Orts-, zeit- und kostenbasiertes Ressourcenmanagement im Cloud Computing

Anna Schwanengel

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

Orts-, zeit- und kostenbasiertes Ressourcenmanagement im Cloud Computing

Anna Schwanengel

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

vorgelegt von
Anna Schwanengel

1. Berichterstatter:	Prof. Dr. Claudia Linnhoff-Popien
2. Berichterstatter:	Prof. Dr. Stefan Fischer
Tag der Einreichung:	24. Januar 2014
Tag der Disputation:	21. März 2014

Eidesstattliche Versicherung

(siehe Promotionsordnung vom 12.07.11, §8, Abs. 2 Pkt. 5)

Hiermit erkläre ich an Eidesstatt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

Anna Schwanengel

Danksagung

Als Doktorandin der Ludwig-Maximilians-Universität München arbeitete ich von 2011 bis 2014 an dem vorliegenden Werk in einer Kooperation des Lehrstuhls „Mobile und Verteilte Systeme“ mit der Siemens AG.

Mein Dank richtet sich zuerst an Frau Prof. Dr. Claudia Linnhoff-Popien, die an ihrem Lehrstuhl stets ein motivierendes Umfeld mit einem ausgewogenen Verhältnis von wissenschaftlicher und industriebezogener Arbeit gestaltet. Mit persönlichem Interesse hat sie mich durch fachkundige Betreuung und konstruktive Hinweise zur Arbeit vorangebracht. Vielen Dank für die aufmerksame und kompetente Unterstützung. Weiterhin gilt mein Dank Prof. Dr. Stefan Fischer für die Erstellung des Zweitgutachtens dieser Dissertation.

Meinen Kollegen am Lehrstuhl Michael Beck, Florian Dorfmeister, Dr. Markus Duchon, Dr. Michael Dürr, Sebastian Feld, Florian Gschwandtner, Dr. Moritz Kessel, Dr. Robert Lasowski, Philipp Marcus, Chadly Marouane, Marco Maier, Valentin Protschky, Lorenz Schauer, Kim Schindhelm, Mirco Schönfeld, Kay Weckemann, Dr. Martin Werner und Kevin Wiesner möchte ich für das außerordentlich gute und freundschaftliche Arbeitsklima sowie die moralische Unterstützung danken.

Auch die Mitarbeiter der Siemens-Gruppe im Bereich des Cloud Computings begleiteten mich bei der Promotion. Die Fachdiskussionen mit Dr. Gerald Käfer unterstützten mich bei der Themenfindung und Durchführung der Promotion. Zudem gaben mir Sebastian Dippl, Dr. Uwe Hohenstein, Dr. Michael Jäger, Dr. Reto Krummenacher, Dr. Achim Luhn, Ludwig Mittermeier und Dr. Birgit Schiemann neue Anstöße und sorgten für die Fokussierung auf die praktische Anwendbarkeit der Konzepte. Für die angenehme und fachlich zielführende Zusammenarbeit bedanke ich mich herzlich.

Danken möchte ich außerdem vor allem meiner Familie und meinen Freunden, die jederzeit für mich da sind und diese Arbeit mit Interesse verfolgten.

Zusammenfassung

Cloud Computing bietet dem Nutzer die Möglichkeit, virtualisierte Applikationen, Plattformen und sogar Hardware wie Dienste zu nutzen. Der bereitstellende Provider kann die nötige Dienstkapazität in der Cloud elastisch an den aktuellen Bedarf anpassen. Durch die beliebige Ressourcenzuschaltung erwächst jedoch eine erhebliche ökologische und ökonomische Verantwortung. Zudem wird meist teuer überprovisioniert, um im Falle von Lastspitzen das Ablehnen von Anfragen zu vermeiden.

Bis heute bietet noch kein Anbieter eine voll automatisierte Lösung zum optimalen Ressourcenmanagement an. Zur Gewährleistung der Skalierbarkeit eines Cloud-basierten Systems bestehen lediglich regelbasierte Mechanismen. Der Nutzer muss die Logik manuell einstellen, um die benötigte Anzahl an Maschinen und ihre Lokalität festzulegen. So sind viele Fragen zu der Ressourcenverwaltung und den entstehenden Kosten für den Cloud-Anwender ungelöst.

In dieser Arbeit wird ein Protokoll entwickelt, das eine verbesserte Reihenfolge der Anfragenbearbeitung festlegt und die nötige Ressourcenmenge bestimmt. Die Ressourcenzuteilung basiert zum einen auf der Bedarfsreservierung durch die Ressourcen. Zum anderen kann das Nutzungsverhalten durch den Dienst beeinflusst werden. Die Simulationsergebnisse zeigen so stark reduzierte Antwortzeiten und Verwurfsraten.

Im Cloud-Umfeld ist die effiziente Bearbeitung der Anfragen allerdings häufig aufgrund von Abhängigkeiten zwischen den dienstbetreibenden Maschinen beeinträchtigt. Deshalb wird das Protokoll um einen Selbstkalibrierungsmechanismus und ein Ressourcenregelverfahren erweitert. Mit der Abbildung der Abhängigkeiten auf einen Graphen ist das Gesamtsystem skalierbar, ohne die Auslastung aller Ressourcen einzeln kontrollieren zu müssen.

Vom menschlichen Benutzer kann man jedoch keine Vorabreservierung bezüglich des Zeitpunkts seiner Dienstnutzung fordern – so wie das von Maschinen problemlos möglich ist. Für diesen Fall ermöglicht die vorliegende Arbeit deshalb die Extrapolation der Nutzerdaten aus Aufzeichnungen sozialer Netzwerke. Ohne Belastung der Anwender wird die Identifikation des Ressourcenbedarfs an einem bestimmten Ort realisiert.

Für eine solche Systemadaption führt der in dieser Arbeit entworfene Algorithmus eine ortsspezifische Vorhersage der nötigen Ressourcenanzahl durch. Diese Informationen dienen als Input für das entwickelte Protokoll und bieten so eine wohlproportionierte Provisionierung.

Die bei Skalierungen entstehenden Kosten sind meist schwer abzuschätzen. Aus diesem Grund werden im Verlauf dieser Arbeit Kostenfunktionen für den Nut-

zer und den Anbieter erstellt. Sie machen das optimale Mittel zwischen geringeren Kosten bei niedriger Ressourcenmenge und höherer Nutzerzufriedenheit bei großzügiger Kapazitätsabdeckung berechenbar.
Eine prototypische Umsetzung einschließlich verschiedener Simulationen zeigt, dass die entwickelten Ansätze ein deutlich verbessertes automatisiertes Ressourcenmanagement umsetzen.

Abstract

Cloud computing offers the possibility to use virtual applications, platforms and even hardware as a service. The cloud provider can elastically adapt resource capacity to the users' demand. However, from any desired resource provisioning also an enormous ecological and economical responsibility accrues. Besides, often costly overprovisioning is conducted in order to avoid request drops on load peaks.

Until today, instance allocation is not yet fully automated by vendors. Only rule-based mechanisms exist to react on load changes. The user needs to manually implement logics for defining the amount of resources, as well as the instance location. Hence, many questions about resource management and emerging costs remain.

In this work, a protocol is developed, which defines an optimal schedule for all clients and needed resources. The resource management is based on the demand reservation by the user. On the other hand the client usage can be delayed by the cloud service. The simulation shows good results regarding response times and drop rates.

Efficient scheduling in cloud systems, however, is often restricted by dependencies among the claimed resources. For that reason, the protocol was extended by a self-calibration- and a resource-regulation-method. With an appropriate mapping, it is possible to scale the whole system based on the dependency model without observing each instance utilization.

Human users can – contrary to machines – not be forced to determine their service usage in advance. By extrapolating records of the social Web, resource demands can be procured without burden the user. With this data, increasing load on popular services at a specific location can be deducted and resources can be scaled accordingly.

For system adaptation, the algorithm presented in this work enables location-based determination of required resource amounts. This serves as input for the developed protocol, while offering well-proportioned provisioning.

On scaling, generally emerging costs are difficult to estimate. For that reason cost functions for users and providers are developed. This enables the finding of an optimized trade-off between low emerging costs and a high user satisfaction.

Proven by prototypical implementation and simulations, all developed approaches enable an optimized and automated resource management.

Inhaltsverzeichnis

1 Einleitung **1**

2 Einführung in das Cloud Computing **5**
2.1 Cloud Computing – Definition und Begriffe 6
2.2 Cloud-Technologie im praktischen Einsatz 11
2.3 Angebote des Cloud-Umfelds 13
2.4 Zusammenfassung 22

3 Arbeitslastmanagement in Cloud-basierten Systemen **23**
3.1 Statisches IT-Ressourcenmanagement 24
3.2 Dynamisches IT-Ressourcenmanagement 25
3.3 Monitoring-Lösungen im Cloud Computing 28
3.4 Zusammenfassung 31

4 Konzeption eines Systems für effizientes Ressourcenmanagement **33**
4.1 Herausforderungen beim effizienten Ressourcenmanagement 34
4.2 Identifikation von Lastmustern 36
4.3 Ökonomische und ökologische Aspekte des Ressourcenmanagements 38
4.4 Lösungsansatz zum Ressourcenmanagement 41
4.5 Zusammenfassung 43

5 Ein Protokoll zur Ressourcenverwaltung **45**
5.1 Verwandte Arbeiten zur automatischen Ressourcenskalierung . . . 46
5.2 Ressourcenmanagement mittels Reservierung und Feedback 48
5.2.1 Informationsmodell 49
5.2.2 Kommunikationsmodell 51
5.2.3 Aktivitätsmodell 52
5.3 Entwurf der System-Architektur 55
5.4 Mathematische Analyse des Protokolls 57
5.5 Experimenteller Aufbau und Ergebnisse zum Protokoll 60
5.5.1 Szenario 1: Periodischer File-Download 61
5.5.2 Szenario 2: Permanenter Daten-Download im Hintergrund . . 64
5.5.3 Szenario 3: Ereignisgetriebene Nachrichtenverbreitung 67
5.6 Zusammenfassung 70

6 Proaktives automatisches Instanzenmanagement abhängiger Ressourcen 71
6.1 Verwandte Arbeiten zu Skalierung und Scheduling abhängiger Ressourcen . . . 72
6.2 Lastspitzen-Glättung per Ressourcenmanagement-Protokoll 74
6.3 Selbstkalibrierungsmechanismus durch den Service-Load-Manager . 77
6.4 Ressourcenregelverfahren des Managers 78
6.5 Abstrakte Analyse der Protokollerweiterung beim Skalierungsprozess 79
6.6 Ergebnisse der Protokollerweiterung 82
6.6.1 Evaluation des protokollspezifischen Scheduling-Verfahrens . . 83
6.6.2 Ergebnisse der Anwendungssimulationen 86
6.7 Zusammenfassung 100

7 Ressourcenfestlegung mit Nutzerdaten sozialer Netze 101
7.1 Verwandte Arbeiten zur vorausschauenden Ressourcenbereitstellung 102
7.2 Ressourcenmanagement mit sozialen Netzwerk-Daten 105
7.2.1 Identifikation von steigender Arbeitslast 107
7.2.2 Ortsspezifische Bedarfsbestimmung 112
7.3 Evaluation des Algorithmus zur ortsbasierten Ressourcenallokation 116
7.3.1 Fehler-Quantifizierung der Chi-Quadrat-Methode 117
7.3.2 Simulation zur Bestimmung der Antwortzeiten 120
7.3.3 Ergebnisse und Bewertung der Simulationen 122
7.4 Mathematische Analyse und Entwicklung der Kostenfunktionen . . 125
7.5 Zusammenfassung 135

8 Zusammenfassung und Ausblick 137

Literaturverzeichnis 144

1 Einleitung

„Cloud Computing – Revolution oder Evolution?“ [151] – diese Frage trifft exakt den Kern des konträr diskutierten Themas Cloud Computing. Dabei stehen sich zwei Lager gegenüber. Eine Seite ist der Meinung, dass Cloud Computing eine neue Denkweise und Revolution der IT-Struktur bedeutet. Die andere Seite sieht Cloud-Systeme als bereits bekannte, lediglich minimal veränderte Computing-Strukturen an.

