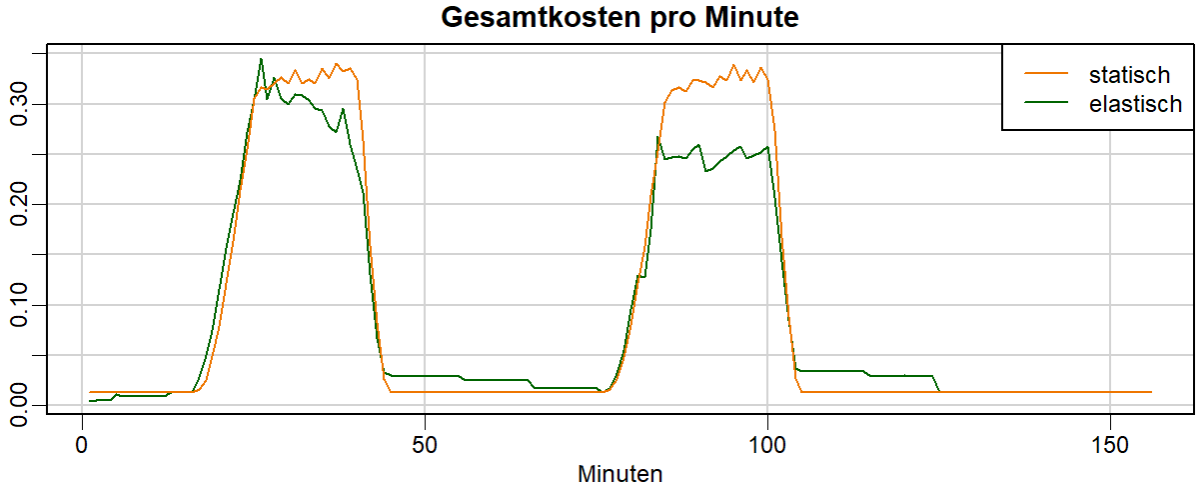


Abbildung 4-33 Sinus-Auslastung mit 150 virtuellen Benutzern und 15 Mandanten: Diagramm 11: Vergleich der Gesamtkosten (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

In den Diagrammen von Testlauf 3 ist der Zusammenhang zwischen steigenden Ressourcen und sinkenden Antwortzeiten besonders gut ersichtlich. In beiden Belastungsphasen wurden zu Beginn weniger verfügbare Ressourcen und höhere Antwortzeiten gemessen. Im Verlauf der fünfzehnminütigen Systembelastung ist deutlich die zusätzliche Bereitstellung von Ressourcen und die gleichzeitige Reduzierung der durchschnittlichen Antwortzeit zu erkennen. Dieses Verhalten spiegelt sich in allen anderen Messwerten des Testlaufes, von der Anzahl der Anfragen, über die Antwortzeit-Quantile bis zu den Kosten, wider. Auffallend dabei ist, dass das elastische System in der zweiten Belastungsphase aus den übriggebliebenen Ressourcen der ersten Belastungsphase profitiert. So sind die durchschnittlichen Antwortzeiten und damit auch die Unterversorgung und dessen Kosten geringer als in der ersten.

4.1.4 Testlauf 4: Sinus - 250 Benutzer - 25 Mandanten - 5 VM statisch

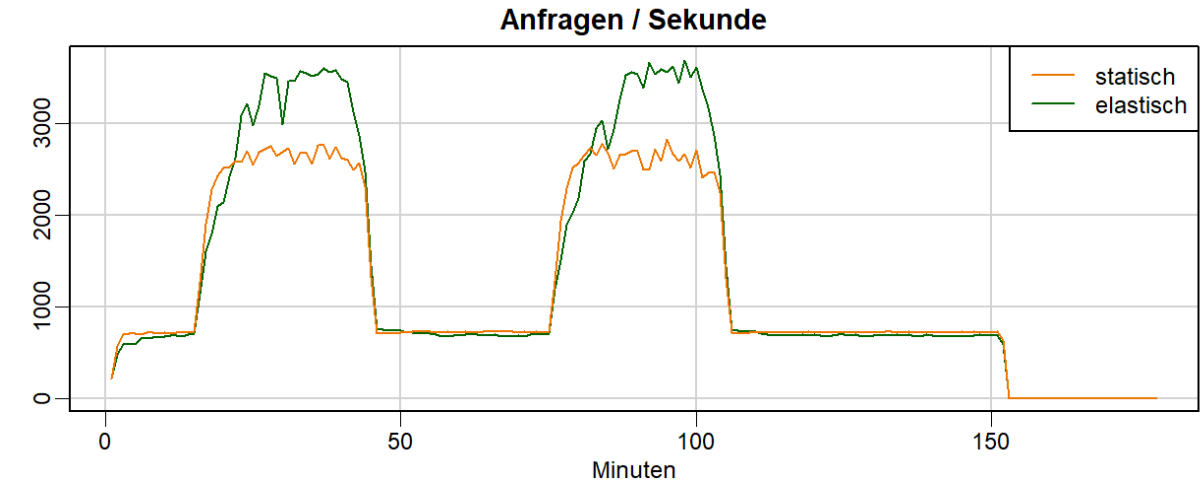


Abbildung 4-34 Sinus-Auslastung mit 250 virtuellen Benutzern und 25 Mandanten: Diagramm 1: Vergleich der Anfragen pro Sekunde (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

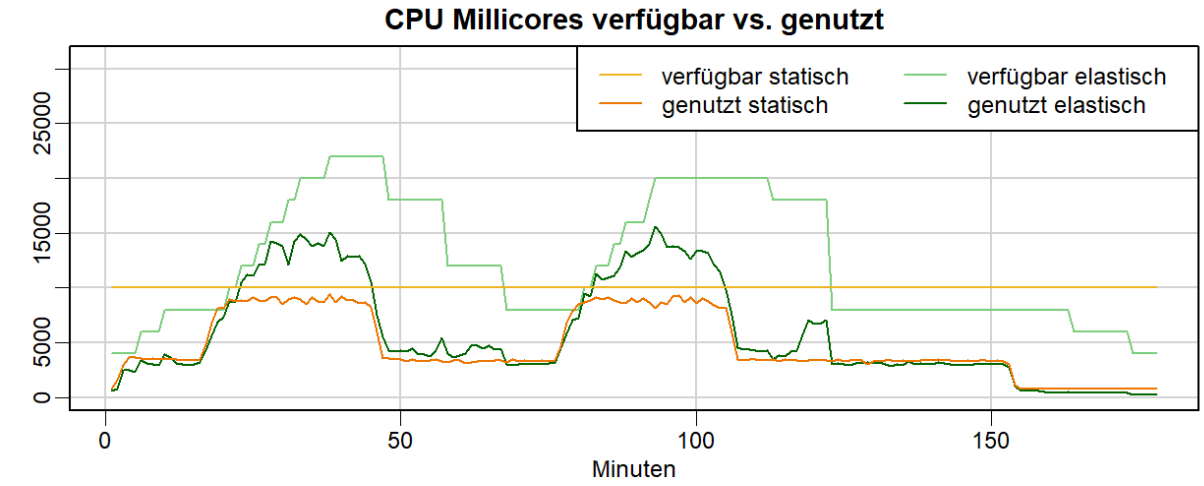


Abbildung 4-35 Sinus-Auslastung mit 250 virtuellen Benutzern und 25 Mandanten: Diagramm 2: verfügbare und genutzte CPU (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

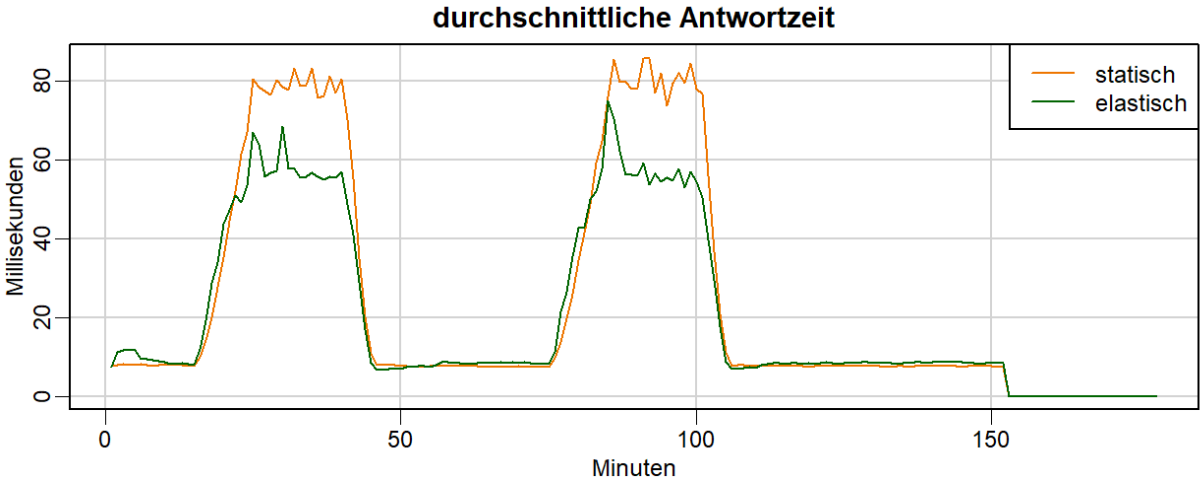


Abbildung 4-36 Sinus-Auslastung mit 250 virtuellen Benutzern und 25 Mandanten: Diagramm 3: Vergleich der durchschnittlichen Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

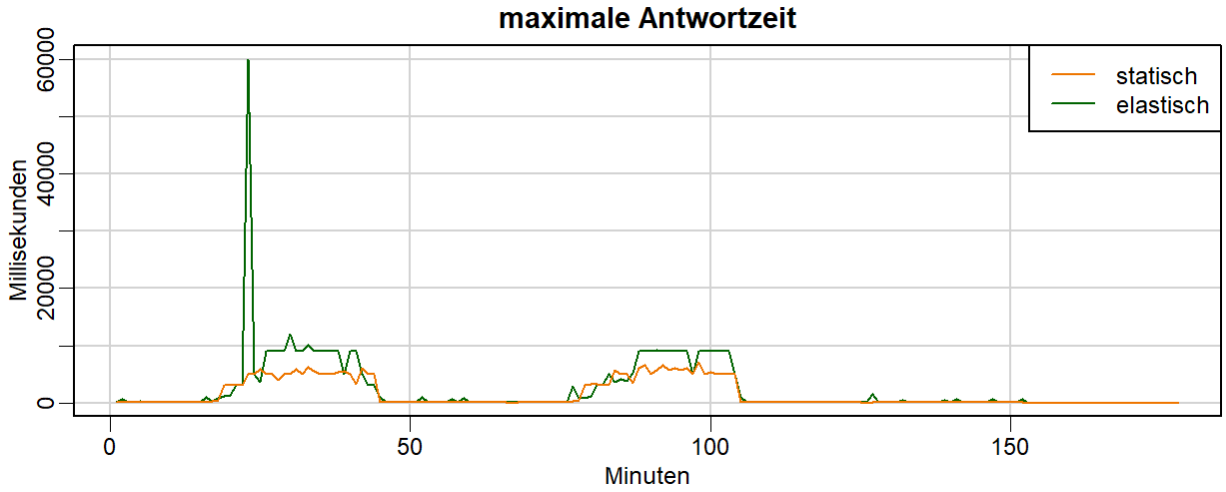


Abbildung 4-37 Sinus-Auslastung mit 250 virtuellen Benutzern und 25 Mandanten: Diagramm 4: Vergleich der maximalen Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

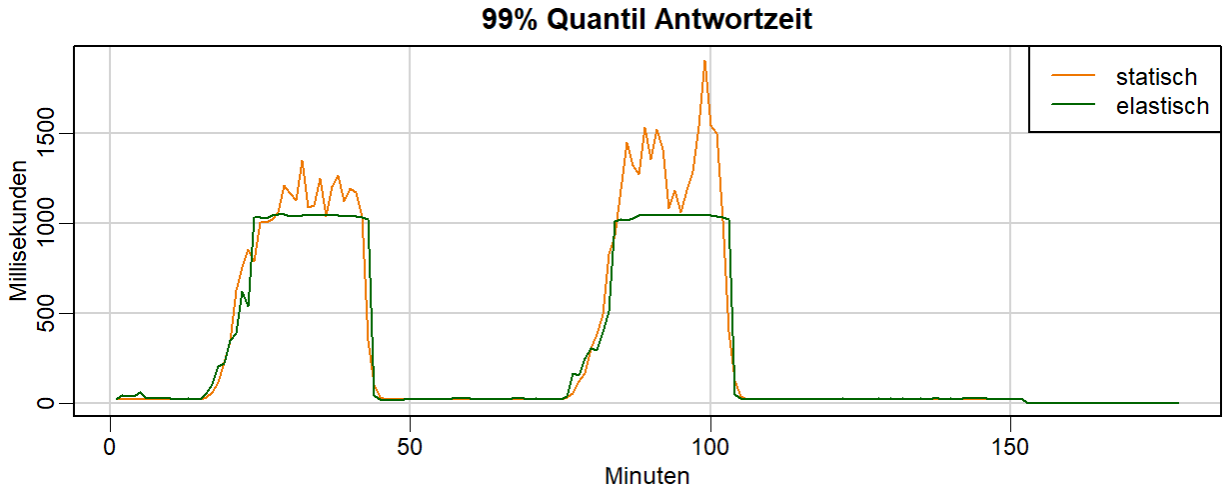


Abbildung 4-38 Sinus-Auslastung mit 250 virtuellen Benutzern und 25 Mandanten: Diagramm 5: Vergleich der 99% Quantile der Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