Nichtsdestotrotz lässt sich das Interesse rund um den IT-Sektor deutlich an aktuellen Statistiken ablesen. So erklärte der Branchenverband BITKOM für Deutschland im Jahr 2012 einen Umsatzanstieg der Informationstechnologien bezüglich Hardware, Software und Diensten von 3,1 % auf 72,4 Milliarden Euro. Dabei stiegen die Marktumsätze im Bereich der Cloud-basierten Dienste auf 5,3 Milliarden Euro an [26]. Bis Ende 2012 war laut Informationen des BITCOM „Cloud Monitors 2013“ und dem Netzwerk des Wirtschaftsprüfungs- und Beratungsunternehmen KPMG der Anteil deutscher Unternehmen, die Cloud Computing im Einsatz hatten, bereits von 28 % auf 37 % gestiegen. Für weitere 29 % stand die Cloud-Technologie zur Debatte. Diese Daten ergaben sich aus einer repräsentativen Befragung von 436 Unternehmen [27].

Für das Jahr 2013 prognostizierte BITKOM einen weiteren 47-prozentigen Marktanstieg des Cloud Computings auf schon 7,8 Milliarden Euro. Mit 4,6 Milliarden Euro sollen 53 % des Wachstums unternehmerische Dienste umfassen und die verbleibenden 3,2 Milliarden Euro Umsatz auf Privatkunden entfallen. Laut weiterer Hochrechnungen wird bis 2016 eine Umsatzsteigerung der Cloud-Dienste auf 20,1 Milliarden Euro erwartet [27]. Dies zeigt die jetzige und zukünftige Bedeutung der Cloud-Technologie für die Computerbranche.

Cloud Computing wird heutzutage als eine Lösung verstanden, mit der IT, sei es in Form von Hardware-Infrastruktur, von Plattformen oder von Software-Anwendungen, als Dienst aus einer entfernten virtuellen Umgebung genutzt werden kann [110]. In den letzten Jahrzehnten haben Überprüfbarkeit von Quality-of-Service-Anforderungen, Standardisierungen und Technologien wie Virtualisierung und Selbstmanagement genau diesen Prozess begünstigt [151]. Folglich lässt sich Cloud Computing als eine Kombination mehrerer Aspekte verstehen: Die Evolutionsgeschichte der IT-Welt führt zu einer revolutionären, neuen Art des Nutzungsmodells der zugehörigen IT-Komponenten.

Dabei werden Ressourcen in isolierten Rechenzentren angemietet und über Internet-Verbindungen genutzt. So muss der ermittelte Bedarf nicht mehr im eigenen Unternehmen zur Verfügung stehen. Vielmehr kann der Nutzer die Instanzen gegen eine Gebühr direkt beim Dienstanbieter in Anspruch nehmen.

Bei abfallender Last können nicht mehr benötigte Maschinen wieder freigegeben und von anderen Nutzern weiterverwendet werden.
Elastizität ist in diesem Kontext ein signifikantes Cloud-Merkmal. Das bedeutet, zusätzlich erforderliche Ressourcen können nahezu jederzeit virtualisiert und in beliebigem Umfang im Rechenzentrum hinzugemietet werden. Diese Bereitstellung in Form von weiterer CPU-, Speicher- oder Software-Kapazität wird auch Provisionierung genannt. Ebenso kann der Nutzer Instanzen dort sofort wieder abziehen, wenn er sie nicht mehr benötigt. Dabei müssen keine physikalischen Rechner betriebsintern an- und abgeschafft werden.
Da nach BITKOM-Prognosen die Arbeitslast auf den bereitgestellten Servern weiterhin zunimmt, wird von heutigen IT-Systemen eine Bearbeitung von Millionen von Anfragen erwartet. Dies ist grundsätzlich durch die Verteilung der Aufgaben auf mehrere Rechner über Lastbalancierer realisierbar.
Mithilfe von Mechanismen zur Virtualisierung und Lastverteilung sowie durch die Bereitstellung großer Datenbanken erscheinen die Cloud-Kapazitäten nahezu unbegrenzt. Allerdings führt die beliebig große Instanzenmenge nicht automatisch zum besten Preis-Leistungsverhältnis. Heute wird meist noch überprovisioniert, um in Zeiten großer Arbeitslastanstiege keine Anfragen verwerfen zu müssen. Beispielsweise führen Beobachtungen des „The Economist“ und des „McKinsey Reports“ zu der Annahme, dass sich in den letzten Jahren die Serverauslastung der Rechenzentren in einem Bereich von nur 5 % bis 10 % bewegt [108], [141], [162].
Die Ermittlung, mit welcher minimalen Anzahl an Maschinen der Dienst maximal effizient erbracht werden kann, ist deshalb entscheidend. Eine Herausforderung stellt dabei die Kompromissfindung dar zwischen kostengünstigem, geringem Volumen physikalischer IT-Ressourcen und gleichzeitiger hochverfügbarer Diensterbringung. Dabei müssen zuvor vereinbarte Nutzerverträge – sogenannte Service-Level-Agreements, kurz SLAs – eingehalten werden.
Obwohl immer neue Ansätze für die Bereitstellung virtueller Maschinen und ihre Integration auf physikalischen Betriebsmitteln entstehen, gibt es heutzutage auch von großen Cloud-Anbietern keine voll automatisierte Lösung. Das Ressourcenmanagement erfordert den manuellen Eingriff durch den Nutzer, um die Anzahl zu startender und stoppender Komponenten und ihre jeweilige Lokalität festzulegen [4]. Um eine Skalierbarkeit des Cloud-Systems zu gewährleisten, bestehen lediglich regelbasierte Mechanismen, die die Implementierung der Logik durch einen Nutzer bedingen [104].
Somit ergeben sich für den Cloud-Nutzer bei der Provisionierung viele Fragen bezüglich der entstehenden Gebühren und der Ressourcenverwaltung, da das nötige manuelle Nachjustieren aufwändig und teuer ist. Wo neue Ressourcen am besten hinzuzuschalten sind, ist ebenfalls nicht ohne weitere Analysen zu beantworten. Die Erstellung umfassender Lastprognosen ist häufig zeitintensiv und kostenbehaftet. Insofern sind die essentiellen Herausforderungen der Ressourcenbereitstellung trotz jahrelanger Erforschung noch nicht befriedigend bewältigt. Genau dafür bietet diese Arbeit eine angemessene Lösung.

Sowohl die Provisionierung neuer Instanzen als auch ihre Vergabe und Verteilung auf derselben physikalischen Hardware verursachen nicht vernachlässigbare Verzögerungen. Bei Lastspitzen führt ein nicht ausreichender Ressourcenbestand zu einem rasanten Anstieg der Bearbeitungszeit aller Clients. Es müssen Anfragen verworfen werden, um nicht die Funktionalität des Gesamtsystems zu gefährden. Vor allem gilt das, wenn die Zeit zum Starten neuer Maschinen zu lang ist und nicht schnell genug reagiert wird. Dieser Fakt unterstreicht die Bedeutung eines effizienten Ressourcenmanagements.

Der Arbeitslastverteilung und dem zugehörigen Ressourcenmanagement kann man sich von zwei Seiten nähern: durch Reservierung oder mittels Vorhersage [176]. Im industriellen Umfeld folgt der Dienstaufruf durch einen Client, also durch eine Maschine, oft wiederkehrenden Mustern. Falls die Clients bereits vorab wissen, wann sie einen Service anfordern, können sie ihn im Voraus reservieren. Generell führen zu große Schwankungen der gesamten Ressourcenanzahl aufgrund des Verwaltungsaufwandes zu unnötigen Kosten. Deshalb ist ein möglichst konstantes Kapazitätenvolumen bereitzustellen.

Zu diesem Zweck stellt diese Arbeit ein Protokoll vor, das eine optimale Abarbeitungsreihenfolge – synonym als Schedule bezeichnet – für viele anfragende Clients bestimmt. Gleichzeitig wird die benötigte Ressourcenmenge festgelegt. Der Service-Load-Manager kann dabei die Zuteilung der Ressourcen beeinflussen und auf die Bedarfsreservierung durch den Dienstnutzer eingehen. Außerdem verschafft das Protokoll dem Cloud-Dienst die Möglichkeit, im Rahmen zuvor vereinbarter Service-Level-Agreements das Nutzerverhalten zu beeinflussen. Die Ergebnisse der Simulationen zeigen starke Verbesserungen der gemessenen Antwortzeiten und Anfragenverwürfe.

Häufig sind Abhängigkeiten zwischen den angefragten Ressourcen, also den dienstbetreibenden Maschinen, zu erkennen. Um unnötigen Überwachungsaufwand zu vermeiden, ist es folglich sinnvoll, diese Relationen abzubilden. Anhand der jeweiligen Verbindungsnachweise kann das Gesamtsystem skaliert werden, ohne die Auslastung jeder einzelnen Instanz kontrollieren zu müssen.

Das Protokoll wird dafür um einen Selbstkalibrierungsmechanismus erweitert. Dieser bestimmt die Abhängigkeiten vor dem Regelbetrieb und baut einen Graphen auf. Während des Betriebs erfolgt dann anhand des Abhängigkeitsgraphen das Ressourcenregelverfahren durch den Service-Load-Manager. Die Evaluation zeigt äußerst vielversprechende Ergebnisse bezüglich der Abarbeitungszeit im Vergleich zu standardmäßigen Scheduling-Verfahren.

Die andere Möglichkeit, effizientes Ressourcenmanagement zu gewährleisten besteht darin, Vorhersagen bezüglich der künftigen Lastentwicklung durch Einbindung von Nutzerinformationen zu treffen. Bei Bedarf können dann mittels dieser Prognose Ressourcen vorzeitig gestartet werden. Allerdings kann von menschlichen Nutzern – im Gegensatz zu Maschinen – nicht erwartet werden, dass sie sich im Voraus festlegen, wann sie einen bestimmten Dienst nutzen und diesen aktiv reservieren. Deshalb müssen entsprechende Daten aus Nutzerangaben extrahiert werden, ohne den Benutzer zusätzlich zu belasten.

Dafür eignen sich soziale Netzwerke, da ihre Nutzer darüber wissentlich oder unbewusst Informationen über ihre Interessen und Aufenthaltsorte mitteilen. Mit diesen Daten kann auf ein steigendes Lastvolumen an bestimmten Orten geschlossen werden. Die Bestimmung der genauen Anzahl benötigter Ressourcen in Abhängigkeit des Hauptbedarfsortes ist mit dem im Rahmen dieser Arbeit entworfenen Algorithmus möglich. Werden wenige Ressourcen bereitgestellt, sind die Kosten zwar geringer, doch muss mit Einbußen bei der Nutzerzufriedenheit gerechnet werden. Diese ist bei großzügiger teurer Kapazitätsabdeckung eher zu garantieren. In Ergänzung zu dem vorgestellten Algorithmus werden Funktionen entwickelt, anhand derer ein Cloud-Anbieter einen optimalen Kompromiss zwischen Kosten und Nutzerzufriedenheit findet.

Diese in der Einleitung genannten Aspekte werden in der vorliegenden Arbeit ausführlich erörtert. Sie strukturiert sich wie folgt: Kapitel 2 dient der Einführung in die Cloud-Technologie. Es werden bedeutende Begrifflichkeiten der Cloud-Technologie und bestehende Cloud-Angebote erläutert. In Kapitel 3 wird das historisch gewachsene Ressourcenmanagement in Cloud-basierten Systemen mit verwandten Arbeiten dargelegt.

Nachdem die Grundlagendefinitionen herausgearbeitet und der Stand der Technik erfasst sind, befasst sich Kapitel 4 mit der Konzeption der in dieser Arbeit neu entwickelten Lösung zum effizienten Ressourcenmanagement. Dieses Kapitel skizziert für die identifizierten Herausforderungen den erarbeiteten Lösungsansatz in den Grundzügen.

Der erste Teil des entworfenen Systems zum effizienten Ressourcenmanagement wird in Kapitel 5 beschrieben. Es stellt das entwickelte Protokoll zur Bestimmung des Schedules und der benötigten Ressourcenmenge vor. Die Protokollbestandteile werden mathematisch und experimentell evaluiert.

Die Erweiterung dieses Ansatzes zum proaktiven automatischen Instanzenmanagement von abhängigen Ressourcen als zweiten Lösungsschritt erläutert Kapitel 6. Hier wird erklärt, wie die protokollspezifische Selbstkalibrierung und das Ressourcenregelverfahren eine Lastspitzen-Glättung begünstigen.

Der dritte Lösungsbeitrag in Kapitel 7 befasst sich mit dem Fall, dass keine Reservierungen vorgenommen werden. Infolgedessen ist die zukünftige Lastentwicklung nicht bekannt und zeitnahe Bedarfsdeckung erschwert. Zur Bewältigung dieser Situation schätzt der ausgearbeitete Algorithmus den Ressourcenrahmen mithilfe sozialer Netzwerke. Mittels Extrapolation von Nutzerdaten werden Arbeitslastprognosen identifiziert und der Ort des hauptsächlichen Ressourcenbedarfs abgeleitet. Schließlich wird dort zusätzliche Kapazität bereitgestellt. Dieses Kapitel erklärt die entwickelten Funktionen zur Berechnung der genauen Anzahl benötigter Ressourcen und der entstehenden Kosten. Die daraus abgeleitete Formel bestimmt ein Optimum der Ressourcenmenge.

Eine abschließende Zusammenfassung und Darstellung der gewonnenen Erkenntnisse bietet Kapitel 8. Dieser Abschnitt gibt einen Ausblick einschließlich weiterführender Leitgedanken und Möglichkeiten im Forschungsgebiet der Cloud-basierten Systeme.

2 Einführung in das Cloud Computing

Der Begriff Cloud Computing bezeichnet die Nutzung von angemieteter Rechenkapazität oder auch externem Speicherplatz. Danach werden einzelne Nutzer-Anwendungen nicht mehr ausschließlich in eigenen Rechenzentren bereitgestellt, sondern die Anbieterunternehmen beziehen Rechenleistung von Dritten über ein Netzwerk. So dient Cloud Computing zum einen der Entwicklung von Anwendungen auf entfernten Servern. Sie werden dort den Nutzern zur Verfügung gestellt. Zum anderen ist dadurch Datenspeicherung in verteilten Umgebungen realisierbar [15].

Der IT-Trend Cloud Computing hat sich in den letzten Jahren etabliert und ist aus der heutigen Unternehmenswelt nicht mehr wegzudenken. Im Jahr 2010 führte der dauernd steigende Umfang und das weltweite Marktpotenzial des Cloud Computings bis 2014 zu einem von Gartner geschätzten Volumen von 148,8 Milliarden US-Dollar [140]. Auch in Deutschland ist die Technologie des Cloud Computings fortschreitend auf dem Vormarsch und erwirtschaftete 2013 fast acht Milliarden Euro [27]. Das unterstreicht seine Bedeutsamkeit für den heutigen informationstechnologischen Markt deutlich.

Im Cloud Computing gibt es verschiedene Ebenen der Nutzung, von der Anmietung der reinen Infrastruktur, über die Instrumentalisierung einer in der Cloud laufenden Entwickler-Plattform, um eigene Anwendungen zu entwerfen, bis zur Verwendung einer lauffähigen Cloud-Applikation. Gewählt werden kann zusätzlich zwischen einem öffentlichen, einem ausschließlich privat genutzten System und einer hybriden Lösung.

Alle transparent vom Anwender genutzten Ressourcen liegen in der Cloud. Durch die externe Speicherung und Bearbeitung größerer Datenmengen sollen allerdings angemessene Antwortzeiten auf Anfragen nicht beeinflusst werden. Einen hohen Stellenwert nehmen dabei die zugrunde liegende Service-Architektur, das Utility Computing sowie Autonomic Computing und natürlich die Technologie der Virtualisierung ein [180].

Utility Computing beschreibt ein Nutzungskonzept. In einer entsprechenden Service-Struktur werden Ressourcen gegen Bezahlung in Anspruch genommen, um die gewünschten Dienste auszuführen [127].

Angelehnt an den menschlichen Körper werden beim Autonomic Computing Regeln definiert, nach denen das System selbstständig agiert. So muss die verteilte Umgebung nicht fortlaufend durch Administratoren direkt überwacht und reguliert werden. Darin werden die vier Aspekte der Selbstkonfiguration

der Komponenten, der Selbstheilung im Fehlerfall, der Selbstoptimierung zur bestmöglichen Ausnutzung der Ressourcen und des Selbstschutzes zur Erkennung und Verhinderung von Angriffen vereint [82].
Der folgende Abschnitt bietet einen einleitenden Überblick über den aktuellen Stand der Dinge im Bereich des Cloud Computings und analysiert verschiedene Definitionen des Cloud Computings. Außerdem wird ausgeführt, welche Anbieter im Cloud-Markt dominieren und welche Produkte und Dienste durch sie angeboten werden.

2.1 Cloud Computing – Definition und Begriffe

Cloud Computing ist noch weit von einheitlichen Standards entfernt. Aus vielen Bereichen sind unterschiedliche Aspekte für die Technologie charakteristisch. Im Folgenden werden die wichtigsten und bekanntesten Interpretationsansätze aufgezeigt. Das schafft eine einheitliche Basisdefinition.
Nach Vaquero et al. [174] beschreibt Cloud Computing einen großen Pool an einfach nutzbaren und verfügbaren virtuellen Ressourcen, wie Hardware, Plattformen und Diensten, die dynamisch rekonfigurierbar sind, um sich an Veränderungen im Anfrageverhalten anzupassen. Der Ressourcen-Pool wird typischerweise als Pay-per-Use-Modell angeboten mit Absicherungen über kundenspezifische Service-Level-Agreements. Im Artikel „What Cloud Computing really means“ wird Cloud Computing außerdem sehr umfangreich beschrieben und umfasst viele, auch altbekannte Aspekte [75]:

- **Software-as-a-Service**: SaaS bietet Browser-Applikationen für Tausende Kunden gleichzeitig über dieselbe Architektur an. Dadurch entstehen weniger Kosten für den Kunden bezüglich eines Vorab-Invests und für den Anbieter bezüglich der Wartung. Es handelt sich um Anwendungen für Human Resources, Customer Relationship Management (CRM) und Enterprise Resource Planning (ERP) – z.B. von *Salesforce* Enterprise-Applikationen und von *Google Apps* Desktop-Applikationen.

- **Utility Computing**: Anbieter wie *Amazon*, *Oracle* oder *IBM* stellen Speicher und virtuelle Server bereit, die von der IT-Welt on-demand angefordert werden können.

- **Web Services**: in der Cloud fokussieren sie sich auf den Software-Aspekt und stellen APIs bereit, die es Entwicklern ermöglichen, über das Internet genutzte Anwendungen mit zusätzlichen Funktionalitäten auszustatten. Beispielsweise bieten *Strike Iron* und *Xignite* Business Services oder *Google Maps* seine API für Erweiterungen bereit.

- **Platform-as-a-Service**: ermöglicht die Erstellung eigener Applikationen auf externer Infrastruktur. Dies reicht von leichtgewichtiger Entwicklung auf *Yahoo Pipes* über Varianten wie *Legos*, die Design und Funktionalitäten vom Anbieter übernehmen, bis zu hoher Programmier-

freiheit bei *Force.com* von *Salesforce*, der *Google App Engine* oder der *Microsoft Azure Plattform*.

- **Managed Services**: Die MS-Provider stellen mit der ältesten Form des Cloud Computings kontrollierte Dienste als Anwendungen für die IT-Welt bereit. Beispiele sind Virus Scanner für Emails oder Application Monitoring Services z.B. von *Mercury*, oder Sicherheitsdienste von *SecureWorks* oder Googles *Postini*. Produkte wie *CenterBeam* oder *Everdream* stellen Desktop Management Services.
- **Service Commerce Platforms**: sind hybride Lösungen aus SaaS und MS, die sich mit der Handelsumgebung befassen. So ermöglichen es *Rearden Commerce* oder *Ariba*, dass Nutzer Reisen oder Bürodienste über diese Plattform abwickeln können. Sie übernimmt dann die eigentliche Dienstausführung und Bezahlung.
- **Internet Integration**: Für Cloud-Integrationen entwickelt beispielsweise *OpSource* einen Service Bus. Auch der SaaS-Anbieter *Workday* stellt eine Business-to-Business Integration inklusive Enterprise Service Bus mit *CapeClear*.

Sriram et al. [164] führen in einer umfassenden Forschungsagenda 51 verschiedene akademische Arbeiten auf, die sich mit der Thematik des Cloud Computings befassen. Die wissenschaftliche Literatur wird von den Autoren in die folgenden sechs Kategorien unterteilt: Generelle Einführung, Definitionen, Protokolle, Interfaces und Standards, Erkenntnisse aus verwandten Technologien, Aufbau einer Cloud sowie realistische Anwendungsfälle.
Eine weitere Agenda, welche Forschungsfelder in diesem Bereich noch zu bearbeiten sind, stellen Birman et al. [20] auf. Dies umfasst:

- das Management existierender Leistung und anstehender Arbeitslast innerhalb der Rechenzentren,
- die Entwicklung von stabilen „Event Notification“-Plattformen und entsprechenden Technologien zum Management,
- die Verbesserung der Virtualisierungstechnologie und
- das Schaffen des Verständnisses, um effizient mit einer großen Anzahl an low-end und fehleranfälligen Komponenten arbeiten zu können.

Eine weitere Definition zum Cloud Computing gibt der Wegbereiter des Grid Computings, Ian Foster [59]. Er stellt fest, dass Cloud und Grid Computing sehr große Ähnlichkeiten aufweisen. Die Vision, durch Ressourcennutzung von Dritten Kosten zu reduzieren und dabei gleichzeitig Ausfallsicherheit sowie Flexibilität zu erhöhen, ist bei beiden dieselbe. Es ergeben sich auch dieselben Probleme. Außerdem ist der Bedarf nach enormen Kapazitäten, um die immer größer werdenden Datenmengen der heutigen Zeit zu analysieren und zu speichern, deutlich angestiegen. Foster et al. [60] arbeiten neue Anforderun-

gen heraus und zeigen große Veränderungen in Business Modellen, Security Bereichen, Ressourcenmanagement und in der Abstraktion auf.
Nachfolgend wird auf die weitgehend akzeptierte Beschreibung des amerikanischen National Institute of Standards and Technology (NIST) aufgesetzt. Sie beinhaltet die bedeutendsten Aspekte von Cloud Computing:

> „Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction“ [110].

Danach gilt als **Definition**, dass Cloud Computing ein Modell für den bedarfsgerichteten Zugang zu einem geteilten Pool von konfigurierbaren Computing-Ressourcen ist, die sehr schnell und mit minimalem Managementaufwand bereitgestellt und auch wieder freigegeben werden.
Die entscheidenden **Charakteristika der Cloud** sind nach NIST die Realisierungen von *On-demand Self-Services*, *Broad Network Access*, *Resource Pooling*, *Rapid Elasticity* und *Measured Services*.
Der Cloud-Begriff umfasst dabei **vier Entwicklungsmodelle**: Die *Private Cloud* bedient allein eine Organisation. In *Community Clouds* teilen sich verschiedene Organisationen die Infrastruktur zur Unterstützung gemeinsamer Interessen. Daneben gibt es die *Public Cloud*, die Ressourcen für die breite Masse stellt und die *Hybrid Cloud*, die eine Kombination der Cloud-Arten ist.
Weiter existieren **drei Service-Modelle**: *Software-as-a-Service* (SaaS), *Platform-as-a-Service* (PaaS) und *Infrastructure-as-a-Service* (IaaS). Mittels SaaS können beim Anbieter laufende Applikationen über Schnittstellen von verschiedenen Geräten genutzt werden. PaaS bietet dem Kunden eine Plattform zur Entwicklung von Anwendungen für dessen Nutzer. Bei IaaS stellt der Anbieter die Infrastruktur, auf die der Nutzer aufbaut [110].
Abbildung 2.1 veranschaulicht einen verallgemeinerten Cloud-Stack. Aufbauend auf den Definitionsgrundlagen werden nun die wichtigsten Begriffe des Cloud Computings im Detail vorgestellt.