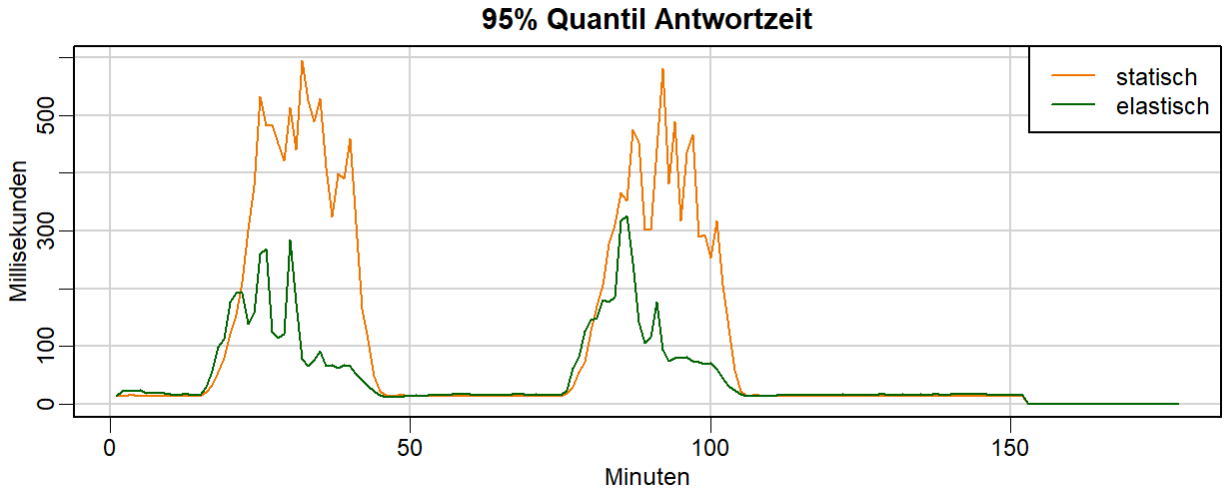


Abbildung 4-39 Sinus-Auslastung mit 250 virtuellen Benutzern und 25 Mandanten: Diagramm 6: Vergleich der 95% Quantile der Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

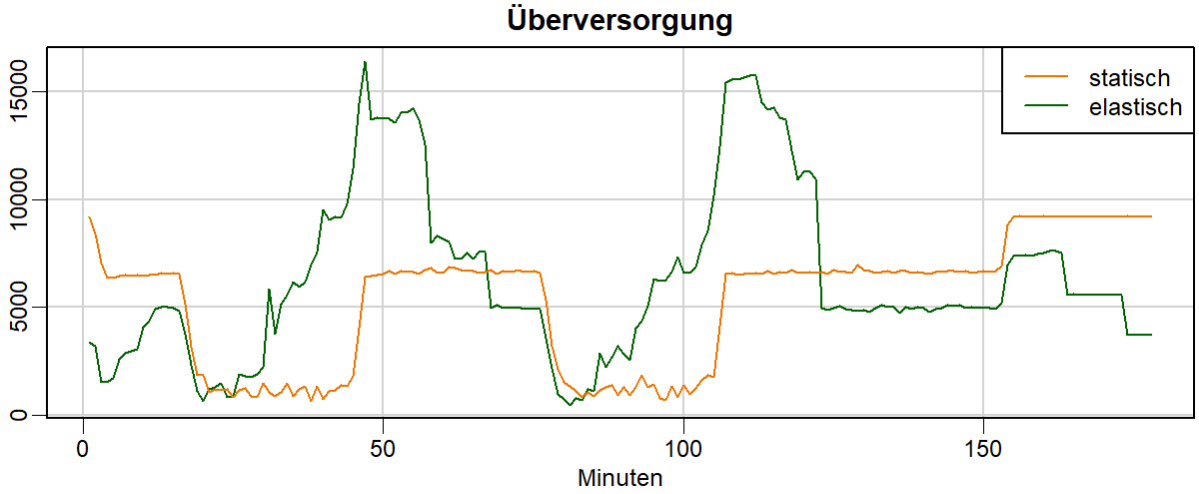


Abbildung 4-40 Sinus-Auslastung mit 250 virtuellen Benutzern und 25 Mandanten: Diagramm 7: Vergleich der Überversorgung (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

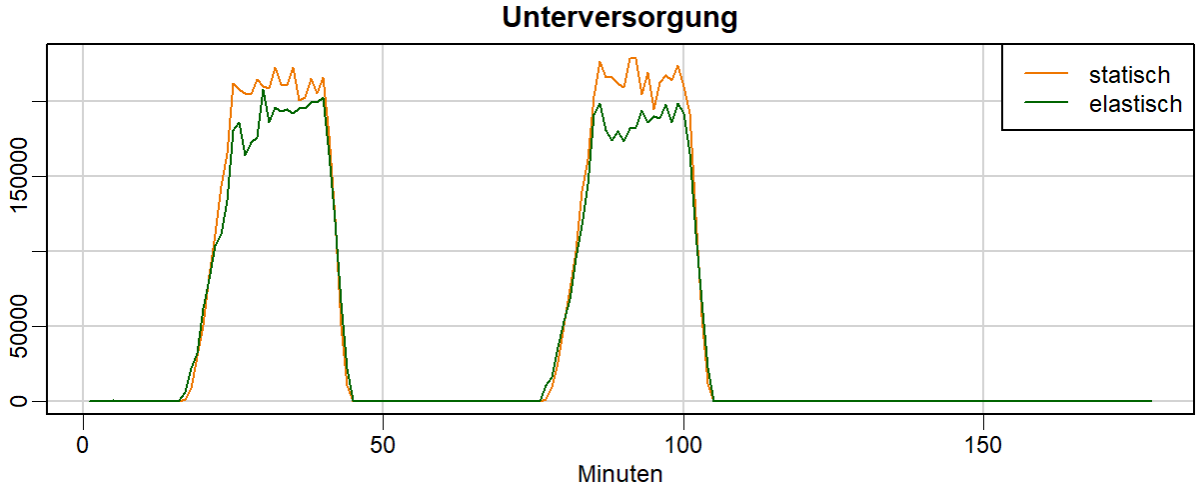


Abbildung 4-41 Sinus-Auslastung mit 250 virtuellen Benutzern und 25 Mandanten: Diagramm 8: Vergleich der Unterversorgung (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

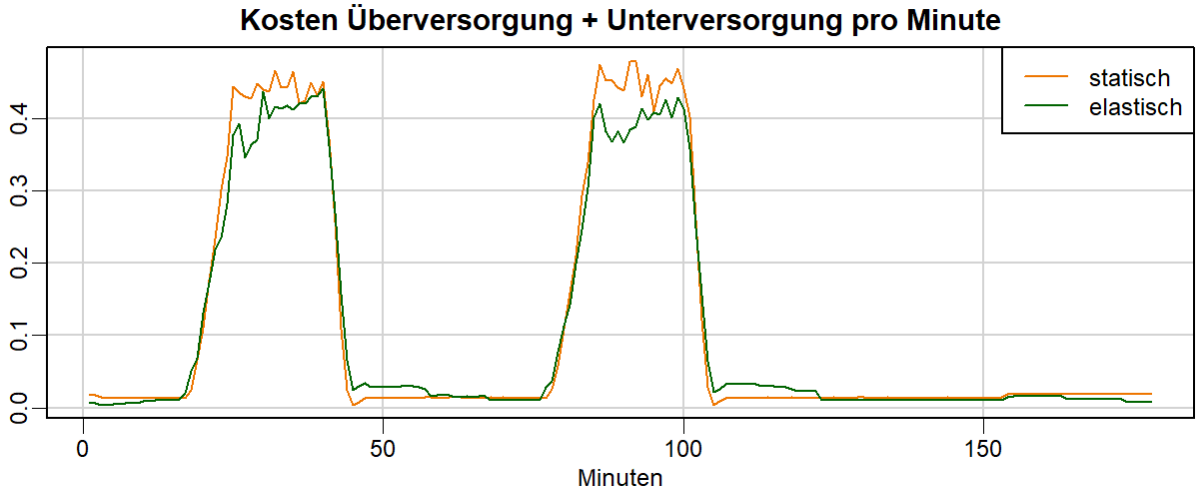


Abbildung 4-42 Sinus-Auslastung mit 250 virtuellen Benutzern und 25 Mandanten: Diagramm 9: Vergleich der Kosten für Überversorgung + Unterversorgung (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

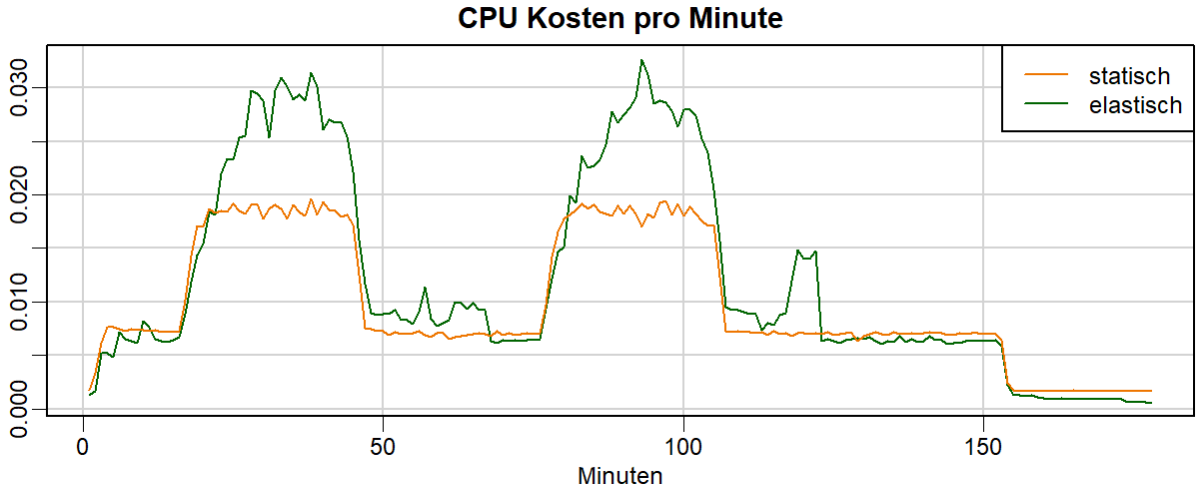


Abbildung 4-43 Sinus-Auslastung mit 250 virtuellen Benutzern und 25 Mandanten: Diagramm 10: Vergleich der Kosten der tatsächlich genutzten CPU-Kapazitäten (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

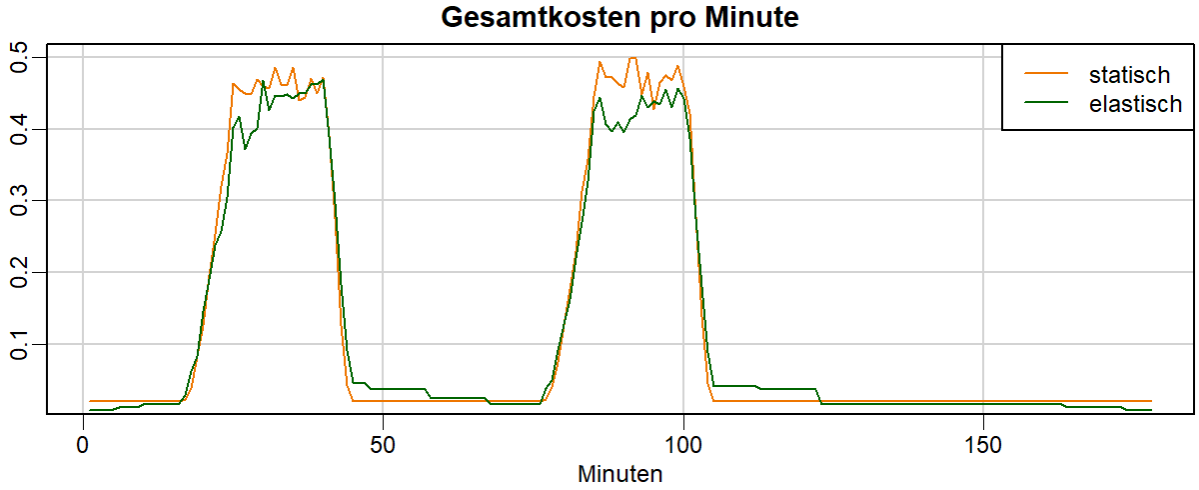


Abbildung 4-44 Sinus-Auslastung mit 250 virtuellen Benutzern und 25 Mandanten: Diagramm 11: Vergleich der Gesamtkosten (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

Im Testlauf 4 reagierte das elastische Testsystem auf die steigende Workload ebenso gut mit der Bereitstellung zusätzlicher Ressourcen. Allerdings scheint es, als ob die verfügbaren Ressourcen nicht effizient genutzt werden konnten. Anders als im Testlauf 3 wurde nicht so eine deutliche Senkung der durchschnittlichen Antwortzeit gemessen und es ist in der zweiten Belastungsphase auch kein Ressourcenvorteil aus der ersten erkennbar. Im Diagramm der maximalen Antwortzeit ist besonders auffällig, dass nicht nur die Kurve des elastischen Testsystems über der des statischen Systems liegt, sondern dass zu Beginn der ersten Belastungsphase im elastischen System auch ein extremer Ausreißer bei ungefähr 60 Sekunden gemessen wurde.