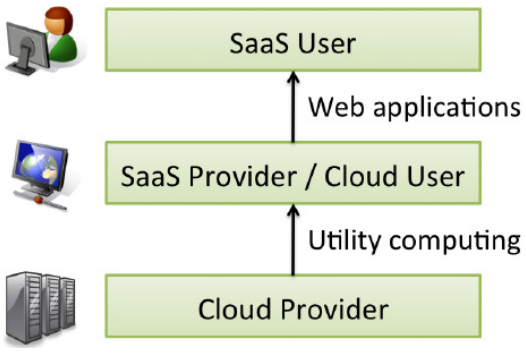


Abbildung 2.1: Cloud-Stack nach [16]

Dienstmodelle des Cloud Computings

Cloud Computing lässt sich grundsätzlich in drei verschiedene Service-Kategorien unterteilen. Sie alle greifen das Prinzip auf, das jeweilige Angebot wie einen voll funktionsfähigen Dienst „out of the box“ zu verstehen:

- **Software-as-a-Service:** Im SaaS-Modell stellt der Anbieter über alle Schichten lauffähige Applikationen für verschiedenste Kunden bereit. Die gebrauchsfertige Software liegt in der Cloud und ist über ein Netzinterface nutzbar. Der Anbieter kann die Wünsche einzelner Kunden per individueller Konfiguration realisieren. Da die Anwendung nicht lokal auf dem jeweiligen Kundenrechner installiert wird, entfällt für die Endkunden der Einrichtungs- und Instandhaltungsaufwand [1].

  Beispielsweise bietet *Microsoft* mit den *Office Web Apps* einen Service zum online-basierten Erstellen, Bearbeiten und Speichern von PowerPoint-, Word-, Excel- und OneNote-Dokumenten und erweitert dies mit *Office 365* auf Email- und Webkonferenz-Zugriffsmöglichkeiten. Vergleichbares wird über die *Google Apps* gestellt. Ebenso werden Kundenverwaltung und Planungsmodelle über CRM- beziehungsweise ERP-Tools zum Beispiel von *Salesforce* oder *Comarch* online angeboten.

- **Platform-as-a-Service:** PaaS bietet seinem Nutzer eine funktionsfähige Entwicklungsumgebung mit eingebundener Datenbank, Middleware und Anwendungssoftware. Der Cloud-Nutzer kann für seine jeweiligen Kunden Anwendungen entwerfen und diese in der Cloud hosten lassen.

  Dieser Cloud-Service richtet sich hauptsächlich an Entwickler. Ihr Nutzer entwirft die Software zunächst auf lokalen Rechnern und kann sie dann auf Cloud-Servern des Anbieters ausführen lassen, ohne sich um darunterliegende Strukturen kümmern zu müssen [110, 18]. Beispielhaft auf dieser Ebene sind die *Google AppEngine*, die *Microsoft Windows Azure Platform* und die Entwicklungsplattform *Force.com* von *Salesforce*.

- **Infrastructure-as-a-Service:** IaaS dient der Diensterbringung auf unterster Infrastrukturebene. Die Kapazitäten werden in Form von Rechenleistung, Speicher oder Netzwerken bereitgestellt [110]. In dieser ausgelagerten virtuellen Umgebung kontrolliert der Nutzer seine eigene Middleware und seine Datenbanksysteme sowie Applikationsschicht.

  Diese Infrastrukturdienste bieten demnach ein Maximum an Flexibilität bezüglich Installation des Betriebssystems, Festsetzung des Durchsatzes und Größenbestimmung der Datenbankspeicher. Allerdings bedeutet diese Einstellungsfreiheit auch ein hohes Maß an Verwaltungs- und Kontrollaufwand. Vor allem *Amazon* bietet auf dieser Ebene ein großes Repertoire an Diensten wie die *Elastic Compute Cloud* (EC2) oder den *Simple Storage Service* (S3).

Nutzungsmodelle des Cloud Computings

Die Kategorisierung der Cloud-Technologie erfolgt im Folgenden anhand verschiedener Nutzerklassen, welche die erklärten Service-Modelle umsetzen. Genauer gibt es vier organisatorische Ausprägungen der Entwicklungsmodelle.

- **Public Cloud:** Über das Internet wird dem Public Cloud-Anwender der Zugriff auf öffentliche IaaS-, PaaS- oder SaaS-Modelle ermöglicht. Mehrere Benutzer können gleichzeitig und unabhängig dasselbe Angebot aufrufen. Die Cloud-Eigenschaft der Multimandantenfähigkeit realisiert die Unterscheidung der einzelnen Nutzer.
  Teure Vorabinvestitionen in eigene Hard- und Software entfallen durch die Auslagerung in die Cloud des Dienstanbieters. Dieser übernimmt die Verantwortung für Bereitstellung und Wartung der Ressourcen.
- **Private Cloud:** Für eine exklusive Nutzung stehen Private Clouds lediglich ausgewählten Personen einer Organisation zur Verfügung. Nur dieser Anwenderkreis kann unternehmensintern auf die Dienste zugreifen. Die Services werden speziell an die Anforderungen des privaten Kunden angepasst. Das Ressourcenmanagement erfolgt entweder on-premise, sprich in unternehmenseigenen Rechenzentren oder off-premise über Dritte.
  Vorteilhaft beim Einsatz der Private Cloud ist die Kontrolle, die das Unternehmen über die Übertragung und Ablage von Daten und Informationen hat. Dadurch minimieren sich Sicherheitsrisiken. So ist dies eine sinnvolle Alternative, wenn Daten aus rechtlichen Gründen, beispielsweise im Gesundheitswesen, das Land nicht verlassen dürfen [18].
- **Community Cloud:** Das Konzept der Community Cloud dient der Errichtung einer Struktur für die exklusive Nutzung durch eine spezifische Interessensgemeinschaft von Organisationskunden mit gleichen Anliegen. Diese nutzen dann gemeinsam den Ressourcenpool der Cloud, der von einem unabhängigen Anbieter oder von den beteiligten Organisationen bereitgestellt wird. Durch die gemeinsame Nutzung der Ressourcen können die benötigten Kapazitäten reduziert und anfallende Kosten für die Verbreitung der Cloud-Dienste geteilt werden [107].
- **Hybrid Cloud:** Eine Hybrid Cloud stellt eine Kombination der verschiedenen Cloud-Strukturen dar. Sie realisiert die Vorzüge der einzelnen Varianten. Dabei entscheidet das Unternehmen selbst, welche Services über die Public und welche über die Private Cloud genutzt werden. Standardisierte Prozesse mit unsensiblen Daten sind kostengünstiger in der Public Cloud zu verarbeiten. Die Ablage kritischer Daten kann in der sicheren Private Cloud erfolgen. Generell gilt jedoch die Kombination verschiedener Varianten innerhalb ein und desselben Systems als Herausforderung. Es gibt noch kein standardisiertes Verfahren zur reibungslosen Integration von Cloud-Elementen in ein bestehendes on-premise System.

2.2 Cloud-Technologie im praktischen Einsatz

Nachdem wichtige Begriffe des Cloud Computings umrissen sind, widmet sich dieser Abschnitt ihrer unterschiedlichen Betrachtungsweise. Die Cloud bietet einerseits viele Vorzüge, die ihre Bedeutung in der IT-Welt verständlich macht. Andererseits birgt sie Gefahren, die Kritiker enorm riskant einschätzen.

Vorteile durch Cloud Computing

Der größte Nutzen des Cloud Computings ist die fertige Infrastruktur. Sie steht dem Anwender zur freien Verfügung, ohne sie eigenständig warten und betreiben zu müssen. Somit entfällt Installations- und Instandhaltungsaufwand. Da die Ressourcen nach Beanspruchung abgerechnet werden, verringern sich außerdem anfallende Kosten für ungenutzte Kapazitäten. Aufgrund von dynamischer Anpassungsfähigkeit und globaler Verfügbarkeit gewinnt der Cloud-Nutzer an Flexibilität. Im Folgenden werden weitere Vorteile erläutert.

- **Skalierbarkeit und Elastizität:** Elastizität bezeichnet die flexible Anpassung der Ressourcenmenge an den aktuellen Verbrauch. Indem genau das nötige Maß an Maschinen bereitsteht, können Lastspitzen abgefangen werden. Die Instanzen warten nicht im Leerlaufmodus, dem Idle-Modus, auf Anfragen und trotzdem sind alle Nachfragen zu bewältigen [112]. Ein Leerlaufmodus führt zu Unterauslastung und unnötigen Kosten.
  Cloud-Kunden sind also einerseits nicht mehr gezwungen, Ressourcen durch *Überprovisionierung* zu verschwenden. Andererseits laufen sie nicht Gefahr, durch *Unterprovisionierung* zahlende Nutzer zu verlieren [15]. Überprovisionierung entspricht dabei der großzügigen Bereitstellung zu vieler Ressourcen. Dagegen steht Unterprovisionierung für ungedeckte Ressourcennachfrage. In Standardsystemen entsteht diese Situation, wenn die Provisionierungszeit zusätzlicher Kapazität zu lang ist [57].

- **Mobilität und Ortsunabhängigkeit:** Mit der Möglichkeit, alle Daten und entsprechende Anwendungen auf Cloud-Instanzen auszulagern, muss sie der Nutzer nicht mehr lokal speichern. Die Ressourcenkapazitäten sind also nicht mehr ortsgebunden. Der mobile Arbeitsplatz wird somit realisierbar. Ferner sind Informationsaustausch sowie die Datenmodifikation jederzeit auch von entfernten Orten aus möglich [1].

- **Datenredundanz:** Bei der Ablage, Lagerung und Entfernung von Daten kann es jederzeit vorkommen, dass Informationen unwiederbringlich verloren gehen. Sind die Daten nicht repliziert und einmalig, lediglich auf einem lokalen Computer oder einem portablen Endgerät vorhanden, ist dies besonders fatal, wenn es zu einem Plattencrash kommt [112]. Mit der Replikation von Daten in der virtuellen Umgebung des Cloud-Anbieters sind solche Datenverluste vermeidbar.

- **Kostenersparnis:** Die Cloud-Technologie verspricht hohes Potenzial zur Kosteneinsparung. Durch die Auslagerung der IT müssen Kunden nicht mehr in die Beschaffung von Software-Lizenzen und leistungsstarker physischer Hardware investieren. Somit ist ein Wandel von Kosten der Vorabinvestitionen, dem sogenannten CAPEX (CAPital EXpenditure) zu laufenden Kosten, dem sogenannten OPEX (OPerational EXpenditure) möglich [86]. Ebenso reduzieren sich Betriebs-, Wartungs- und Upgrade-Kosten. Zusätzlich können Kosten durch ungenutzte Instanzen vermieden werden, da meist nach dem Pay-per-Use-Modell bezahlt wird.
- **Ressourcenschonung:** Schließlich sind auch im Zuge der Nachhaltigkeit Vorteile erkennbar. Die Umwelt kann ganz im Sinne der Green IT geschont werden, da in Summe weniger Geräte laufen. Nur wirklich benötigte Ressourcen sind im Einsatz, wodurch der insgesamte Energie- und Kühlbedarf der Maschinen gesenkt wird. Das führt zu geringerer CO^2-Emission und reduzierter Umweltbelastung [19].

Einwände gegen das Cloud Computing

Obwohl Cloud Computing Vorteile bietet, birgt der Einsatz ebenso Risiken. Beispielsweise existieren konträre Auffassungen, was die Zuverlässigkeit betrifft. Einerseits gilt die Cloud aufgrund ihrer Beschaffenheit als ausfallsicher im Vergleich zu privaten Infrastrukturen, andererseits bestehen jedoch starke Bedenken im Hinblick auf Leistungsstörungen, Insolvenzrisiko und Kontrollverlust. Zusätzliche Problematiken sind:

- **Abhängigkeit vom Cloud-Anbieter:** Ein aus Anwendersicht oftmals kritisierter Fakt ist die Abhängigkeitsbeziehung, die man mit dem Cloud-Anbieter eingehen muss. Dieser verwaltet Daten und Anwendungen in seinem spezifischen Format. Sie unterliegen folglich seiner Kontrolle. Aus Profitgründen bemühen sich die Betreiber nicht um Standardisierung von Schnittstellen und Datenformaten. Dadurch ist ein kurzfristiger Anbieterwechsel bei einer Preisanhebung durch den ursprünglichen Betreiber kaum realisierbar. Der Transfer von Applikationen und Daten zum günstigeren Anbieter geht mit unverhältnismäßigem Zusatzaufwand und Kosten einher. Dadurch ist der Kunde nachhaltig an den anfänglichen Anbieter gebunden, was als Vendor-Lock-In-Effekt bezeichnet wird [15].
- **Datenverfügbarkeit und Performanz:** Für den Zugriff auf die extern gespeicherten Daten und Anwendungen in der Cloud ist eine dauerhafte Netzanbindung nötig. So spielen auch die Übertragungsgeschwindigkeit der eigenen Infrastruktur und der Übermittlungsdurchsatz, also die Leistungen des Internetanbieters, eine bedeutende Rolle. Vor allem wenn sich verschiedene Kunden dieselbe Hardware teilen, können I/O-Interferenzen die durchschnittliche Performanz negativ beeinflussen [16].

- **Sicherheit:** Bei der Sicherheitsthematik im Cloud Computing liegt das Hauptaugenmerk auf der vertraulichen Behandlung der Daten und ihrer Sicherheit [78]. Durch die Datenauslagerung in die Cloud können unternehmenskritische oder persönliche Informationen nicht mehr durch eigene, interne Mechanismen geschützt werden. So besteht das Risiko unerwünschter Überwachung oder Datenmissbrauchs. Ein Cloud-Nutzer muss sich folglich bezüglich seiner gesamten Datensicherheit auf die Infrastruktur des Anbieters verlassen. Sogar der Dienstanbieter kann nur aus der Ferne Sicherheitseinstellung vornehmen, ohne überprüfen zu können, ob sie tatsächlich umgesetzt werden [191].
  Speziell der Datenschutz wird kritisch betrachtet, da der Kontrollverlust zu Intransparenz über den Speicherort der eigenen Daten führt. Die Einhaltung des Datenschutzrechts ist für den Anwender nicht garantiert, da die Kontrolle an den Cloud-Anbieter übertragen wird. Das erschwert die Überprüfung der Regelungen des Bundesdatenschutzgesetzes [25]. Insgesamt muss hier ein Wandel von Kontrolle zu „Vertrauen“ erfolgen. Der Kunde muss sich auf die Garantien des Anbieters bezüglich Zugriffskontrolle und Authentifizierung verlassen [29, 37, 137].

2.3 Angebote des Cloud-Umfelds

Die Nachfrage nach der Cloud-Technologie steigt. So gibt es auch immer mehr Anbieter, die Cloud-Dienste zur Verfügung stellen. Laut Fersthman et al. [55] dominiert in den USA der Anbieter Amazon im IaaS-Bereich mit einem Marktanteil von 80 bis 90 %. Google und Microsoft sind führend im PaaS- und SaaS-Umfeld. Diese und andere wichtige Anbieter mit ihren jeweiligen Cloud-Angeboten werden nun vorgestellt. Das schafft einen Überblick über die Aspekte der Speicherung, Bedienbarkeit, Performanz und Kosten.

Amazon Web Services

Amazon ist vielfach lediglich als großes Online-Portal mit Web-Diensten bekannt. Das Unternehmen ist aber auch ein Vorreiter des Cloud-Gedankens. Amazon begann damit, die enorm gewachsene Infrastruktur in Zeiten zu vermieten, in denen die Lastspitzen verkaufsstarker Jahreszeiten ausblieben und Ressourcen ungenutzt waren. Inzwischen bedient Amazon verschiedene Cloud-Bereiche mit Rechenleistung, Datenbanken und Verwaltung.
Auf IaaS-Ebene ist vor allem der Rechenleistungsdienst *Elastic Compute Cloud* (*EC2*) bekannt. *EC2* stellt seinen Benutzern virtuelle Server in vier verschiedenen Leistungsstufen. Die nutzerspezifischen Anwendungen werden im on-demand Fall stundenweise abgerechnet. Außerdem können Instanzen vorreserviert werden, um einen sicheren Zugriff auf die Ressourcen zu garantierten.

Dabei fällt eine jährliche Gebühr von 23 US-Dollar für Mikro-Instanzen bis zu 2576 US-Dollar für reservierte Instanzen mit hoher E/A-Aktivität an [7].
Vor dem Start einer *EC2*-Instanz muss der Nutzer ein vorgefertigtes oder selbsterstelltes Amazon Machine Image auswählen. Es bestimmt die Eigenschaften beispielsweise bezüglich des Instanz-Betriebssystems. Dann wird die Ressource konfiguriert, um die gewünschte Rechenkapazität oder Speichergröße und den Standort festzulegen. Abschließend ist die Instanz per Public-Key-Verfahren mit persönlichem Schlüssel zum Start bereit. Eine standardmäßige *EC2*-Instanz ist nicht persistent. Zur dauerhaften Zustandsspeicherung benötigt man eine Anbindung an den Amazon *Elastic Block Store* (*EBS*) [9].
Der hochverfügbare Speicher *EBS* wird in die laufende *EC2*-Instanz eingebunden. Sein Speicherumfang reicht von einem Gigabyte bis zu einem Terabyte. Zu Performanzzwecken werden mehrere Volumen einer Instanz zugewiesen. Dabei sind die Volumen in der Verfügbarkeitszone, der Availability Zone (AZ), automatisch repliziert und existieren unabhängig von Instanzen. Für die sichere Wiederherstellung von Daten nach Ausfällen sind Snapshots anzufertigen. Diese werden auf dem Amazon *Simple Storage Service* (*S3*) gespeichert [6].
Amazon *S3* ist ein hochskalierbarer Cloud-Massenspeicher. In ihm speichern Nutzer über ein Web-Service-Interface große Datenmengen. Mittels *S3* kann eine unbegrenzte Anzahl an Objekten mit einer Größe von einem Byte bis zu fünf Terabyte gespeichert, gelesen und gelöscht werden [11]. Dabei wird für ein Jahr eine 99,999999999-prozentige Zuverlässigkeit und 99,99-prozentige Verfügbarkeit der Objekte zugesichert. Zudem sind Objekte privat oder öffentlich zu speichern und die Rechte zum Zugriff nach einer Authentifizierung per REST-API überprüfbar [7]. Das REpresentational State Transfer ist ein Modell der Programmierung von Web-Diensten. API steht für Application Programming Interface, also für die entsprechende Programmierschnittstelle.
Die Objekte werden in Buckets abgelegt. Sie sind eindeutig mit einem Bucket-Key und dem Objekt-Namen identifiziert [18]. Die Regionen für die Speicherung dieser Buckets werden vom Nutzer selbst festgelegt. Damit können die Daten nicht ohne Wissen des Nutzers in andere Regionen transferiert werden. Grundlegend sichert der Anbieter dem Nutzer eine abschließende Konsistenz zu. Dies wird Eventual Consistency genannt [11].
Zusätzlich bietet Amazon den *Relational Database Service* (*RDS*). Er ist die kostengünstigere Variante des *S3* zum Betrieb relationaler Datenbanken. Die Daten werden zwar weniger oft repliziert, doch wird für ein Jahr eine Objekt-Zuverlässigkeit und Verfügbarkeit von 99,99 % per SLA garantiert. Weiterhin gibt es die Möglichkeit, sogenannte Multi-AZ-Datenbankinstanzen zu erstellen. Dabei schafft *RDS* automatisch eine primäre Instanz und repliziert die Daten synchron zu einer Standby-Instanz einer anderen Verfügbarkeitszone [10].
Im Bereich der Datenbanken gibt es auch die *SimpleDB*. Sie ist neben Verfügbarkeit und Zuverlässigkeit eher auf Einfachheit optimiert und nicht auf transaktionale Operationen ausgerichtet. Trotz hoher Skalierbarkeit und Flexibilität, eignet sie sich also nicht zur Erstellung relationaler Datenbanken und

für komplexe Datenhaltung. Vorteilhaft ist jedoch, dass ein nur geringer Administrationsaufwand besteht, da die Daten vom Nutzer lediglich über einfache Web-Dienstanfragen gespeichert und abgefragt werden [12].
Daneben stellt Amazon ein umfangreiches Angebot zur Verfügung: angefangen von Accountmanagement- und Billing-Systemen für Anwendungen per *DevPay* und *Flexible Payments Service*, über Content Distribution Networks per *CloudFront* und dem Messaging Service *Service Simple Queue* bis zu einem gehosteten *Hadoop*-Framework [7]. Das freie, Java-basierte Apache *Hadoop* eignet sich gut für intensive Rechenprozesse mit großen Datenmengen [177].

Microsoft Windows Azure

Neben Amazon hat sich Microsoft stark auf dem Cloud-Markt etabliert. Es bietet eine Plattform für das bedarfsgerechte Anmieten von Rechenleistung und Speicherplatz sowie die Entwicklung oder Nutzung von Cloud-Anwendungen. Die Ressourcen und Dienste stehen flexibel mit hoher Verfügbarkeit und Skalierbarkeit bereit. Das Cloud-Betriebssystem von Microsoft heißt *Windows Azure* und bedient die Laufzeitumgebung der Plattform. Durch den *Fabric Controller* wird die Zuverlässigkeit überwacht und die Umgebung verwaltet. Die *Windows Azure Platform* bietet verschiedene Komponenten als Dienst an, wie *Compute*, *Data Services*, *Networking*, *App Services* und *Commerce* [118].
Die Windows Azure *AppFabric* erweitert die Windows Server um Hosting, Verwaltung und Caching der Web-Anwendungen und Dienste. So erleichtern die Features der Internet-Informationsdienste (IIS) und des Windows Prozess-Aktivierungsdiensts (WAS) die Konfiguration der Applikation. Durch das Hinzufügen eines verteilten Objekt-Caches im Arbeitsspeicher der Windows Server verbessert sich die Anwendungsskalierung [113].
Über *Compute*, auch *Execution Model* genannt, können *Virtual Machines*, *Web Sites* sowie *Cloud-* und *Mobile Services* genutzt werden. Für die Implementierung von Anwendungen werden per Software Development Kit drei sogenannte Rollen bereitgestellt. Die Web-Rolle dient als Container der Web-Anwendung und die Worker-Rolle verarbeitet im Hintergrund rechenintensive Aufgaben. Die VM-Rolle stellt die Infrastruktur mit virtuellen Maschinen, die auf Windows Servern 2008 R2 in der Cloud laufen [120]. Mit Microsoft *Exchange*, *SharePoint* und *Office 365* wird auch die Kategorie Software-as-a-Service bedient. Die Nutzung der Cloud-Dienste wird monatlich abgerechnet [1].
Im Bereich der *Data Services* stellt Windows Azure für die unterschiedlichen Anforderungen der Daten verschiedene Speichermöglichkeiten bereit. Unterschieden wird zwischen *Binary Large Objects* (*Blobs*), *Tables* und der *SQL Database*. In *Blobs* werden in einfacher Hierarchie ohne Schachtelung unstrukturierte, binäre Daten beispielsweise eines Videos abgespeichert. Die Größe eines *Blobs* beläuft sich auf maximal ein Terabyte. Die Speicher-Accounts kön-

nen in Containern mehrere *Blobs* gruppieren. Zur persistenten Speicherung von Daten wird ein Windows Azure *Drive* an einen *Blob* angeschlossen [118].
Tables lassen die Speicherung von verschiedenen strukturierten Datentypen, wie Strings oder Integer-Werten, im Key/Value-Modus zu. Diese *Tables* sind nicht vergleichbar mit Tabellen eines relationalen Modells, sondern vielmehr ein Beispiel eines NoSQL-Speicherverfahrens. Zwar sind komplexe Operationen wie Joins nicht möglich, dafür wird der Zugriff auf die Daten außerordentlich schnell umgesetzt. Diese hochskalierbaren Tabellen können Billionen von Datensätzen, sogenannten Entities, und Terabytes an Daten speichern [115].
Wird ein relationales Datenbankmodell für die Anwendung benötigt, ist auf die *SQL Database* zurückzugreifen, die auf SQL-Servern betrieben wird. *SQL Database*, auch oft als *SQL Azure* bezeichnet, bietet Funktionalitäten eines relationalen Datenbankmanagementsystems, wie Ausführbarkeit atomarer Transaktionen und paralleler Datenzugriff für mehrere Nutzer bei gleichzeitiger Datenintegrität. Obwohl dabei die *SQL Database* die grundlegende Administration bezüglich der Hardware und Datenaktualisierung übernimmt, behält der Benutzer trotzdem die Kontrolle über den Zugriff auf die Daten [116].
Zur Business Analyse kommt oft das *SQL Reporting* und *Hadoop* zum Einsatz. Damit können gespeicherte Daten als Basis für Geschäfts-Reports verwendet werden und die Analyse von großen Datenmengen ist performant realisierbar [117]. Weiterhin stellen die *Networking Services* generell die grundlegende Verbindung per Windows Azure *Virtual Network* und die maßgebliche Nachrichtenbalancierung per *Traffic Manager* sicher [118].
Der *Service Bus* und der *Queue*-Service, also eine Warteschlange, kümmern sich um den verlässlichen asynchronen Nachrichtenaustausch zwischen Rollen-Instanzen [117]. Ein Vorteil der Queue-Architektur liegt in der hohen Fehlertoleranz. Bei großer Belastung des Kommunikationsnetzes, werden Pakete in der Warteschlange gepuffert. Beim Ausfall eines Knotens werden die Anfragen hierin gehalten, bis weitere Ressourcen zur Verfügung stehen [114].