4.1.5 Testlauf 5: Auslastungsspitze 360 Benutzer - 15 Mandanten – 3 VM statisch

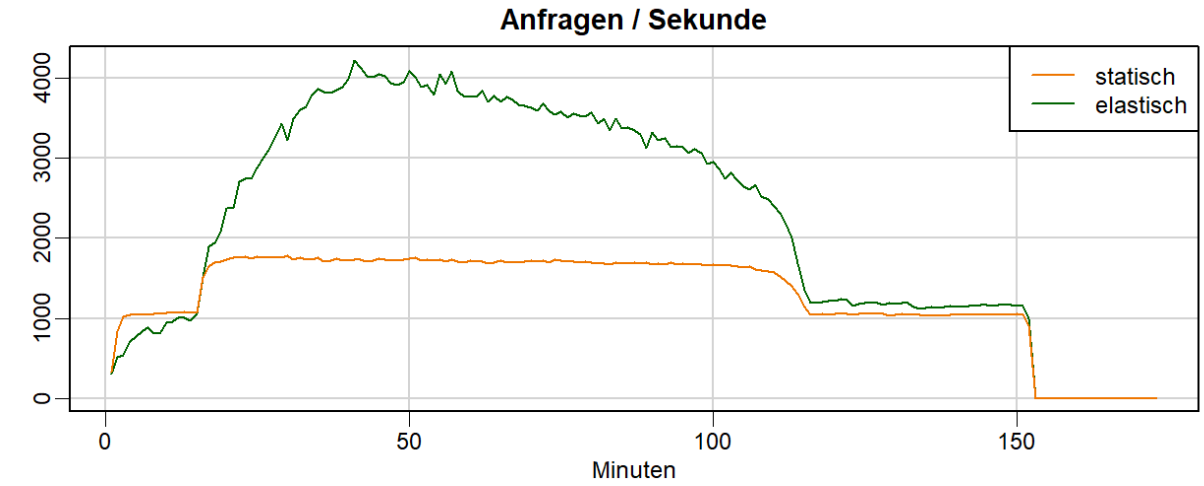


Abbildung 4-45 Auslastungsspitze mit 360 virtuellen Benutzern und 15 Mandanten: Diagramm 1: Vergleich der Anfragen pro Sekunde (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

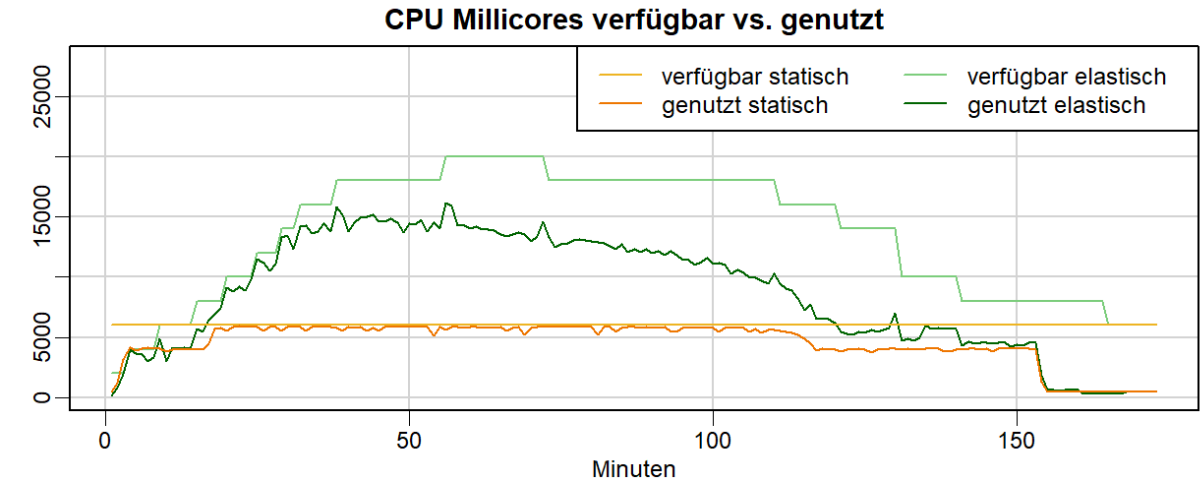


Abbildung 4-46 Auslastungsspitze mit 360 virtuellen Benutzern und 15 Mandanten: Diagramm 2: verfügbare und genutzte CPU (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

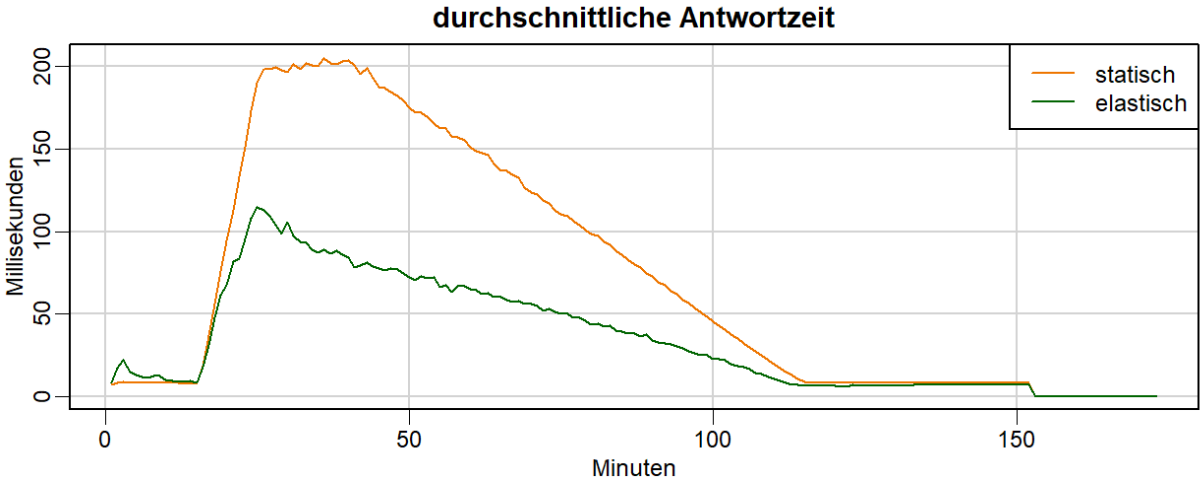


Abbildung 4-47 Auslastungsspitze mit 360 virtuellen Benutzern und 15 Mandanten: Diagramm 3: Vergleich der durchschnittlichen Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

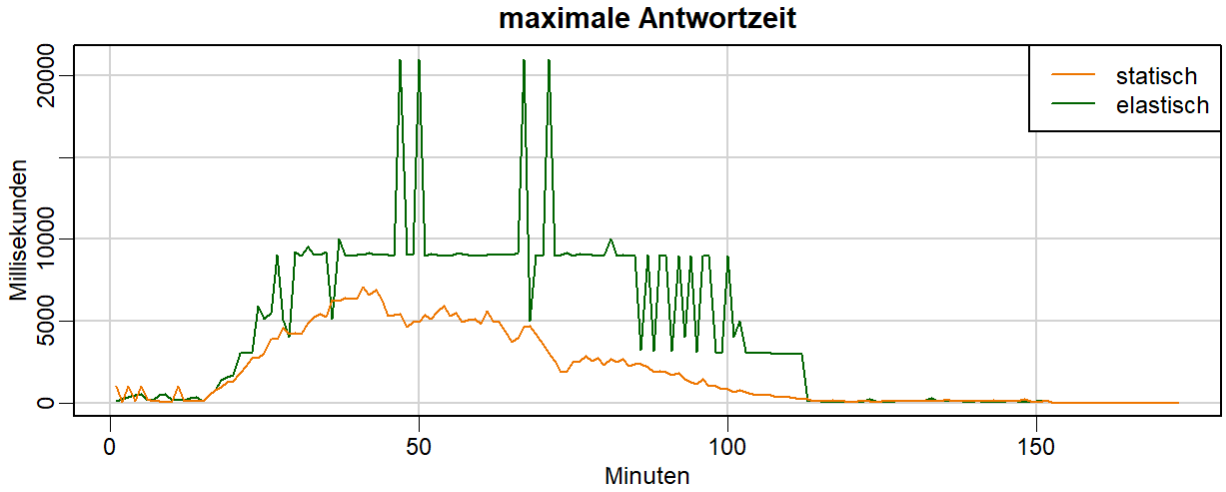


Abbildung 4-48 Auslastungsspitze mit 360 virtuellen Benutzern und 15 Mandanten: Diagramm 4: Vergleich der maximalen Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

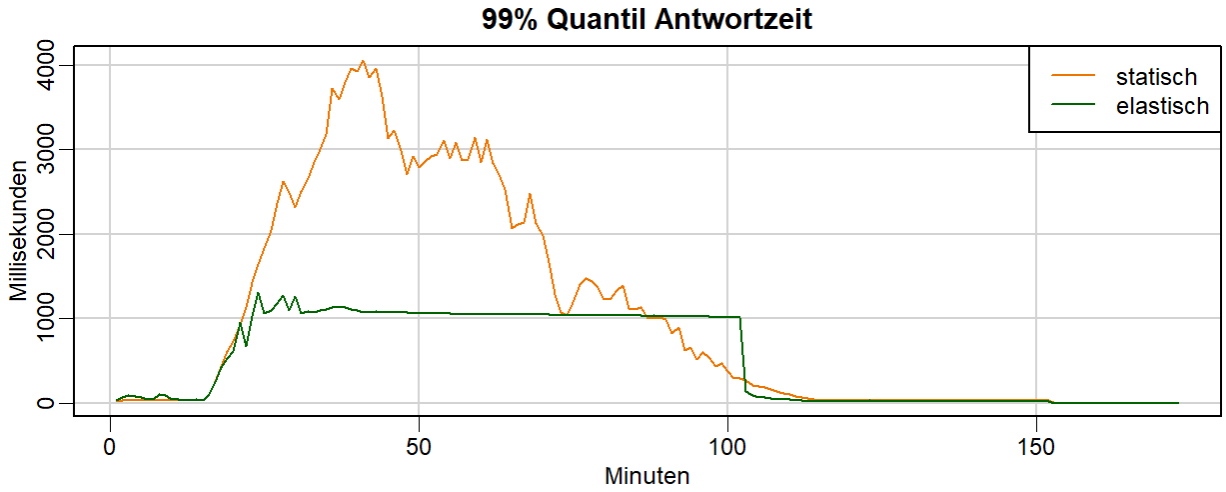


Abbildung 4-49 Auslastungsspitze mit 360 virtuellen Benutzern und 15 Mandanten: Diagramm 5: Vergleich der 99% Quantile der Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

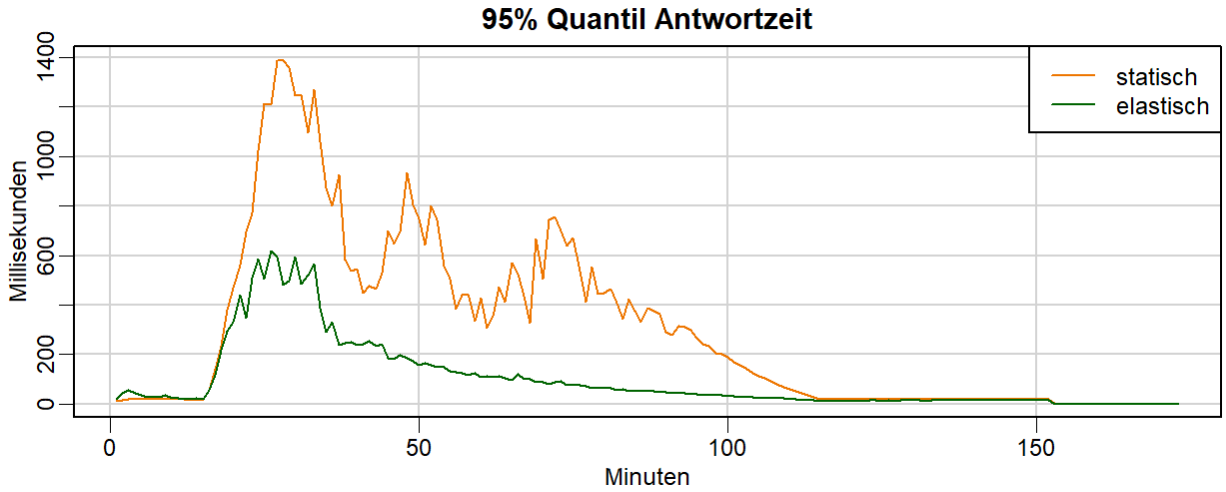


Abbildung 4-50 Auslastungsspitze mit 360 virtuellen Benutzern und 15 Mandanten: Diagramm 6: Vergleich der 95% Quantile der Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

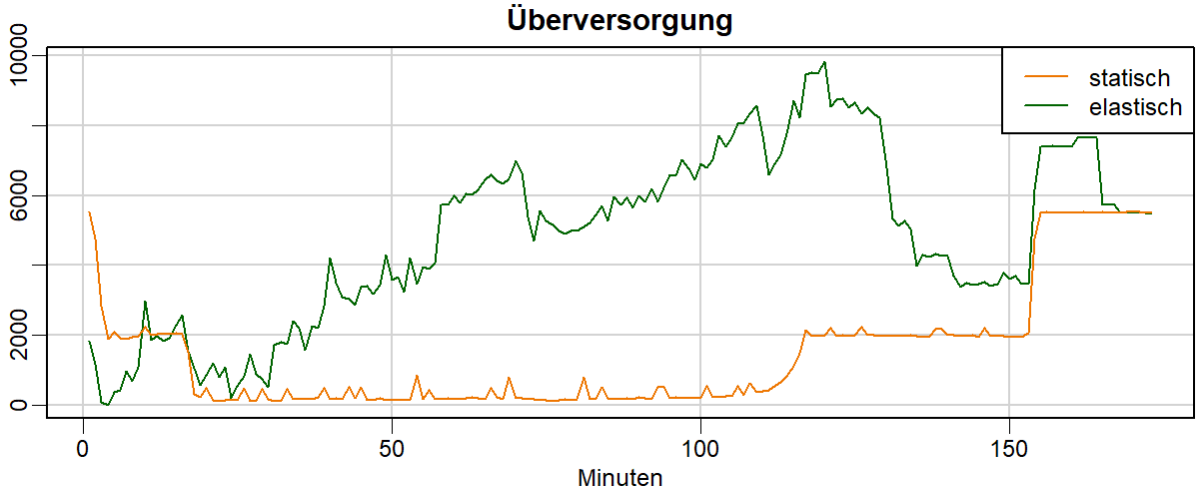


Abbildung 4-51 Auslastungsspitze mit 360 virtuellen Benutzern und 15 Mandanten: Diagramm 7: Vergleich der Überversorgung (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

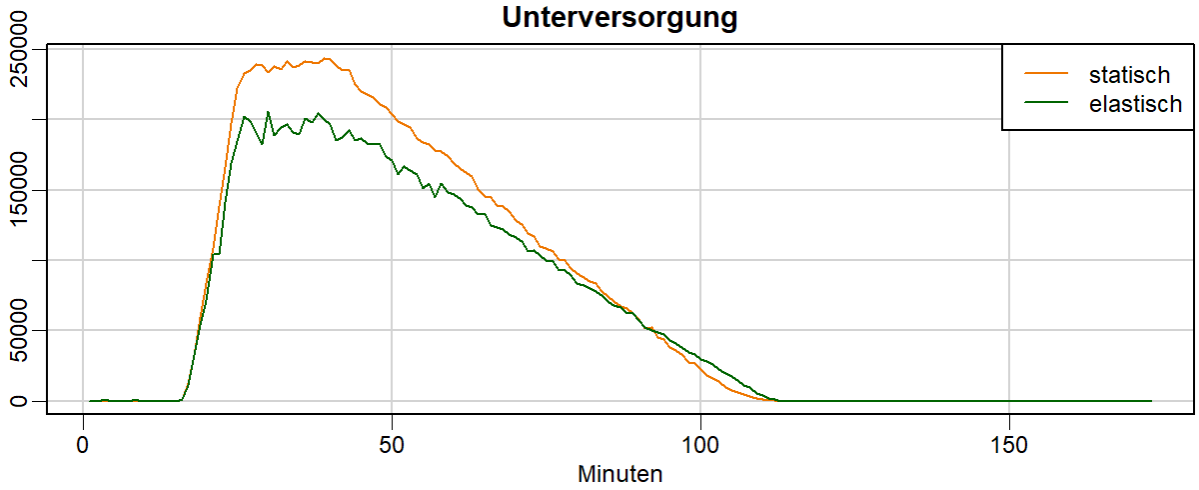


Abbildung 4-52 Auslastungsspitze mit 360 virtuellen Benutzern und 15 Mandanten: Diagramm 8: Vergleich der Unterversorgung (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