Salesforce.com

Salesforce gehört wie Amazon zu den Wegbereitern der Cloud-Angebote. Das Unternehmen stellt vor allem Produkte für Vertrieb, Kunden-Service und Applikationsentwicklung bereit [147]. Zum einen bedient Salesforce mit Lösungen zum Enterprise Resource Planning und Customer Relationship Management die Sparte der Geschäftsanwendungen auf SaaS-Ebene. Andererseits bietet *Force.com* eine Entwicklungsumgebung auf PaaS-Ebene [1].
Grob wird bei Salesforce unterschieden zwischen der *Sales Cloud*, der *Service Cloud*, *Force.com* und *Chatter*. Die Vertriebsanwendungen der *Sales Cloud* stellen Unternehmen Funktionalitäten für Vertrieb und Marketing bereit. Die *Service Cloud* dient dem Kundenservice-Management und schafft Vorteile durch die Nutzung sozialer Medien im Unternehmen. Die Entwicklung von Anwen-

dungen für soziale Netzwerke und mobile Endgeräte erfolgt über die Cloud-basierte Plattform *Force.com*. Den einfachen Austausch von Ideen und Dokumenten innerhalb eines Unternehmens in Echtzeit ermöglicht *Chatter* [147]. Die Programmierung erfolgt über Sprachen wie Java, Ruby, C# und PHP und es wird pro Monat abgerechnet [1, 62].

Die Datenbankebene von Salesforce heißt *Database.com*. Diese verspricht hohe Elastizität, Skalierbarkeit, Sicherheit und automatische Backups bei wenig Administrationsaufwand. Der *Database*-Service ist nach der Registrierung mit einem Account als relationaler Datenspeicher nutzbar. Er unterstützt verschiedene Datentypen sowie Prozeduren und ein Sicherheitsmodell, über das der Datenzugang mit jedem API-Aufruf überprüft wird. Die Speichermöglichkeiten reichen von Dokumenten über umfangreiche User-Profile bis zu Videos und Bildern, die in relationalen Datenstrukturen abgelegt werden. Die Speicherkapazität für Daten ist quasi unbegrenzt. Bereits existierende Datenbanken sind mittels Migrationsskripten in *Database.com* integrierbar [40].

Punkten kann *Database.com* vor allem mit seinem Preismodell: Drei Nutzer mit bis zu 100.000 Datensätzen und 50.000 Transaktionen pro Monat sind kostenlos. Weiterführende Services für die automatische Administration der Daten und des Ablagesystems sind jedoch gebührenpflichtig. Jeder zusätzliche Nutzer kostet 10 US-Dollar pro Monat. Alle weiteren Blöcke mit je 100.000 Datensätzen und 150.000 Transaktionen werden mit 10 US-Dollar pro Monat abgerechnet. *Database.com* eignet sich vor allem für soziale und mobile Applikationen, die gemeinsame Zusammenarbeit unterstützen [41].

Google

Google bedient verschiedene Sparten der Cloud-Technologie. Zum einen bietet Google die sogenannten *Apps for Business* für die kollaborative Arbeit in einem Team mit einfacher Einrichtung, Verwendung und Verwaltung [74]. Die SaaS-Dienste im Sektor der Office-Anwendungen umfassen *Google Mail*, *Google Calendar*, *Google Drive*, *Google Docs* und vieles mehr [44].

Daneben kann der Nutzer mit der *Google App Engine* auf eine Entwicklungsplattform zugreifen, um eigene Anwendungen zu entwickeln und zu betreiben. Es wird neben den Programmiersprachen Java und Python, auch die freie, von Google entwickelte Programmiersprache Go unterstützt, die höhere Leistung bei rechenintensiven Anwendungen verspricht. Im Backend können große und speicherintensive Applikationen vom Entwickler gezielt gestartet und gestoppt werden. Anstatt pro CPU-Stunde zahlen die Nutzer für Instanz-Stunden [83]. Problematisch ist bei Google, dass der Nutzer keine Kontrollmöglichkeit darüber hat, wo die Daten abgelegt werden. Ebenso bekommt er keinen Einblick über die Verwaltung der Rechenzentren und Daten [150].

Die *Google App Engine* nutzt den *Blobstore* als Datenspeicher. Übersteigen die Daten die zulässige Größe für den eigenständigen Google-Speicherdienst,

werden den Anwendungen Datenobjekte, sogenannte *BLOBs*, über die *Blobstore*-API bereitgestellt. Die *BLOBs* entstehen durch das Hochladen von Dateien per HTTP-Anfrage über die Anwendungen. Dafür füllt der Nutzer ein Formular mit dem Feld für den Datei-Upload aus. Bei Formular-Übermittlung werden dann durch den *Blobstore* aus den Datei-Inhalten *BLOBs* erzeugt und eine Referenz als *BLOB*-Schlüssel abgelegt.
Soll ein *BLOB*-Wert erfragt werden, legt die Anwendung einen Header auf die ausgehende Antwort. Diese wird dann durch die *App Engine* mit dem tatsächlichen *BLOB*-Wert ersetzt. Erzeugte *BLOBs* können gelöscht, jedoch nicht geändert werden [72].
Zur Datenspeicherung bietet Google den *Cloud Storage*. Über den REST-basierten Dienst können Daten auf der Google-Infrastruktur in einem skalierbaren hochverfügbaren Objektspeicher abgelegt und schnell abgerufen werden. Diese Infrastruktur kann über die *Compute Engine* auch als alleinstehender Dienst genutzt werden. Er lässt die Verarbeitung von hoher Last auf Linux-basierten virtuellen Maschinen zu. Per *BigQuery*-Dienst kann eine interaktive Analyse von enorm großen Datenmengen erfolgen [73].

Analyse und Bewertung der Datenspeicher- und Datenbank-Lösungen

Im Umfeld der Datenspeicher müssen gewisse Anforderungen erfüllt sein, um eine korrekte Arbeitsweise der Datenbanken und der korrespondierenden Managementsysteme zu garantieren. In Datenbanken werden im Mehrbenutzerbetrieb zur Abfrage und Änderung von Daten Transaktionen verwendet, die wie eine Einheit unterbrechungsfrei als eine Folge von Operationen ausgeführt werden. Dabei kommen die vier sogenannten ACID-Eigenschaften zum Tragen. Dies steht für Atomicity, Consistency, Isolation und Durability.
Sie besagen, dass Transaktionen nicht zerlegbare, atomare Einheiten sind, die ganz oder gar nicht ausgeführt werden. Dabei wird in der Ausführung die Datenbank von einem konsistenten Zustand in einen anderen konsistenten Zustand überführt oder im Fehlerfall in den Anfangszustand zurückgesetzt. Parallele Transaktionen dürfen sich in diesem logischen Einbenutzerbetrieb nicht gegenseitig beeinflussen und erfolgen isoliert. Wenn eine Transaktion erfolgreich war, werden die Änderungen dauerhaft gespeichert [90].
Obwohl diese grundlegenden Forderungen an Transaktionen in verteilten Datenbanksystemen wünschenswert sind, lässt sich nach dem CAP-Theorem bewiesenermaßen nicht alles gleichzeitig vollständig realisieren [138]. Das CAP-Theorem von Brewer zeigt, dass zu einem Zeitpunkt die Einhaltung von nur zwei der drei wichtigsten Eigenschaften eines verteilten Systems, Konsistenz, Verfügbarkeit und Partitionstoleranz, garantierbar ist [23]. Mathematisch ist bewiesen, dass es unmöglich ist, ein Datenmanagementsystem zu schaffen, das

zur gleichen Zeit immer verfügbar sowie partitionstolerant ist und zudem dauerhafte Konsistenz der Daten gewährleistet [66].

Cloud-Anbieter achten bei ihren Diensten hauptsächlich auf Skalierbarkeit, Verfügbarkeit und Kosteneffizienz und machen dafür Abstriche bei der Konsistenz. So entstand als Alternative zum ACID-Prinzip der traditionellen relationalen Datenbanken für den Cloud-Bereich das BASE-Konzept, was für Basic Availability, Soft-state und Eventual consistency steht. Damit wird garantiert, dass ein Dienst immer verfügbar ist und aus Effizienzgründen auch das Lesen von Zwischenzuständen der Daten ermöglicht wird. Hierbei wird keine dauerhafte Konsistenz nach jeder Transaktion gefordert, sondern abschließende, letztendlich konsistente Zustände sind ausreichend [24].

Im Folgenden wird in Anlehnung an die Ergebnisse von Schwanengel [152] eine ausführliche Analyse der erläuterten Lösungen bezüglich der Einhaltung des ACID-Konzepts beschrieben. Die Bewertung erfolgt über die Kriterien der Performanz, Konsistenz, Skalierbarkeit, Verfügbarkeit, Zuverlässigkeit, Replikation, Fehlertoleranz sowie der Kosten.

Wie erwähnt, lässt die Amazon *SimpleDB* im Standardfall Eventually Consistent Reads zu. Obwohl diese Datenbank dabei die drei ACID-Eigenschaften, Atomarität, Isolation und Dauerhaftigkeit garantiert, ist sie nicht vollständig ACID-konform, da Konsistenz nicht zu jedem Zeitpunkt eingehalten wird.

Hingegen sind im Modus der Multi-AZ-Bereitstellung beim Amazon *RDS* alle ACID-Kriterien, also auch die Datenkonsistenz, erfüllt. Diese Cloud-Datenbank ist folglich ACID-konform. Auch Microsoft *SQL Azure* entspricht dem ACID-Prinzip. Die verteilten Transaktionen werden bei mehreren Server-Verbindungen per gespeicherten Connection Strings unterschieden. Falls der korrespondierende Connection String exakt der aktuell betrachteten und bearbeiteten Verbindung entspricht, wird die Transaktion nicht ausgeführt.

Beim Google *App Engine Datastore* ist Datenkonsistenz je Entity Group gewährleistet. Transaktionen werden mit optimistischer Concurrency Control ausgeführt und Änderungen für ein Zeitintervall so lange wiederholt, bis konkurrierende Prozesse beendet sind. Folglich wird eine Operation entweder ganz oder gar nicht ausgeführt. ACID-Eigenschaften sind garantiert. Salesforce macht keine SLA-Angaben zur Datenkonsistenz in *Database.com*. Für diese Datenbank ist nicht klar zu entscheiden, ob sie ACID-konform ist.

An Datenbanksysteme werden häufig neben ACID- und CAP-Charakteristika noch weitere Anforderungen gestellt, wie beispielsweise verschiedene Ausprägungen der Transparenz und der Unabhängigkeit [90]. Aspekte der Datenunabhängigkeit, Unabhängigkeit von zentralen Systemfunktionen, Hardware-, Betriebssystem- und Netzunabhängigkeit sind durch die Eigenschaften der Cloud-Technologie per Definition bei allen Lösungen gegeben.