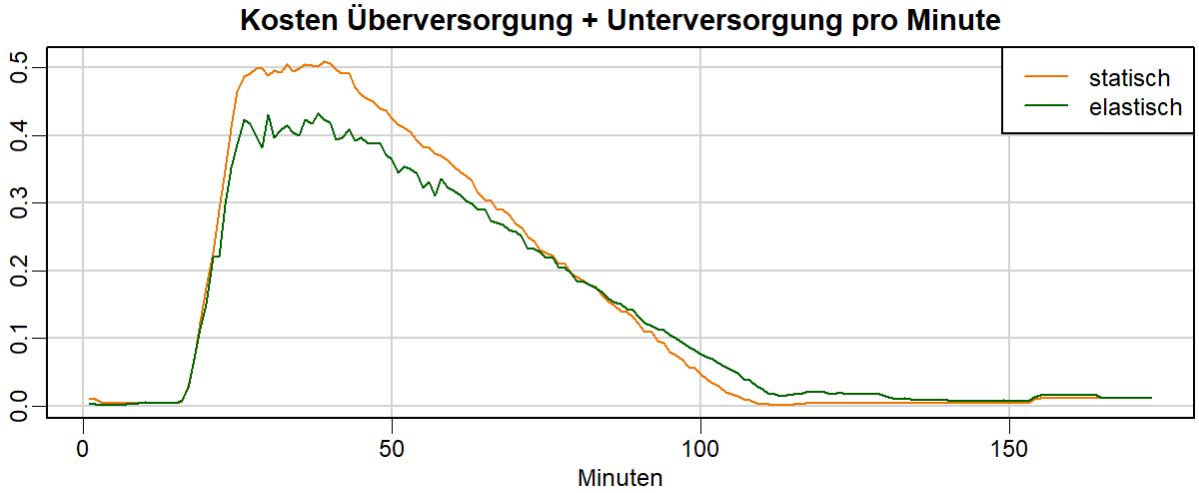


Abbildung 4-53 Auslastungsspitze mit 360 virtuellen Benutzern und 15 Mandanten: Diagramm 9: Vergleich der Kosten für Überversorgung + Unterversorgung (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

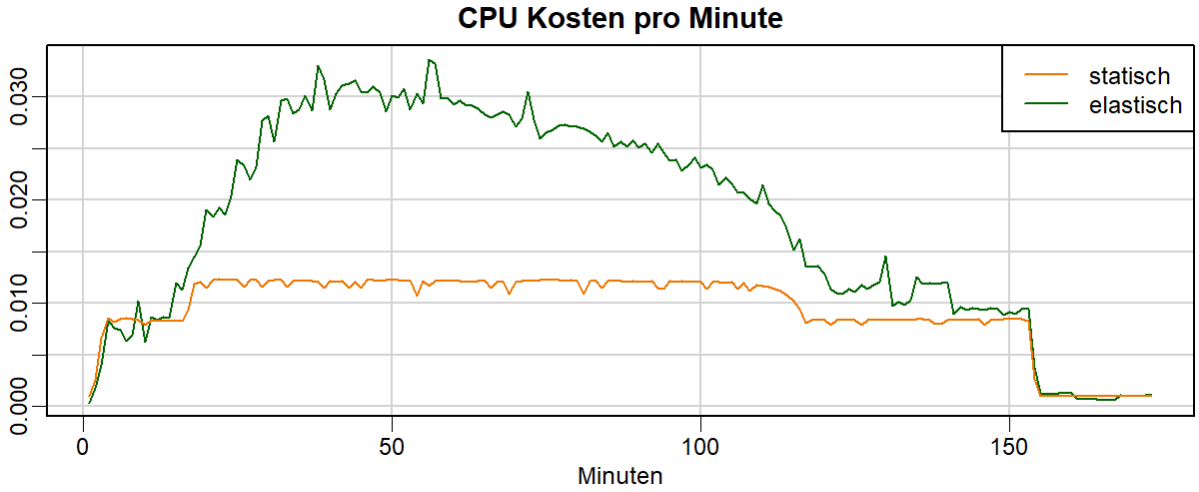


Abbildung 4-54 Auslastungsspitze mit 360 virtuellen Benutzern und 15 Mandanten: Diagramm 10: Vergleich der Kosten der tatsächlich genutzten CPU-Kapazitäten (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

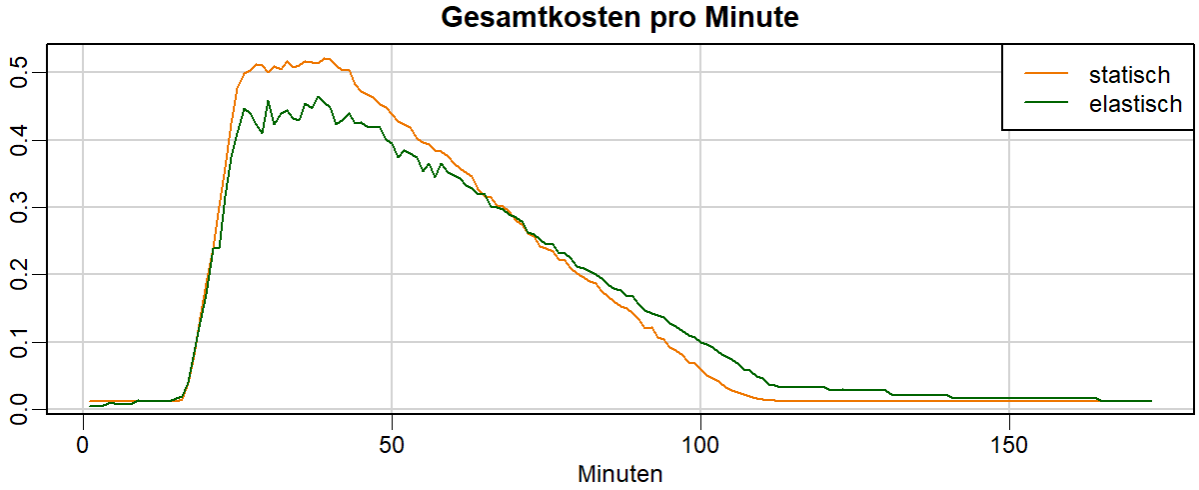


Abbildung 4-55 Auslastungsspitze mit 360 virtuellen Benutzern und 15 Mandanten: Diagramm 11: Vergleich der Gesamtkosten (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

In den Testläufen 5 und 6 wurden die Systeme den höchsten Workloads ausgesetzt, wodurch hier auch entsprechend lange Antwortzeiten gemessen wurden. In den meisten anderen Diagrammen verhalten sich die Messwerte wie erwartet und ähnlich den vorangegangenen Testläufen. Auffällig ist jedoch, dass die Ressourcen im System weniger schnell wieder freigegeben wurden, als die anfallende Workload gesunken ist. Dadurch ergaben sich im Verlauf steigende Überversorgungskosten, welche schlussendlich auch eine entsprechende Auswirkung auf die Gesamtkosten mit sich brachten.

4.1.6 Testlauf 6: Auslastungsspitze 600 Benutzer - 25 Mandanten – 5 VM statisch

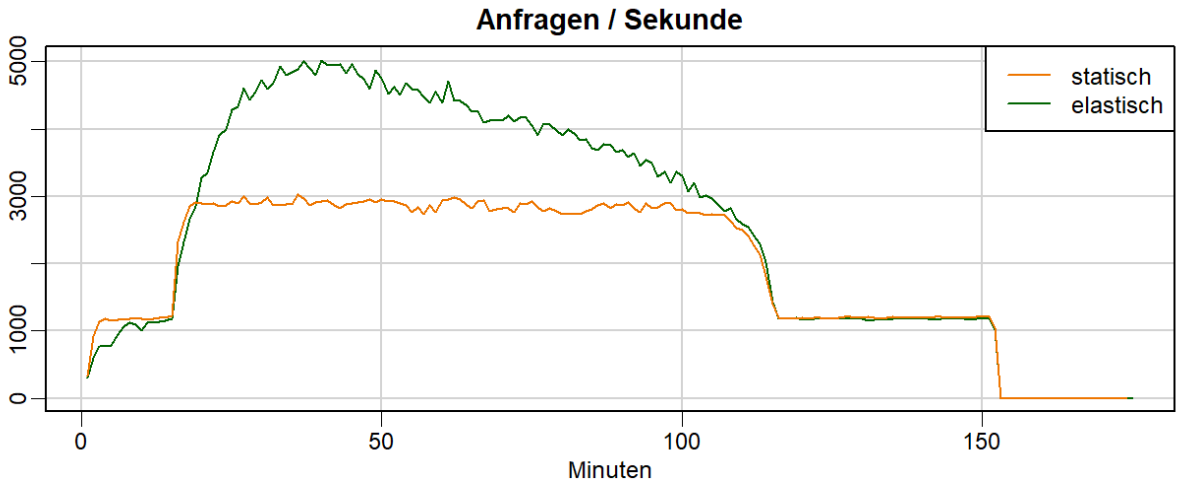


Abbildung 4-56 Auslastungsspitze mit 600 virtuellen Benutzern und 25 Mandanten: Diagramm 1: Vergleich der Anfragen pro Sekunde (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

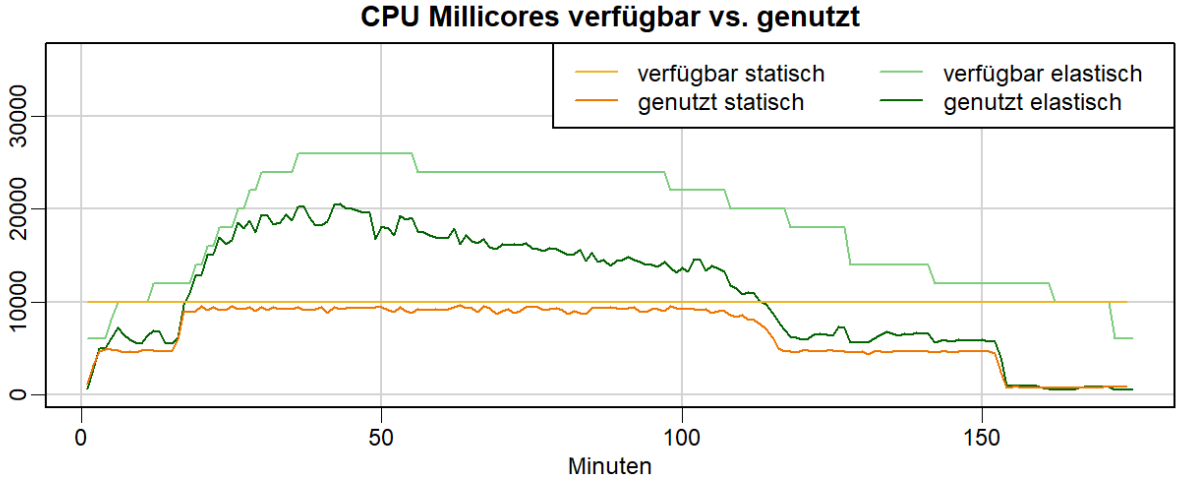


Abbildung 4-57 Auslastungsspitze mit 600 virtuellen Benutzern und 25 Mandanten: Diagramm 2: verfügbare und genutzte CPU (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

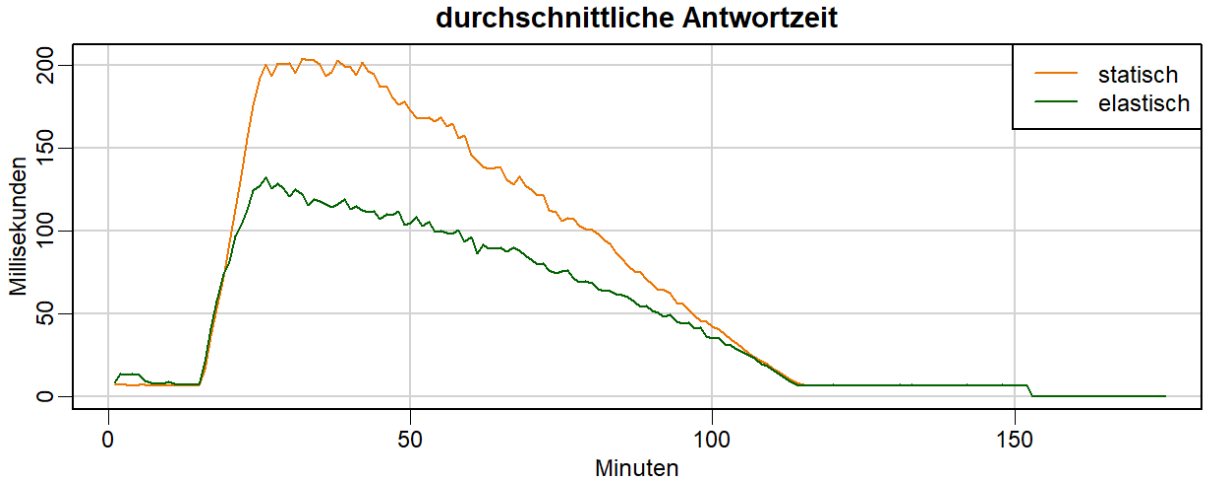


Abbildung 4-58 Auslastungsspitze mit 600 virtuellen Benutzern und 25 Mandanten: Diagramm 3: Vergleich der durchschnittlichen Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

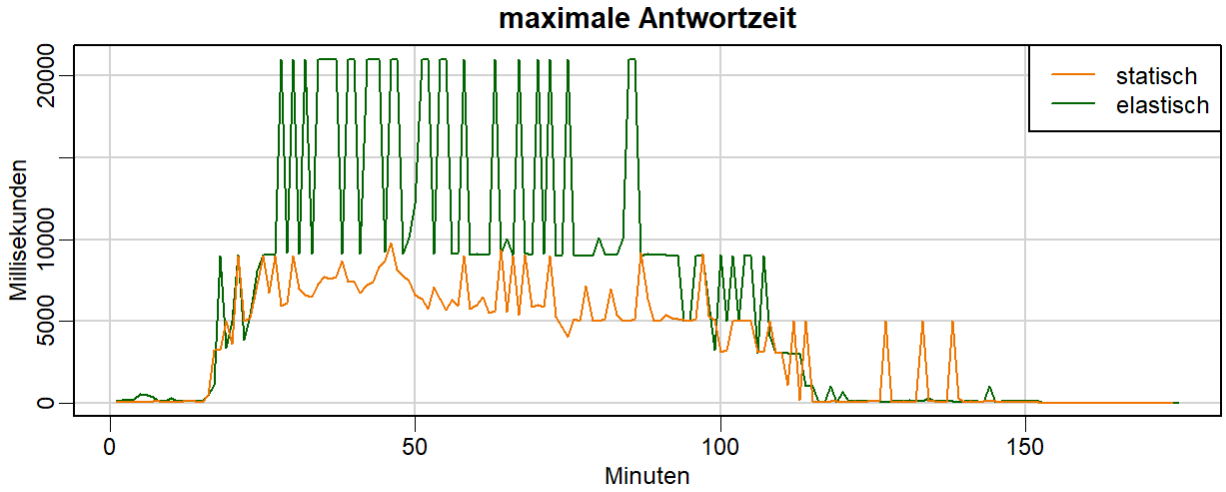


Abbildung 4-59 Auslastungsspitze mit 600 virtuellen Benutzern und 25 Mandanten: Diagramm 4: Vergleich der maximalen Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

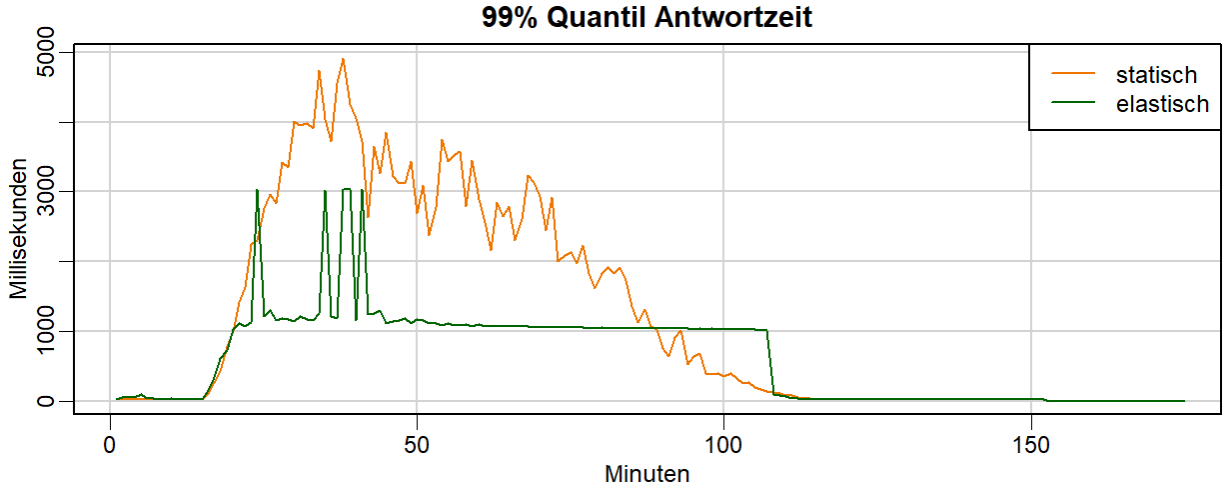


Abbildung 4-60 Auslastungsspitze mit 600 virtuellen Benutzern und 25 Mandanten: Diagramm 5: Vergleich der 99% Quantile der Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

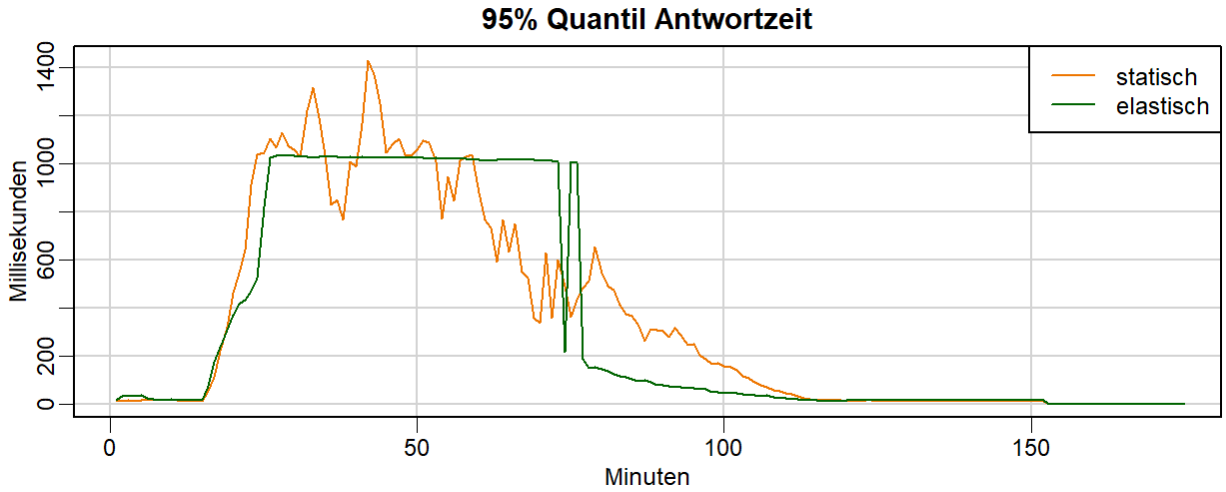


Abbildung 4-61 Auslastungsspitze mit 600 virtuellen Benutzern und 25 Mandanten: Diagramm 6: Vergleich der 95% Quantile der Antwortzeit (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

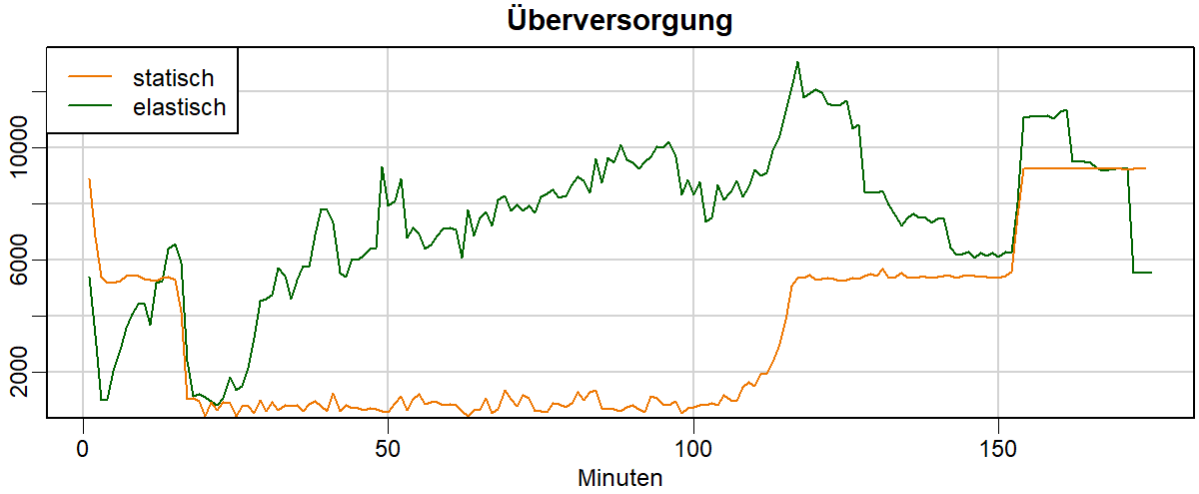


Abbildung 4-62 Auslastungsspitze mit 600 virtuellen Benutzern und 25 Mandanten: Diagramm 7: Vergleich der Überversorgung (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

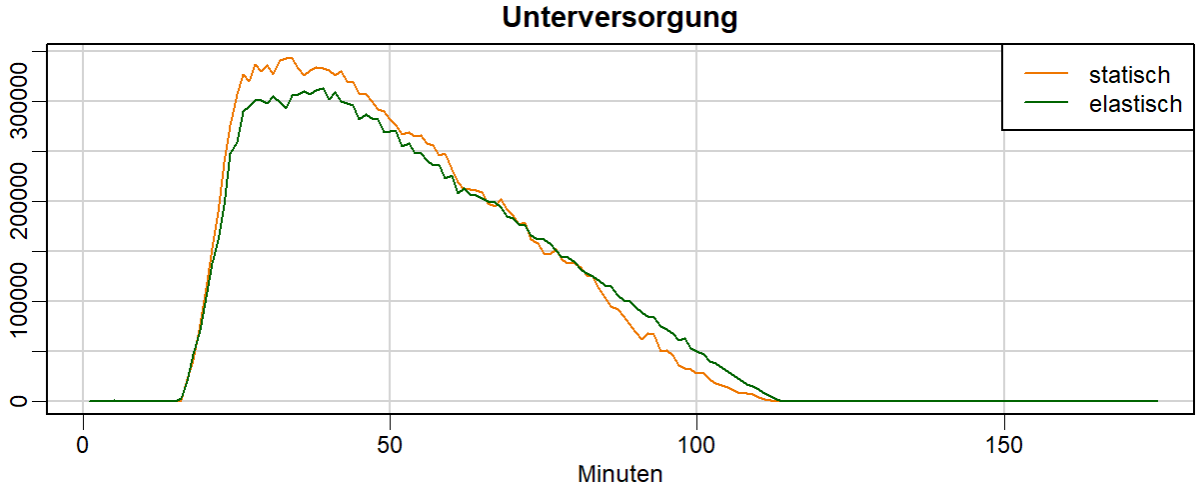


Abbildung 4-63 Auslastungsspitze mit 600 virtuellen Benutzern und 25 Mandanten: Diagramm 8: Vergleich der Unterversorgung (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

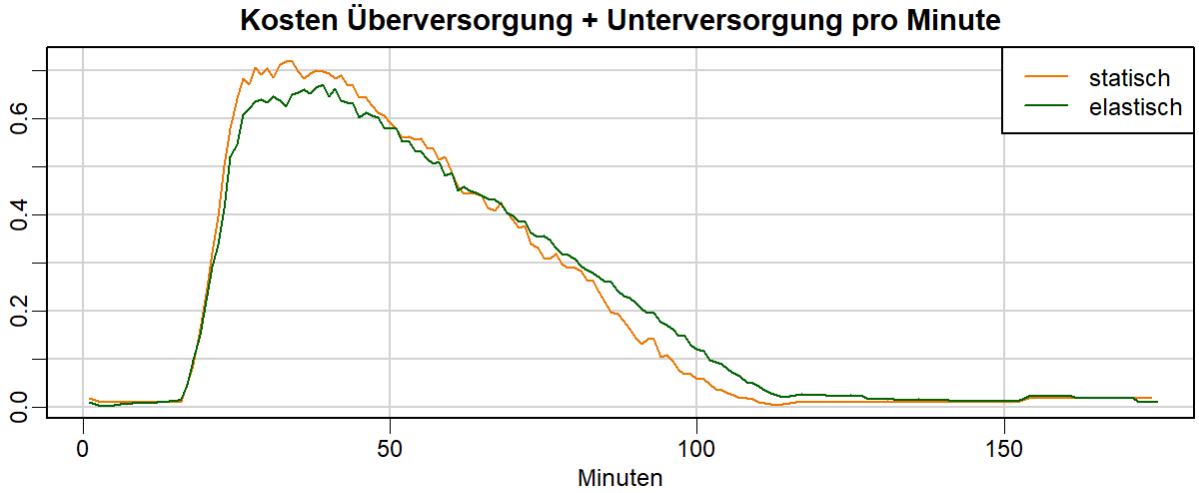


Abbildung 4-64 Auslastungsspitze mit 600 virtuellen Benutzern und 25 Mandanten: Diagramm 9: Vergleich der Kosten für Überversorgung + Unterversorgung (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

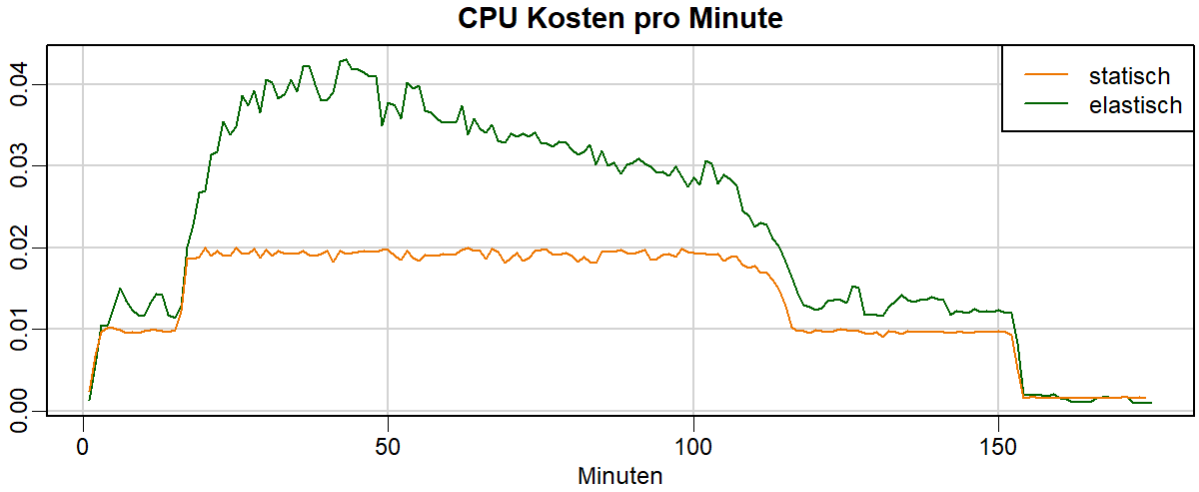


Abbildung 4-65 Auslastungsspitze mit 600 virtuellen Benutzern und 25 Mandanten: Diagramm 10: Vergleich der Kosten der tatsächlich genutzten CPU-Kapazitäten (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

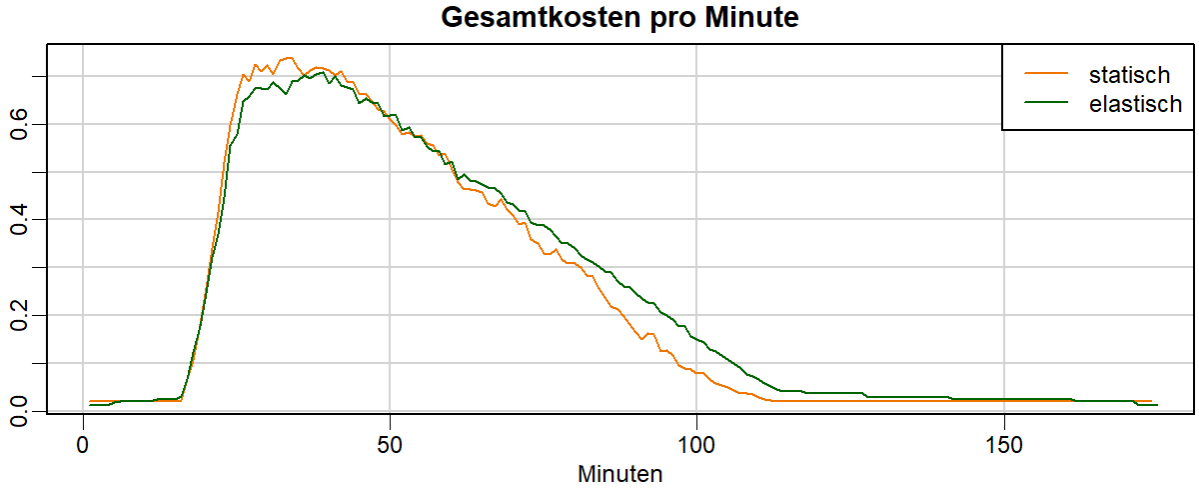


Abbildung 4-66 Auslastungsspitze mit 600 virtuellen Benutzern und 25 Mandanten: Diagramm 11: Vergleich der Gesamtkosten (Quelle: eigene Darstellung mit RStudio Version 1.1.383)

Im Testlauf 6 folgen die Messwerte ähnlichen Verläufen wie im Testlauf 5, sodass diese erhöhte Antwortzeiten und Überversorgungskosten aufweisen. Passend zur größten Workload wurden in diesem Testlauf auch die höchsten verfügbaren und genutzten Ressourcen gemessen. Ansonsten ist zwischen dem Testlauf 5 und 6 eine deutliche Erhöhung der bearbeiteten Anfragen im statischen Testsystem zu verzeichnen, jedoch eine weniger starke Erhöhung im elastischen. Diese Werte passen auch zu der wesentlichen Ressourcenerhöhung im statischen Testsystem von 6.000 auf 10.000 Millicores und dem anteilsmäßig weniger hohen Zuwachs von maximal 20.000 auf maximal 26.000 Millicores im elastischen System.