Aufgrund der Cloud-Technologie wird außerdem von allen vorgestellten Anbietern auch Verteilungs-, Orts- und Fragmentierungstransparenz per se erfüllt. Je nach unterstützter Konsistenz ist zusätzlich Replikationstransparenz gegeben.

Da mit wachsender Anzahl der Replikate die Konsistenzwahrung erschwert wird, ist zwischen Replikation und Konsistenz abzuwiegen.
Abhängig von der Verteilung auf verschiedene Verfügbarkeitszonen, schwankt auch die Leistungstransparenz. Dies ist durch Speicherung der Daten in verschiedenen Bereichen bedingt. Da bei großen Entfernungen unvorhersehbare Latenzen in der Datenübertragung entstehen, ist unabhängig von den beschriebenen Betreibern die Leistungstransparenz selten erfüllt.
Die abschließende Bewertung zeigt, welche Datenbank sich in welchem Szenario eignet. Amazon *SimpleDB* wird wegen der hohen Skalierbarkeit und Flexibilität sowie des geringen Verwaltungsaufwands häufig im Sektor der Online-Spiele eingesetzt. Auch bei der Metadaten-Indizierung von *S3*-Objekten erweist sich die *SimpleDB* als äußerst geeignet. Dabei können Entwickler Metadaten der *S3*-Objekte in der *SimpleDB* speichern und die preiswerte Datenbank-Suchfunktionalität von *S3* nutzen. In der *SimpleDB* ist es allerdings nur möglich, Strings zu speichern und zu verarbeiten. Dabei gilt keine vollständige Konsistenz der Daten. Eine Domäne fasst maximal zehn Gigabyte.
Werden Berechnungen oder Join-Operationen benötigt, fällt die Wahl eher auf eine andere Datenbank. Sonst muss *S3* integriert werden, um größere Datenobjekte zu speichern. Werden die Funktionalitäten einer relationalen Datenbank benötigt, wird oft auf den Amazon *RDS* zurückgegriffen, da hier die Standardfunktionen der *SimpleDB* nicht ausreichen. Da *RDS* eine gute Bedienbarkeit und geringen Verwaltungsaufwand hat, wird es vor allem bei der Verarbeitung komplexer Daten im relationalen Modell verwendet.
Vorteilhaft an Microsoft *SQL Azure* ist die hohe Skalierbarkeit und die Fähigkeit, strukturierte wie auch unstrukturierte Daten zu verarbeiten. Diese relationale Datenbank bietet Zusatzfunktionen durch SQL-Server-Unterstützung. Hauptsächlich eignet sie sich für die Einbindung in Anwendungen, die auf einem reinen Windows-Stack aufbauen [88]. Da die Datenreplikation jedoch nur auf Cloud-Level erfolgt, ist das Anlegen eigener Kopien oder die Änderung bereits bestehender Server-Konfigurationen nicht möglich.
In Anwendungsbereichen mit stetig wachsendem Datenbestand, stellt der Google *App Engine Datastore* eine herausragende Skalierbarkeit. Obwohl er Transaktionen unterstützt und Abfragen zulässt, ist er allerdings keine eigenständige relationale Datenbank. So fehlt beispielsweise die Funktionalität des Joins. Zudem ist die Anzahl der Entitäten eines Typs nicht bestimmbar. Da Google im Unterschied zu anderen Anbietern wenige Informationen zum Speicherort der Daten und ihrer Sicherheit offenlegt, ist diese Lösung zur Verarbeitung kritischer Daten ungeeignet [77].
Database.com wird von Salesforce explizit auf die Anforderungen der zahlenden Unternehmen entwickelt und tritt somit nicht direkt in Konkurrenz zu den anderen Datenbanken, die eher auf die breite Masse abzielen. So eignet sie sich für die Entwicklung von Geschäftsanwendungen und stellt mit dem speziellen Datenmodell ein gutes Entwicklungsumfeld für Anwendungen mit Profilen, Gruppen und Zusatzfunktionen für soziale Applikationen.

Weitere Cloud-Angebote

Im Folgenden wird ein kleiner Ausschnitt weiterer kommerzieller und frei verfügbarer Cloud-Lösungen vorgestellt. Das Apache-lizenzierte Software-Projekt *OpenStack* wurde ursprünglich von Rackspace und der NASA angestoßen und wird heute von unzähligen Anbietern, wie unter anderem SUSE, Dell, Intel, Red Hat und IBM unterstützt. Die Entwicklungssprache ist Python. Da die Hypervisoren Xen, der Linux-basierte KVM und die Virtualisierungsplattform Hyper-V für Windows Server 2008 und Windows 8 verfügbar sind, eignet sich *OpenStack* gut, um eine eigene Cloud-Umgebung aufzusetzen.

Ein Hypervisor, auch Virtual Machine Monitor genannt, bezeichnet die Softwareschicht der Virtualisierung. Er bewerkstelligt die gleichzeitige Ausführung mehrerer virtueller Maschinen, kurz VMs, sowie ihre Steuerung auf dem zugrundeliegenden physischen Rechner. Dafür werden die Hardware-Ressourcen eines Rechners bezüglich der verfügbaren RAM-, Prozessor-, Ein-/Ausgabe-Kapazitäten und die übrigen Komponenten transparent aufgeteilt [36].

VMware bietet zum einen das Software-Paket *vCloud* an, das es ermöglicht, eine eigene Cloud-Infrastruktur zu etablieren. So ist die Last zwischen der Private Cloud und der kompatiblen Public *vCloud* verschiebbar. Zum anderen gibt es im PaaS-Bereich das Angebot *Cloud Foundry* mit den Ausprägungen *Cloudfoundry.org* als Open-Source-Lösung in Ruby und *Cloudfoundry.com* basierend auf *vSphere* und durch VMware lizenziert.

Eine Erweiterung der PaaS-Lösung *Cloud Foundry* bietet *Stackato* von ActiveState. Es erlaubt den Betrieb eines privaten Platform-as-a-Service-Systems auf *vSphere* von VMware sowie einer öffentlichen PaaS auf Amazon oder Rackspace. Es kann verschiedene PaaS-Umgebungen realisieren.

Rackspace Inc. stellt außerdem die eigene *Rackspace Cloud* bereit. Die Dateien ähneln aufgrund ihrer REST-API dem Amazon *Simple Storage Service*. Diese Cloud normiert Systemschnittstellen auf eine standardisierte Menge von HTTP-Aktionen wie Get, Put oder Post und setzt auf ein zustandsloses Client-Server-Protokoll auf [56]. Zudem ist die *Rackspace Cloud* mit einem Akamai Content Delivery Network (CDN) kompatibel. Ein solches CDN stellt ein Netz aus lokal verteilten, über das Internet verbundenen Rechnern dar, worüber Inhalte und umfangreiche Mediendateien verteilt werden. Das Unternehmen Akamai bietet in diesem Kontext eine umfassende Plattform für eine hochperformante Auslieferung an. Es deckt heutzutage in etwa 15 bis 20 Prozent des gesamten Web-Datenverkehrs [128].

Die XEN- und Citrix-basierten *Rackspace Cloud*-Server gleichen den Amazon *EC2*-Rechnern und es werden unter anderem PHP, Python, MySQL, .NET und Microsoft SQL Server 2008 unterstützt.

Eher im IaaS-Bereich anzusiedeln ist die Lösung *SmartCloud Enterprise* (*SCE*) von IBM. Sie bewahrt festgelegte Service-Level-Agreements und ist als Private, Public oder Hybrid Cloud-Modell mit eigenen IBM-Servern umsetzbar.

Auch Oracle bietet Cloud-Unterstützung auf IaaS-Ebene. Das Produkt

Exadata ist die Datenbankvorrichtung zur Unterstützung von sowohl OLTP-Systemen für transaktionsbasierte Vorgänge als auch von OLAP-Systemen für analytische Reporting-Funktionen [130]. Die Oracle *Exalogic* entspricht einer Systemtechnik, die Rechen- Netzwerk- und Speicherkapazität umfasst [129]. Außerdem bietet Oracle auf dem PaaS-Layer noch beispielsweise WebLogic-as-a-Service und im SaaS-Bereich Systeme zum Customer Relationship Management, Human Capital Management und Enterprise Resource Planning.

Häufig werden von den Anbietern gemischte Cloud- und Nicht-Cloud-Umgebungen – im Englischen als off-premise beziehungsweise on-premise bezeichnet – unterstützt. So stellt *CastIron* zum Beispiel eine Plattform für die Integration von Cloud-basierten Anwendungen von führenden SaaS-Anbietern mit on-premise Software von IBM, Amazon, Google Apps, Salesforce.com oder Oracle CRM. Folglich eignet es sich gut für die Verknüpfung von Public Clouds und betriebsinternen Private Cloud-Systemen.

2.4 Zusammenfassung

Die Cloud-Technologie hat sich in den letzten Jahren enorm entwickelt. In Fachkreisen ist sie inzwischen ausgesprochen akzeptiert. Hauptsächlich wird zwischen den drei Service-Kategorien *Software-as-a-Service* (SaaS), *Platform-as-a-Service* (PaaS) und *Infrastructure-as-a-Service* (IaaS) unterschieden.

Weiterhin bieten sich vier Nutzungsmodelle. Die *Public Cloud* stellt eine kostengünstige Umgebung für die breite Öffentlichkeit. Hingegen dienen die Ressourcen der *Private Cloud* nur einem bestimmten zahlenden Nutzerkreis. Eine *Community Cloud* wird von mehreren Organisationen mit ähnlichen Anforderungen gemeinsam in Anspruch genommen. Die *Hybride Cloud* bildet schließlich einen Zusammenschluss der beschriebenen Strukturen.

Die Cloud bietet dem Nutzer eine funktionsfertige Computing-Basis. Diese kann er bedarfsgerecht verwenden, ohne sie selbst betreiben zu müssen. Das reduziert Installations- und Instandhaltungsaufwand. Gleichzeitig erhöht sich die Systemflexibilität durch Elastizität und Skalierbarkeit. Allerdings besteht eine konträre Auffassung was Zuverlässigkeit und Sicherheit der Cloud betrifft. Einerseits verspricht man sich im Vergleich zu privaten Infrastrukturen höhere Ausfallsicherheit aufgrund von Replikation, andererseits bestehen Zweifel im Hinblick auf Transparenz und Kontrollverlust über Daten und Prozesse.

Neben renommierten Forschern versuchen auch etablierte IT-Anbieter dieser Kritik entgegenzuwirken. Beispielsweise bieten Amazon, Microsoft und Google Lösungsansätze mit ihren Cloud-Systemen. Dabei stellt sich die Frage, wie benötigte Ressourcen zu verwalten und überwachen sind, um effizient und umweltschonend die entsprechenden Anforderungen umzusetzen. Das nächste Kapitel widmet sich dem diesbezüglichen Stand der Technik.

3 Arbeitslastmanagement in Cloud-basierten Systemen

In verteilten Systemen existieren verschiedene Arten von Arbeitslast. Sie kennzeichnen die korrespondierende Beanspruchung der Computing-Umgebung. Zur Ermittlung des Nutzungsumfangs, werden beispielsweise die Prozessorverwendung, die Arbeitsspeicherauslastung und die Festplattenbelegung herangezogen. Überfüllte Zustände dieser Komponenten führen zu erheblichen Einbußen der Performanz. Weiter spielt die Datenübertragungsrate der Netzwerkschnittstelle, die durch die Architektur vorgegeben ist, eine wichtige Rolle. Außerdem fällt die Anzahl der ein- und ausgehenden Nachrichten einzelner Systemteilnehmer ins Gewicht [42].

Im Verlauf der Arbeit wird synonym zum Begriff Nachricht auch der Ausdruck Request oder Anfrage verwendet. Der Ausdruck Last bezeichnet das Maß der Systembeanspruchung, die auf verschiedene Weise messbar ist.

Lastmanagement beschreibt in dieser Arbeit die Organisation von Last und ihre notwendige Balancierung. Es bezieht sich folglich auf die Verteilung der an das Gesamtsystem gestellten Aufgaben zwischen seinen Einzelteilen. Als Maßstab für die Verteilung dient die ermittelte Arbeitslast auf den jeweiligen Systemkomponenten, also den Ressourcen. Dabei stellt sich die Frage, wie der nötige Umfang an virtuellen Maschinen organisiert werden muss, um die anliegende Last möglichst ausnahmslos zu verarbeiten.