4.2 Auswertung

Auf Basis der in Kapitel 4.1 dargestellten Ergebnisse werden pro System und Testlauf die durchschnittlichen Werte zu den Performanz- und Kosten-Metriken errechnet. Diese Mittelwerte werden gegenübergestellt sowie der prozentuale Unterschied zwischen dem statischen und elastischen System errechnet. Auf diese Weise werden unter anderem die Vergleichswerte für Performanz, Elastizität und Kosten zum jeweiligen Testlauf errechnet.

Performanz: Das Verhältnis der durchschnittlichen Antwortzeit $\bar{r}$ beschreibt den Vergleichswert der Performanz $P_{S_1,\ w_x}$ für das jeweilige Auslastungsmuster w_x. Ein Wert kleiner als 1 bedeutet eine Performanz-Verbesserung im elastischen System, ein Wert größer als 1 bedeutet eine Verschlechterung.

Elastizität: Über das Verhältnis der durchschnittlichen Ressourcenabweichungskosten C_{o+u} wird der Vergleichswert der Elastizität $E_{S_1,\ w_x}$ für das jeweilige Auslastungsmuster w_x berechnet. Ein Wert kleiner als 1 weist auf geringere Ressourcenabweichungskosten im elastischen System gegenüber dem statischen System hin. Ein Wert größer als 1 bedeutet höhere Ressourcenabweichungskosten im elastischen System. Nach der verwendeten Definition und Berechnung ist die Elastizität umso besser, je geringer die Ressourcenabweichungskosten sind.

Kosten: Aus dem Verhältnis der durchschnittlichen Gesamtkosten C ergibt sich der Vergleichswert für die Gesamtkosten $T_{S_1,\ w_x}$ für das jeweilige Auslastungsmuster w_x. Ist der Wert kleiner als 1, so sind die Gesamtkosten im elastischen System geringer als im statischen System. Umgekehrt bedeutet ein Wert größer als 1 höhere Kosten im elastischen System.

In den nachfolgenden Auswertungstabellen sind die Ergebnisse abhängig von einer Verbesserung oder einer Verschlechterung im elastischen System eingefärbt. Grün hinterlegte Ergebnisse bedeuten eine Verbesserung der entsprechenden Metrik im elastischen System. Orange hinterlegte Ergebnisse weisen auf eine Verschlechterung im elastischen System hin.

4.2.1 Testlauf 1: Plateau - 240 Benutzer - 15 Mandanten - 3 VM statisch

	statisch	elastisch	Verhältnis elastisch/statisch
Performanz			
$\bar{r}$	88,97	51,79	**0,58**
Kosten			
C_d	0,0089	0,0172	1,93
C_o	0,0036	0,0123	3,42
C_u	0,1395	0,1269	0,91
$C_u\ /\ C_o$	38,83	10,33	-
C_{o+u}	0,1431	0,1392	**0,97**
C	0,1520	0,1564	**1,03**

Tabelle 4-2 Auswertung Performanz und Kosten: Plateau-Auslastung mit 240 virtuelle Benutzer und 15 Mandanten (Quelle: eigene Berechnungen)

4.2.2 Testlauf 2: Plateau - 240 Benutzer - 25 Mandanten - 5 VM statisch

	statisch	elastisch	Verhältnis elastisch/statisch
Performanz			
$\bar{r}$	55,74	49,09	**0,88**
Kosten			
C_d	0,0146	0,0195	1,34
C_o	0,0063	0,0144	2,29
C_u	0,1646	0,1931	1,17
C_u / C_o	26,07	13,37	-
C_{o+u}	0,1709	0,2075	**1,21**
C	0,1855	0,2271	**1,22**

Tabelle 4-3 Auswertung Performanz und Kosten: Plateau-Auslastung mit 240 virtuelle Benutzer und 25 Mandanten (Quelle: eigene Berechnungen)

4.2.3 Testlauf 3: Sinus - 150 Benutzer - 15 Mandanten - 3 VM statisch

	statisch	elastisch	Verhältnis elastisch/statisch
Performanz			
$\bar{r}$	37,71	28,99	**0,77**
Kosten			
C_d	0,0079	0,0112	1,42
C_o	0,0047	0,0096	2,05
C_u	0,0806	0,0662	0,82
C_u / C_o	17,28	6,93	-
C_{o+u}	0,0853	0,0757	**0,89**
C	0,0932	0,0869	**0,93**

Tabelle 4-4 Auswertung Performanz und Kosten: Sinus-Auslastung mit 150 virtuelle Benutzer und 15 Mandanten (Quelle: eigene Berechnungen)

4.2.4 Testlauf 4: Sinus - 250 Benutzer - 25 Mandanten - 5 VM statisch

	statisch	elastisch	Verhältnis elastisch/statisch
Performanz			
$\bar{r}$	45,87	39,39	**0,86**
Kosten			
C_d	0,0098	0,0122	1,24
C_o	0,0110	0,0133	1,21
C_u	0,1021	0,0923	0,90
C_u / C_o	9,26	6,92	-
C_{o+u}	0,1131	0,1056	**0,93**
C	0,1230	0,1178	**0,96**

Tabelle 4-5 Auswertung Performanz und Kosten: Sinus-Auslastung mit 250 virtuelle Benutzer und 25 Mandanten (Quelle: eigene Berechnungen)

4.2.5 Testlauf 5: Auslastungsspitze 360 Benutzer - 15 Mandanten - 3 VM statisch

	statisch	elastisch	Verhältnis elastisch/statisch
Performanz			
$\bar{r}$	93,52	51,32	**0,55**
Kosten			
C_d	0,0095	0,0180	1,90
C_o	0,0031	0,0101	3,32
C_u	0,1495	0,1299	0,87
C_u / C_o	48,95	12,82	-
C_{o+u}	0,1526	0,1400	**0,92**
C	0,1620	0,1580	**0,98**

Tabelle 4-6 Auswertung Performanz und Kosten: Auslastungsspitze mit 360 virtuelle Benutzer und 15 Mandanten (Quelle: eigene Berechnungen)

4.2.6 Testlauf 6: Auslastungsspitze 600 Benutzer - 25 Mandanten - 5 VM statisch

	statisch	elastisch	Verhältnis elastisch/statisch
Performanz			
$\bar{r}$	99,37	73,71	**0,74**
Kosten			
C_d	0,0139	0,0228	1,64
C_o	0,0070	0,0155	2,22
C_u	0,2074	0,2007	0,97
C_u / C_o	29,78	12,99	-
C_{o+u}	0,2144	0,2162	**1,01**
C	0,2283	0,2389	**1,05**

Tabelle 4-7 Auswertung Performanz und Kosten: Auslastungsspitze mit 600 virtuelle Benutzer und 25 Mandanten (Quelle: eigene Berechnungen)

4.3 Konsolidierung

Aufgrund der errechneten Verhältnisse und Vergleichswerte im Kapitel 4.2 werden nachfolgend die geometrischen Mittelwerte aus den einzelnen Vergleichswerten errechnet. Die Tabelle 4-8 beinhaltet in den Spalten für Performanz, Elastizität und Kosten die jeweiligen prozentualen Unterschiede $P_{S_1,\ w_x}$, $E_{S_1,\ w_x}$ und $T_{S_1,\ w_x}$ aus dem Kapitel 4.2. Mit der Berechnung des geometrischen Mittels über diese Werte ergeben sich die endgültigen Ergebnisse aus dem Vergleich des elastischen und statischen Testsystems in Bezug auf Performanz, Elastizität und Kosten. Auch hier bedeuten Werte unter 1 und eine grüne Einfärbung eine Verbesserung im elastischen System gegenüber dem statischen Testsystem. Werte über 1 und eine orange Einfärbung stehen für eine Verschlechterung im elastischen Testsystem gegenüber dem statischen Testsystem.

	Performanz $P_{S_1,\ w_x}$	Elastizität $E_{S_1,\ w_x}$	Kosten $T_{S_1,\ w_x}$
Testlauf 1	0,58	0,97	1,03
Testlauf 2	0,88	1,21	1,22
Testlauf 3	0,77	0,89	0,93
Testlauf 4	0,86	0,93	0,96
Testlauf 5	0,55	0,92	0,98
Testlauf 6	0,74	1,01	1,05
P_{S_1} | E_{S_1} | T_{S_1}	**0,72**	**0,98**	**1,02**

Tabelle 4-8 Konsolidierung der Ergebnisse (Quelle: eigene Berechnungen)

4.4 Analyse und Erkenntnisse

Im Zuge der Konsolidierung in Kapitel 4.3 werden die finalen Vergleichswerte der Performanz P_{S_1}, Elastizität E_{S_1} und Kosten T_{S_1} berechnet. Diese Werte fassen die Unterschiede der jeweiligen Aspekte zwischen dem statischen und elastischen System in drei komprimierten Kennzahlen zusammen. Auf Basis dieser und unter Berücksichtigung der Besonderheiten in den Vergleichswerten aus Kapitel 4.3 werden nachfolgend entsprechende Analysen zum Verhalten der beiden Systeme in den durchgeführten Tests durchgeführt.

4.4.1 Performanz

Entsprechend des finalen Vergleichswertes P_{S_1} für die Performanz ergibt sich im elastischen Testsystem eine deutliche Verbesserung von 28% gegenüber dem statischen System. Auch bei Betrachtung der durchschnittlichen Antwortzeiten in den Testläufen liegen diese im elastischen System klar unter denen des statischen. Passend zu diesen konsolidierten Werten verhält sich auch der Verlauf der durchschnittlichen Antwortzeit in den Liniendiagrammen der einzelnen Testläufe. In allen Testläufen liegt die Kurve der Antwortzeiten des elastischen Systems zumeist unter der des statischen Systems. Erwartungsgemäß steigt die durchschnittliche Antwortzeit des elastischen Systems nur über die des statischen, wenn die Workload steigt und das elastische System seine verfügbaren Ressourcen noch nicht an die höheren Anforderungen angepasst hat. Ebenso sinkt die durchschnittliche Antwortzeit im elastischen System mit der Bereitstellung zusätzlicher Ressourcen, wie in den Diagrammen von Testläufen 3 und 4 ersichtlich ist.

Während sich die durchschnittlichen Antwortzeiten des elastischen Systems wie erwartet verhalten, gibt es Auffälligkeiten bei der Betrachtung der Quantile. Obwohl die durchschnittliche Antwortzeit im elastischen System deutlich besser ist als im statischen, gilt selbiges nicht für die maximale Antwortzeit. Diese liegt in allen Testläufen deutlich über der des statischen Testsystems. Das 99%-Quantil liegt in den meisten Fällen bereits unterhalb, außer bei Testlauf 2, hier ist dies erst beim 95%-Quantil der Fall.

Aus diesem Verhalten der durchschnittlichen Antwortzeiten sowie der Quantile lässt sich ableiten, dass sich die Gesamt-Performanz des Systems aufgrund des elastischen Verhaltens zwar klar

verbessert, allerdings einige wenige Anfragen dafür eine deutlich höhere Antwortzeit haben. Dieses Verhalten hat in weiterer Folge ebenfalls Auswirkungen auf die Messung der Unterversorgung und somit auch auf die Elastizität und die Gesamtkosten des Systems.

4.4.2 Elastizität

In Bezug auf Elastizität muss in der Terminologie zwischen dem Begriff Elastizität im Sinne von elastischen Eigenschaften eines Systems, und der gemessenen Elastizität eines Systems unterschieden werden. Dies wird im Rahmen dieser Arbeit deutlich, da elastische Eigenschaften nur im elastischen Testsystem gegeben sind, dennoch lässt sich ein Messwert für Elastizität über die Ressourcenabweichungskosten sowohl im elastischen als auch im statischen System erheben. Das heißt, über die Ressourcenabweichungskosten kann ein Wert für die Elastizität in einem System gemessen werden, das eigentlich keine elastischen Eigenschaften besitzt, so wie es hier für das statische System der Fall ist.

Der zusammengefasste Wert für die Elastizität E_{S_1} weist eine leichte Verbesserung der gemessenen Elastizität im elastischen System gegenüber dem statischen auf. In den meisten Testläufen führt die in Kapitel 4.4.1 beschriebene bessere Performanz zu niedrigeren Unterversorgungskosten und dadurch in weiterer Folge zu einer besseren gemessenen Elastizität. Bei Betrachtung der Einzelwerte fällt auf, dass in zwei der sechs Testläufe ironischerweise im statischen System eine bessere Elastizität gemessen wurde als im elastischen System. Diese vermeintlich widersprüchlichen Ergebnisse ergeben sich aufgrund der Methode zur Berechnung dieser Metrik auf Basis der Ressourcenabweichungskosten, welche sich aus den Kosten für Über- und Unterversorgung zusammensetzen.

Generell fällt auf, dass die Ressourcenabweichungskosten stark von den Kosten für Unterversorgung dominiert sind. Das Verhältnis zwischen den Kosten für Unterversorgung und Überversorgung liegt bei allen Testläufen zwischen 6,92 und 48,95 und im Durschnitt bei 19,46. Das heißt, die Kosten für Unterversorgung sind im Schnitt knapp 20-mal höher als die der Überversorgung, weshalb eine prozentuale Veränderung der Unterversorgungskosten sich wesentlich stärker auf die gemessene Elastizität auswirkt als dieselbe Änderung der Überversorgungskosten. Dies liegt unter anderem daran, dass die Überversorgungskosten nach oben begrenzt sind, die Unterversorgungskosten jedoch nicht. Aufgrund dieser dominanten Rolle der Unterversorgungskosten ist die Elastizität weitgehend von der Unterversorgung geprägt.

Beim Verhältnis der Unterversorgungskosten zu den Überversorgungskosten ist weiters auffällig, dass dieses bei allen Testläufen im statischen System deutlich höher ist als im elastischen System. Der Durchschnitt liegt im statischen System bei 28,36 und im elastischen System bei 10,56. Das bedeutet, dass der Anteil der Überversorgungskosten an den Ressourcenabweichungskosten im elastischen System generell höher ist als im statischen System. Dies wird unter anderem dadurch bestätigt, dass die Kosten für Überversorgung in allen Testläufen im elastischen System höher sind als im statischen, und die Kosten für Unterversorgung in allen Testläufen niedriger. Die einzige Ausnahme davon ist Testlauf 2.

4.4.3 Zusammenhang Elastizität und Unterversorgung

Insofern ist Testlauf 2 ein Sonderfall, da er der einzige Testlauf ist, in dem die Unterversorgungskosten im elastischen System höher sind als im statischen. Des Weiteren ist dadurch Testlauf 2 auch der einzige Testlauf, in dem die Ressourcenabweichungskosten des elastischen Systems deutlich über denen des statischen Systems liegen. Dadurch wird die dominierende Rolle der Unterversorgungskosten bestätigt.

Diese Sonderstellung von Testlauf 2 deckt sich mit der bereits in Kapitel 4.4.1 beschriebenen Besonderheit von Testlauf 2, wonach nicht nur die maximale Antwortzeit, sondern auch das 99%-Quantil der Antwortzeit im elastischen System höher ist als im statischen System. Unter anderem ist dies darauf zurückzuführen, dass die Performanz des elastischen Systems im Testlauf 2 die kleinste Performanz-Verbesserung gegenüber dem statischen System aufweist, im Vergleich zu allen anderen Testläufen. Diese kleinere Performanz-Verbesserung ist wiederum durch den verwendeten Testplan begründet. Denn bei Testlauf 1 und Testlauf 2 wird derselbe Testplan mit derselben Auslastungskurve bis zu 240 gleichzeitigen virtuellen Benutzern verwendet, obwohl im Testlauf 1 nur drei virtuelle Maschinen im statischen System bereitgestellt werden und im Testlauf 2 fünf virtuelle Maschinen. Dadurch verbessert sich die Performanz des statischen Testsystems zwischen Testlauf 1 und Testlauf 2 deutlich, während die des elastischen Systems annähernd gleichbleibt. Dies ist deutlich im Vergleich der Diagramme 1, 3, 4 und 5 der Testläufe 1 und 2 zu erkennen.

Aufgrund der verbesserten Performanz des statischen Systems ist im Testlauf 2 gegenüber dem Testlauf 1 die QoS-Grenze niedriger. Die niedrigere QoS-Grenze führt zu höheren Unterversorgungskosten sowohl im statischen als auch im elastischen System, allerdings sind die Auswirkungen für das elastische System aufgrund des höheren 99%-Quantils der Antwortzeit deutlich größer. Diese Umstände führen zu 17% höheren Unterversorgungskosten im elastischen System gegenüber dem statischen System. Zusammen mit den höheren Überversorgungskosten sind die gesamten Ressourcenabweichungskosten um 21% höher und damit die gemessene Elastizität deutlich schlechter als im statischen System.

Auf diese Weise machen Testlauf 1 und 2 sichtbar, dass Elastizität im kurzfristigen Rahmen nur sinnvoll ist, wenn das statische Referenzsystem klare Zustände der Unterversorgung annimmt. Begründet wird dies einerseits mit den höheren maximalen Antwortzeiten im elastischen System, aufgrund derer von Grund auf höhere Unterversorgungskosten anfallen. Diese höheren Unterversorgungskosten müssen mit einer deutlich besseren Performanz ausgeglichen werden. Einerseits kann dies über steigende Workload erreicht werden, durch die das statische System an seine Grenzen stößt, damit seine Performanz sinkt und die Unterversorgungskosten steigen. Andererseits kann die QoS-Grenze durch die bessere Performanz im elastischen System soweit gesenkt werden, dass ein immer größerer Teil der Anfragen des statischen Systems über diese Grenze fallen und dadurch wiederum die Unterversorgungskosten steigen. Beide Fälle führen zu höheren Unterversorgungskosten im statischen System. Das heißt, ein besserer Messwert für Elastizität im elastischen System ergibt sich nicht aufgrund niedrigerer Kosten im elastischen System, sondern aufgrund höherer Kosten im statischen Referenzsystem.

4.4.4 Elastizität und Zeitrahmen

Des Weiteren wird die bedingte Sinnhaftigkeit von Elastizität im kurzfristigen Rahmen damit begründet, dass die Kosten für die Überversorgung in den vorliegenden Ergebnissen im elastischen Testsystem höher sind als im statischen. In den ausgeführten Testläufen sind diese höheren Überversorgungskosten dadurch begründet, dass im elastischen System eine höhere Anzahl an virtuellen Maschinen gleichzeitig ausgeführt wird. Da die CPU-Auslastung der virtuellen Maschinen allerdings in der Regel nicht dauerhaft 100% erreicht, erhöht jede zusätzliche virtuelle Maschine in Summe die ungenutzten Ressourcen und damit auch die Überversorgungskosten.

Im langfristigen Rahmen kann das elastische System diese höheren Überversorgungskosten in Perioden mit niedriger Auslastung ausgleichen, indem die verfügbaren Ressourcen und damit die Überversorgungskosten sowie die CPU Kosten entsprechend gesenkt werden. Im Vergleich dazu kann das statische System seine verfügbaren Ressourcen nicht anpassen, sodass die Überversorgungskosten bei niedriger Auslastung höher sind.

Im kurzfristigen Rahmen ist allerdings zu wenig Zeit für das elastische System, um die vorerst höheren Überversorgungskosten in Perioden von niedriger Auslastung wieder auszugleichen. Dadurch müssen im kurzfristigen Rahmen die höheren Überversorgungskosten des elastischen Systems mit niedrigeren Unterversorgungskosten gegenüber dem statischen System ausgeglichen werden. Wie oben beschrieben wird dies durch eine deutlich bessere Performanz des elastischen Systems im Vergleich zum statischen System erreicht. Stößt das statische System jedoch nicht an seine Kapazitätsgrenzen und gelingt es dem elastischen System nicht, eine deutlich bessere Performanz gegenüber dem statischen System zu erreichen, so können die höheren Überversorgungskosten nicht ausgeglichen werden. Dieses Szenario ist in Testlauf 6 zu sehen, in dem zwar die Unterversorgungkosten im elastischen System geringer sind als im statischen, jedoch nicht ausreichend gering, um die höheren Überversorgungskosten zu decken. Wird der Performanz-Vorteil des elastischen Systems gegenüber dem statischen System noch kleiner, so führt dies zu einem Szenario wie in Testlauf 2, in dem sowohl die Überversorgungs- als auch die Unterversorgungskosten im elastischen System über jene des statischen Systems steigen.

4.4.5 Kosten

Im gesammelten Vergleichswert für die Gesamtkosten T_{S_1} wird eine leichte Erhöhung der Gesamtkosten von 2% ausgewiesen. Bei Betrachtung der Einzelwerte fällt auf, dass, wie auch bei der Messung der Elastizität, der Testlauf 2 einen Ausreißer nach oben darstellt, durch den der Gesamtwert T_{S_1} über 1 gehoben wird.

Die Gesamtkosten werden aus der Summe der Ressourcenabweichungskosten und den Kosten der genutzten Ressourcen berechnet. Da die Ressourcenabweichungskosten im Vergleich zu den Kosten der genutzten Ressourcen aber in allen Testläufen sowohl im statischen als auch im elastischen System um ein Vielfaches höher sind, sind die Gesamtkosten weitgehend von den

Ressourcenabweichungskosten abhängig. Dies wird unter anderem dadurch bestätigt, dass in fünf der sechs Testläufe eine Erhöhung der Ressourcenabweichungskosten auch zu einer Erhöhung der Gesamtkosten sowie eine Senkung der Ressourcenabweichungskosten zu einer Senkung der Gesamtkosten führt.

In dieser Hinsicht stellt Testlauf 1 einen Sonderfall dar, da hier die Ressourcenabweichungskosten im elastischen System niedriger sind, die Gesamtkosten jedoch höher. Zurückzuführen ist dies auf die starke Erhöhung der Kosten für die genutzten Ressourcen, welche im elastischen System um 90% höher sind als im statischen. Ähnliche Messungen sind auch im Testlauf 5 gegeben, allerdings sind hier die Ressourcenabweichungskosten im elastischen System niedrig genug, um die höheren Kosten der genutzten Ressourcen auszugleichen.

Aufgrund der starken Abhängigkeit der Gesamtkosten von den Ressourcenabweichungskosten sind die Gesamtkosten auch indirekt von der Performanz des Systems abhängig. Deshalb beziehen sich die Erkenntnisse aus den Kapiteln 4.4.2, 4.4.3 und 4.4.4 auch weitgehend auf die Gesamtkosten. Aus den im Kapitel 4.4.4 beschriebenen Punkten folgt, dass es im kurzfristigen Rahmen zwei Fälle gibt, in denen das elastische System Kostenvorteile gegenüber dem statischen System besitzt. Im ersten Fall kann das elastische System bei hoher Workload eine deutlich bessere Performanz gegenüber dem statischen System erreichen, wodurch die Unterversorgungskosten des statischen Systems über die des elastischen Systems steigen. Im zweiten Fall ist die Workload nur sehr gering, sodass weder im statischen noch im elastischen System nennenswerte Unterversorgungskosten entstehen. In so einem Fall werden die Gesamtkosten mangels Unterversorgungskosten aufgrund der Überversorgungskosten sowie der Kosten für die genutzten Ressourcen definiert. Aufgrund der Fähigkeit zur Anpassung der verfügbaren Ressourcen entstehen dadurch geringere Überversorgungskosten und somit Kostenvorteile im elastischen System. Ist weder eine hohe noch eine niedrige, sondern eine durchschnittliche, einigermaßen konstante Workload gegeben, mit der das statische System seine Ressourcen gut nutzen kann, jedoch nicht überlastet ist, so sind die Kosten im elastischen System höher als im statischen. Dieser Fall ist im Testlauf 2 eingetreten.

Im mittel- und langfristigen Rahmen ist elastisches Verhalten in Bezug auf die Kosten dann sinnvoll, wenn sowohl hohe als auch niedrige Workloads eintreten. Demnach bringt elastisches Verhalten in einem System tendenziell dann Kostenvorteile, wenn das Auslastungsspektrum im System breit gefächert ist, das heißt, wenn die Workload des Systems über die Zeit um ein Vielfaches variiert.

5 CONCLUSIO

In der vorliegenden Arbeit werden Definition, Quantifizierung, Voraussetzungen und Auswirkungen von Elastizität behandelt. Des Weiteren werden die Themen Mandantenfähigkeit, Container-Technologien sowie Cluster-Management-Systeme und deren Zusammenhänge mit Elastizität aufgezeigt. Daraus werden Ansätze und Herausforderungen bei der Erstellung eines mandantenfähigen, containerbasierten Webshop-Systems mit Elastizität abgeleitet. Im empirischen Teil der Arbeit werden Vorschläge für die Architektur eines mandantenfähigen, containerbasierten Webshop-Systems mit und ohne Elastizität gebracht und in zwei unabhängigen Testsystemen implementiert. Im Zuge der durchgeführten Tests werden die Auswirkungen von Elastizität auf Performanz und Kosten erhoben.

5.1 Zusammenfassung

Elastizität ist als eine von Skalierbarkeit abgeleitete Systemeigenschaft gleichermaßen vielseitig. In Kombination mit Mandantenfähigkeit und Container-Technologien ergeben sich Potenziale und Synergien, aufgrund derer sich die Umsetzung von Elastizität in einem mandantenfähigen, containerbasierten Webshop-System generell anbietet. Für die konkrete Implementierung sollte ein passendes Cluster-Management-System verwendet werden, welches die Aufgaben zur Umsetzung des elastischen Verhaltens übernimmt.

Die im Rahmen der durchgeführten Tests erhobenen Metriken werden auf oberster Ebene zwischen Performanz-Metriken und Kosten-Metriken getrennt. Die maßgebliche Metrik für die Performanz der Systeme ist die durchschnittliche Antwortzeit. Um Elastizität zu messen gibt es verschiedene Ansätze in der Literatur. Einige davon basieren auf einer Berechnung der Kosten für Über- und Unterversorgung, so wie auch der in dieser Arbeit verwendete Ansatz. Dadurch ergibt sich die Metrik für Elastizität aufgrund der summierten, durchschnittlichen Ressourcenabweichungskosten. Damit bildet die Metrik der gemessenen Elastizität die erste maßgebliche Kosten-Metrik. Die zweite Kosten-Metrik wird über die durchschnittlichen Gesamtkosten des Systems abgebildet, welche sich aus den Ressourcenabweichungskosten und den CPU Kosten zusammensetzt.

Aus den Ergebnissen der durchgeführten Testläufe kann eine deutliche Verbesserung der Performanz des Testsystems durch Elastizität festgestellt werden. Durchschnittlich wird die Antwortzeit mit Elastizität um 28% verkürzt. Trotz der klaren Verbesserung der allgemeinen Performanz wird festgehalten, dass sich die maximale Antwortzeit im System durch Elastizität deutlich erhöht.

Der gesammelte Ergebniswert für Elastizität repräsentiert die Ressourcenabweichungskosten, welche hauptsächlich durch die Unterversorgungskosten beeinflusst werden. Die Unterversorgungskosten werden in fast allen Testläufen durch Elastizität verringert, sodass tendenziell auch die Ressourcenabweichungskosten und damit die gemessene Elastizität verbessert wird. Über alle durchgeführten Tests haben sich die gemessenen

Ressourcenabweichungskosten im elastischen System um 2% verringert. Aufgrund der höheren maximalen Antwortzeiten mit Elastizität steigen auch die grundlegenden Unterversorgungskosten. Diese müssen in einem elastischen System über eine deutlich bessere Performanz ausgeglichen werden, andernfalls ergeben sich Kostennachteile aufgrund der Elastizität.

Die Gesamtkosten des Systems erhöhen sich über alle Testläufe mit Elastizität durchschnittlich um 2%, wobei in drei der sechs Testläufe eine Verbesserung und in den anderen drei eine Verschlechterung festgestellt wird. Die Gesamtkosten werden am stärksten durch die Ressourcenabweichungskosten beeinflusst, wodurch eine Senkung der Ressourcenabweichungskosten meist auch eine Senkung der Gesamtkosten mit sich bringt. Aufgrund von elastischen Eigenschaften ergeben sich im kurzfristigen Rahmen Kostenvorteile, wenn die Auslastung entweder sehr gering oder sehr hoch ist. Im mittel- und langfristigen Rahmen ergeben sich Kostenvorteile bei einer stark variierenden Auslastung.

5.2 Bewertung

Die im Kapitel 3.8 aufgestellte Hypothese kann aufgrund der gewonnenen Erkenntnisse nur teilweise bestätigt werden. Der erste Teil der Hypothese wird bestätigt, nach der sich die Performanz im hier verwendeten mandantenfähigen, containerbasierten Webshop-Testsystem in den durchgeführten Testläufen verbessert hat. In Bezug auf die Kosten kann eine leichte Tendenz zu höheren Kosten durch Elastizität festgestellt werden, wodurch die Ergebnisse auf eine Widerlegung dieses Teils der Hypothese hindeuten. Jedoch lässt sich aufgrund des geringen Anstieges von 2% über sechs durchgeführte Testläufe keine eindeutige Bestätigung oder Widerlegung der zweiten Teilhypothese ableiten. Um diese Tendenz zu bestätigen oder zu widerlegen, müssen weitere Tests durchgeführt werden.

Die Verifizierung der Hypothese ist zugleich auch die Antwort auf die Forschungsfrage. Elastizität wirkt sich in einem mandantenfähigen, containerbasierten Webshop-System positiv auf die Performanz aus. Das konkrete Ausmaß der Verbesserung hängt von der Anzahl der virtuellen Maschinen im statischen Referenzsystem sowie vom Auslastungsmuster ab. Ob sich Elastizität positiv oder negativ auf die Gesamtkosten eines mandantenfähigen, containerisierten Webshop-Systems auswirkt, kann aufgrund der vorliegenden Ergebnisse nicht eindeutig beantwortet werden. Dafür sind weitere Testläufe notwendig, welche die gemessene leicht steigende Tendenz der Kosten bestätigen oder widerlegen. Generell lässt sich aufgrund der Ergebnisse jedoch feststellen, dass der positive Effekt von Elastizität auf die Performanz größer ist, als die Veränderung der Kosten.

5.3 Limitierungen

Im Hinblick auf die Ergebnisse und den daraus gezogenen Schlussfolgerungen müssen methodische und technische Limitierungen beachtet werden. Eine Limitierung in Bezug auf die Aussagekraft der Ergebnisse stellt die Anzahl der durchgeführten Testläufe dar. Wie im Kapitel 5.2 beschrieben, lässt sich mit den vorhandenen Ergebnissen zwar eine positive Auswirkung auf die Performanz feststellen, jedoch sind die Ergebnisse in Bezug auf die Kosten nicht eindeutig. Weitere Tests mit unterschiedlichen Auslastungsmustern und Skalierungsstufen sind notwendig, um präzisere Aussagen bezüglich der Auswirkung auf die Kosten treffen zu können.

Neben der Limitierung in der Anzahl der durchgeführten Testläufe muss ebenso die Laufzeit der Testläufe beachtet werden. Mit einer Laufzeit zwischen rund 100 und 180 Minuten lassen sich aus den Ergebnissen lediglich Aussagen über kurzfristige Zeitspannen treffen. Um auch Aussagen über die Langzeitauswirkungen von Elastizität auf Performanz und Kosten machen zu können, werden weitere Tests mit längeren Laufzeiten benötigt. Beispielsweise kann darin das durchschnittliche Auslastungsmuster eines Tages oder einer Woche abgebildet werden. Auf diese Weise ließen sich auch die in den Kapiteln 4.4.4 und 4.4.5 beschriebenen Ansätze zu mittel- und langfristigen Auswirkungen beurteilen.

Aus technischer Sicht gibt es eine Limitierung im Hinblick auf die Datenhaltung in den Testsystemen. Wie im Kapitel 3.6.2 beschrieben, wird pro Applikationsinstanz eine eigene MongoDB-Instanz innerhalb der gleichen Container-Instanz gestartet. Auf diese Weise verwenden die Applikationsinstanzen desselben Mandanten nicht dieselben Daten. In einer realen Anwendung hingegen muss eine gemeinsame Datenbasis pro Mandanten sichergestellt werden. Wie im Kapitel 3.6.2 beschrieben, wäre dies in den verwendeten Testsystemen über einen zentralen Fileserver oder MongoDB-Cluster pro Mandanten möglich. Ob und welche Auswirkungen so eine Zentralisierung der Daten pro Mandanten auf die Performanz und Kosten hat und wie es somit die Ergebnisse in der vorliegenden Arbeit beeinflusst, kann ohne weitere Untersuchungen in diesem Bereich nicht festgestellt werden.

Als weitere technische Limitierung wurden die vorgestellten Testläufe nur auf der Cloud-Plattform Microsoft Azure ausgeführt. Ob es messbare Unterschiede für dieselben Tests zwischen verschiedenen Cloud-Anbietern gibt, müsste im Rahmen weiterführender Tests erhoben werden. Als Plattformen für solche weiteren Tests bieten sich Amazon Web Services (AWS) sowie Google Cloud Platform (GCP) an.

5.4 Ausblick

Die vorliegende Arbeit beschreibt und zeigt die Potenziale von Elastizität in Bezug auf Performanz und Kosten. Vor allem bei stark variierenden Auslastungen können die Stärken dieser Systemeigenschaft vorteilhaft genutzt werden und je größer die Systeme sind und je mehr Ressourcen darin vorhanden sind, umso größer sind auch die Verbesserungspotenziale durch Elastizität.

Moderne Software-Systeme sind aufgrund der fortschreitenden Digitalisierung mit ständig neuen und wachsenden Anforderungen konfrontiert, wodurch diese immer größer und komplexer werden. Gleichzeitig werden durch die laufend steigende Popularität von Cloud-Plattformen mehr und mehr Systeme in der Cloud bereitgestellt und haben somit einfachen und bedarfsorientierten Zugriff auf zusätzliche Ressourcen, welche nur nach deren Verwendung Kosten verursachen.

Aufgrund dieser Entwicklungen ist davon auszugehen, dass die Optimierung von Ressourcennutzung auch zukünftig ein zentraler Bestandteil in den Anforderungen von Cloud-Systemen sein wird. Demzufolge ist zu erwarten, dass weitere Forschungen auf diesem Gebiet zusätzliche Optimierungspotenziale für Performanz und Kosten durch elastisches Verhalten offenbaren werden.

ABKÜRZUNGSVERZEICHNIS

AKS	-	Azure Kubernetes Service
AWS	-	Amazon Web Services
CPU	-	Central Processing Unit
GB	-	Gigabyte
GCP	-	Google Cloud Platform
HPA	-	Horizontal Pod Autoscaler
I/O	-	Input / Output
MB	-	Megabyte
QoS	-	Quality of Service
RAM	-	Random Access Memory
SaaS	-	Software-as-a-Service
SLA	-	Service Level Agreement
VM	-	virtuelle Maschine

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LITERATURVERZEICHNIS

Alfonso, C. de, Calatrava, A. & Moltó, G. (2017). Container-based virtual elastic clusters. *Journal of Systems and Software.* Vorab-Onlinepublikation. https://doi.org/10.1016/j.jss.2017.01.007

Almeida, R. F., Sousa, F. R. C., Lifschitz, S. & Machado, J. C. (2013). On defining metrics for elasticity of cloud databases. https://sbbd2013.cin.ufpe.br/Proceedings/artigos/pdfs/sbbd_shp_12.pdf

Azar, Y., Broder, A. Z., Karlin, A. R. & Upfal, E. (1994). Balanced Allocations. *Proc. 26th ACM Symp. on the Theory of Computing*, 593–602. http://users.eecs.northwestern.edu/~nickle/randAlg/AzarBKU99.pdf

Bass, L., Clements, P. & Kazman, R. (2013). *Software architecture in practice* (3. Aufl.). *Always learning*. Addison-Wesley/Pearson.

Bezemer, C.-P. & Zaidman, A. (2010). *Bezemer, Cor-Paul, and Andy Zaidman. "Multi-tenant SaaS applications: maintenance dream or nightmare? Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*. ACM. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.176.6776&rep=rep1&type=pdf

Bondi, A. B. (2000). Characteristics of Scalability and Their Impact on Performance.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J. & Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, *25*(6), 599–616. https://doi.org/10.1016/j.future.2008.12.001

Chen, T. & Bahsoon, R. (2018). Bridging Ecology and Cloud: Transposing Ecological Perspective to Enable Better Cloud Autoscaling.

Chetty, M. & Buyya, R. (2002). Weaving computational grids: how analogous are they with electrical grids? *Computing in Science & Engineering*, *4*(4), 61–71. https://doi.org/10.1109/MCISE.2002.1014981

Chiang, A. C. & Wainwright, K. (2005). *Fundamental methods of mathematical economics* (4. ed.).

Dragoni, N., Lanese, I., Larsen, S. T., Mazzara, M., Mustafin, R. & Safina, L. (23. Februar 2017). *Microservices: How To Make Your Application Scale*. http://arxiv.org/pdf/1702.07149v1

Fowler, M. (2015). *MicroservicePremium*. https://martinfowler.com/bliki/MicroservicePremium.html

Ganek, A. G. & Corbi, T. A. (2003). The dawning of the autonomic computing era. *IBM systems Journal*(42(1)), 5–18.

Gao, J., Bai, X., Tsai, W. T. & Uehara, T. (2013, März). SaaS Testing on Clouds - Issues, Challenges and Needs. In *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering* (S. 409–415). IEEE. https://doi.org/10.1109/SOSE.2013.98

Herbst, N. R., Kounev, S. & Reussner, R. (2013). Elasticity in cloud computing: What it is, and what it is not.

Islam, S., Lee, K., Fekete, A. & Liu, A. (2012). How a consumer can measure elasticity for cloud platforms. *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, 85–96. https://research.spec.org/icpe_proceedings/2012/p85.pdf

JavaScript - Random with gaussian distribution. (2020). https://riptutorial.com/javascript/example/8330/random--with-gaussian-distribution

Kubernetes. (2020a). *Assign CPU Resources to Containers and Pods*. https://kubernetes.io/docs/tasks/configure-pod-container/assign-cpu-resource/

Kubernetes. (2020b). *Horizontal Pod Autoscaler*. https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

Kubernetes. (2020c). *Taints and Tolerations*. https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/

Kuperberg, M., Herbst, N., von Kistowski, J. & Reussner, R. (2011). Defining and Quantifying Elasticity of Resources in Cloud Computing and Scalable Platforms.

Laux, F., Schmidt, A., Nitta, K. & Savnik, I. (Hg.). (2014). *DBKDA 2014: The Sixth International Conference on Advances in Databases, Knowledge, and Data Applications : April 20-24, 2014, Chamonix, France*. IARIA. https://d-nb.info/1129261743/34#page=134

Lehrig, S., Eikerling, H. & Becker, S. (2015). Scalability, Elasticity, and Efficiency in Cloud Computing: a Systematic Literature Review of Definitions and Metrics. In P. Kruchten, I. Ozkaya & H. Koziolek (Hg.), *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures - QoSA '15* (S. 83–92). ACM Press. https://doi.org/10.1145/2737182.2737185

Liu, H. H. (2009). Software Performance and Scalability.

Marinescu, D. C. (2018). *Cloud Computing: Theory and Practice* (2nd ed.). Morgan Kaufmann.

Mell, P. & Grance, T. (2011). The NIST Definition of Cloud Computing: Recommendations of the National Institute of Standards and Technology. http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf

Microsoft Azure Documentation. (2020). *Use the cluster autoscaler in Azure Kubernetes Service (AKS) - Azure Kubernetes Service*. https://docs.microsoft.com/en-us/azure/aks/cluster-autoscaler

Minhas, U. F., Liu, R., Aboulnaga, A., Salem, K., Ng, J. & Robertson, S. (2012). Elastic Scale-out for Partition-Based Database Systems. *2012 IEEE 28th International Conference on Data Engineering Workshops*, 281–288. https://cs.uwaterloo.ca/~kmsalem/pubs/ElasticVoltSMDB12.pdf

Nah, F. F.-H. (2004). A study on tolerable waiting time: how long are Web users willing to wait? *Behaviour & Information Technology*, *23*(3), 153–163. https://doi.org/10.1080/01449290410001669914

Namiot, D. & Sneps-Sneppe, M. (2014). On Micro-services Architecture.

Sadashiv, N. & Kumar, S.M. D. (2011). *Cluster, Grid and Cloud Computing: A Detailed Comparison*. IEEE. https://www.dcc.fc.up.pt/~ines/aulas/1314/CG/Presentations/Goncalo/papers/06028683.pdf

Taibi, D., Lenarduzzi, V. & Pahl, C. (2017). Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation. *IEEE Cloud Computing*, *4*(5), 22–32. https://doi.org/10.1109/MCC.2017.4250931

Weinman, J. (2011). Time is Money: The Value of “On-Demand”. http://www.joeweinman.com/resources/Joe_Weinman_Time_Is_Money.pdf

Winkler, F. (2019). VERHALTEN DER SKALIERBARKEIT EINES MANDANTENFÄHIGEN WEBSHOP-SYSTEMS UNTER STEIGENDER LAST: Ein Vergleich zwischen Monolithen und Microservices.

Wolff, E. (2017). *Microservices: Flexible software architecture*. Addison-Wesley.

Zeng, H., Wang, B., Deng, W. & Zhang, W. (2017). *Measurement and Evaluation for Docker Container Networking.* https://doi.org/10.1109/CYBERC.2017.78