Laut Assunção et al. [17] ermöglicht die Verwendung virtueller Maschinen durch die Abstraktion und Bündelung der Kapazität mehr Kontrolle über die verwendeten Ressourcen. Die Technologie der Virtualisierung ist stark mit dem Konzept des Cloud Computings verknüpft. Die erforderlichen Instanzen werden bei diesem Ansatz auf einem feingranularen Level hinzugefügt und wieder abgegeben. Auf diese Weise ist die Arbeitslast genauer auf das bestehende Ressourcenvolumen abzubilden [16].

Wie im vorangegangenen Kapitel zu erkennen, bieten die verschiedenen Anbieter die unterschiedlichsten Modelle an, um Cloud Computing zu realisieren, abzurechnen und zu regulieren. Bei der Analyse von Kostenmodellen der relevanten Anbieter wird deutlich, dass die monatliche Abrechnung von vielen Faktoren abhängt [153]. Wichtig sind dabei das Volumen reservierter CPU's und die Dauer der Belegung virtueller Maschinen. Weiterhin kann es sein, dass sich die Preise nach der Größe der Datenbank richten beziehungsweise nach der Menge dort gespeicherter Datensätze. Mit in die Rechnung können auch verfügbare Bandbreite und die Summe an Transaktionen einfließen.

Aufgrund seines Reservierungsmodells für Instanzen bietet Amazon den günstigsten Preis zur Erstellung einer Anwendung, solange diese mehr als 6,5 Stunden CPU-Zeit pro Tag erfordert. Unterhalb dieses Schwellwerts ist die Google App Engine preiswerter, da die Anwendungen zu Anfang umsonst betrieben werden können. Speziell für Windows-Instanzen verlangt Microsoft Azure weniger als Amazon [9, 71, 121].
Zusammenfassend zeigt sich, dass eine große Menge an Faktoren existiert, die Kosten für den Nutzer bedeuten und dabei von Cloud-spezifischen Lastaspekten abhängen. Um allumfassende Annahmen über die zukünftige Arbeitslast treffen zu können, ist es wichtig, alle beeinflussenden Parameter zu berücksichtigen und diese in einem Gesamtbild zu skizzieren. Im Folgenden werden bestehende Mechanismen zum Ressourcenmanagement für die Bewältigung der erzeugten Last in verteilten Systemen analysiert.

3.1 Statisches IT-Ressourcenmanagement

Die Notwendigkeit des Ressourcenmanagements besteht seit Anbeginn der IT-Ära. Da die Nachfrage seither stetig wuchs, sind effiziente Algorithmen und Systeme nötig, um alle Nutzer bedienen zu können. Eine pragmatische und auch die älteste Lösung zur Kapazitätsplanung stellt ein statisches IT-Ressourcenmanagement dar. Dabei wird die Anzahl der benötigten Instanzen im Voraus auf Basis der Service-Level-Agreements definiert, festgelegt und während des Betriebs nicht geändert.
Dieses Reservierungsvorgehen wird zum Beispiel von Zhang et al. [21, 190] im Resource reSerVation Protocol (RSVP) angewandt. Das Hierarchical Mobile RSV Protocol (HMRSVP) von Tseng et al. [171] erweitert diesen Ansatz noch um Aspekte der Mobilität. In [53] wird ein System beschrieben, das mit statischer Vorab-Reservierung des Übertragungskanals arbeitet, um die Übermittlung auf dem Kanal in Echtzeit zu ermöglichen.
Obwohl diese Lösungen einen Ansatz zum Arbeitslastmanagement darstellen, vernachlässigen sie die effiziente allumfassende Ressourcenplanung, da sich die Reservierungen vor allem auf die Netzübertragung beziehen. Andere Komponenten wie Speicher oder Rechenleistung können dabei nicht reserviert werden, was dem heute gängigen Cloud-Paradigma der Diensterbringung widerspricht.
Werden außerdem mehr IT-Ressourcen benötigt als im Service-Level-Agreement vereinbart, kann dieser Mehrbedarf nicht gedeckt werden [142]. Folglich ist die Anfragenbearbeitung in Bursting-Situationen mit ihren kurzzeitigen Ausschlägen zu hohen Lastspitzen enorm erschwert.
Auch die Warteschlangentheorie wird oft zur Bestimmung der nötigen Ressourcenmenge eingesetzt. Damit wird ermittelt, wie stabil die Umgebungen mit der festgelegten Kapazität laufen [91]. Gemäß dem Satz von Little ist für eine gegebene Rate eingehender Nachrichten λ die Zeit, die diese Anfragen im System verbringen, proportional zur Auslastung [100].

Somit lässt sich mit der gemittelten Verweildauer T, auch Delay genannt, je Nachricht die durchschnittliche Anzahl von Nachrichten im System errechnen:

$$N = \lambda \cdot T$$

Weiterhin ist mithilfe der Ankunftsrate λ und Bedienrate μ eingehender Nachrichten je Zeiteinheit die durchschnittliche Systemauslastung $\rho = \frac{\lambda}{\mu}$ zu berechnen. Die Systemlast L und die Last der Warteschlange $L_Q = \rho \cdot L$ werden dann folgendermaßen ermittelt:

$$L = \frac{\lambda}{\mu - \lambda} \quad , \qquad L_Q = \frac{\lambda}{\mu} \cdot \frac{\lambda}{\mu - \lambda}$$

Außerdem sind mit μ und λ die durchschnittlichen Wartezeiten im System $W = \frac{1}{\mu - \lambda}$ und in der Warteschlange $W_Q = \rho \cdot W$ abzuleiten. Dabei gilt generell, dass die Bedienrate μ größer sein muss als die Ankunftsrate λ, da andernfalls mit $\mu \leq \lambda$ die Warteschlange ins Unendliche anwächst [100].
Allerdings sind solche Ansätze heutzutage unbrauchbar, da die Ressourcenreservierung lediglich den Teil der Netzübertragungen abdeckt. Lastspitzen können hierbei kaum abgefangen werden. Die in modernen Cloud-Systemen häufig auftretenden Bursting-Situationen sind somit nicht ausreichend effizient behandelbar. Zudem bleibt bei geringer Auslastung viel Kapazität ungenutzt. Dies hat Ressourcenvergeudung zur Folge.

3.2 Dynamisches IT-Ressourcenmanagement

Eine Verbesserung des Ressourcenmanagements ergibt sich durch eine dynamischere Herangehensweise. Ein möglicher Ansatz besteht in der Adaption auf Client- und Service-Seite. Dies wird zum Beispiel mit dem Toolkit 'SWiFT' möglich [68, 69]. Das System passt sich automatisch dynamischen Laständerungen an. Das kontroll-theoretische Framework basiert auf Netzwerk-Feedback und modularen, rekonfigurierbaren und voraussehenden Kontrollelementen.
Li et al. [97] bieten mit 'QualProbes' eine Middleware an, die mittels Reservierung und Konfiguration von adaptiven Anwendungen eine hohe Dienstgüte, also Quality-of-Service, realisiert. Solche Dienste können Varianzen der Performanz aufgrund von Änderungen in der Ressourcenerreichbarkeit aufzeichnen. Dann sind sie in der Lage, Anwendungen effizienter zu steuern.
Durch dynamische Ressourcenzuschaltung erreichen Iqbal et al. [49] eine maximale durchschnittliche Antwortzeit in einer EUCALYPTUS-basierten heterogenen Compute Cloud. Allerdings vernachlässigen sie dabei die Datenbankschicht und Netzwerklimitationen. Außerdem kann ihr System nur hochskaliert werden. Es werden jedoch keine Instanzen abgegeben, um die Ressourcenanzahl des Gesamtsystems zu reduzieren [84].

Obwohl solche Ansätze besser funktionieren als die klassische statische Ressourcenzuteilung, können bei hoher Anfragedichte einige Anfragen nicht beantwortet werden. Das führt zu einer häufigen Wiederanforderung durch den Client und belastet das Netzwerk durch diese zusätzlichen Versuche unnötig. So wird eine effiziente Adaption unmöglich, wenn die vorhandene Ressourcenkapazität unter eine akzeptable Grenze sinkt.

Durch den Einsatz von Virtualisierung können mehr Kapazitäten zur Dienstерbringung eingesetzt werden. Bei den dynamischen Managementlösungen wird beim Service-Anbieter die Anzahl der nötigen IT-Ressourcen ermittelt. Auf Basis von empirischen Vorhersagen wird so der zukünftige Verlauf einer Dienstnutzung für ein definiertes Zeitintervall bestimmt. Ein periodischer Ressourcenbedarf und Muster, die eine gewisse Art der Nutzung aufweise, können auf diese Weise erkannt werden.

Chen et al. [34] nutzen solch aufgezeichnete Profile, um kommendes Lastverhalten zu prognostizieren. Zur Abschätzung darauf folgender Werte müssen sie sich auf die fehlerfreie Korrektheit sämtlicher Aufzeichnungen verlassen. Die Autoren konzentrieren sich mit der Bereinigung abweichender Werte allerdings nur auf einen kleinen Teil der automatischen Lastadaption.

Die variierende Anzahl von Daten und Instanzen führt zu verschiedenen Lastmustern und -modellen. Paton et al. [135] definieren beispielsweise verschiedene Lastarten unter Betrachtung von Lasteigenschaften, Service-Level-Agreements und dem Wettbewerb um geteilte Ressourcen. In einem Arbeitszyklus werden mögliche Systemzustände auf eine gemeinsame Bewertungsskala abgebildet, die grundsätzlich Antwortzeiten, Quality-of-Service-Ziele oder Gewinne bei den Antworten verkörpert. Allerdings wird von einem begrenzten Ressourcen-Pool ausgegangen, wohingegen in heutigen Cloud-Umgebungen eine theoretisch unbegrenzte Ressourcenkapazität angenommen wird.

Das Signature-driven Load Management System ‘SigLM’ von Gong et al. [70] realisiert Quality-of-Service in geteilten Cloud-basierten Infrastrukturen. Die Verwendung der dynamischen, feingranularen Signaturen ermöglicht eine exakte Ressourcenzählung und -zuteilung.

Brandic [22] ist ebenfalls der Meinung, dass Anwendungen nach einem zuvor festgelegten Ablaufplan abgearbeitet werden können. Da die Nutzerinteraktion mit dem Dienst aufwändig und oft fehleranfällig ist, wird diese hier minimiert. Dabei liegt jedoch der Fokus eher auf der Reduzierung von Fehlerquellen anstatt auf Performanz- und Effizienz-Gewinnen. Chaisiri et al. [31] schlagen dafür einen ‘Optimal Cloud Resource Provisioning’-Algorithmus vor. Durch ein stochastisches Programmiermodell wird ein Reservierungsplan generiert. So sind die Gesamt-Provisionierungskosten reduzierbar.

Ein Beispiel zur Selbstorganisation definieren Gmach et al. [67] mit dem ‘AutoGlobe’-System. Die Umgebung ist in der Lage, Überlast zu erkennen und wie folgt zu reagieren. Bei Diensten mit periodischem Lastverhalten wird eine Kurzzeitprognose durchgeführt. Hingegen werden bei nicht-zyklischem Verhalten und unerwarteten Ereignissen Hinweise durch den Administrator abge-

geben, die sich z.B. auf die Ressourcenauslastung berufen. Dabei wird allerdings lediglich ein kleiner Teil der IT-Businesswelt bedient, nämlich Enterprise-Resource-Planning-Systeme. Sicherlich ist es entscheidender, auch ein größeres Feld an Anwendungen zu untersuchen.

All diese Prozesse, die Vorhersagen mit einbeziehen, führen jedoch zu Problemen. Für die Durchführung von Langzeitprognosen benötigt man viele historische Daten, um durch diverse Analyseverfahren – z.B. per Fourier Analyse [98] – periodisches Verhalten oder verschiedenartige Lastmuster zu erkennen. Dieser Ansatz funktioniert also nur, wenn dieses Verhalten auch gegeben ist und das Lastverhalten damit kategorisierbar ist. Zudem ist eine Auswertung von zuvor gesammelten Daten langwierig und aufwändig, beziehungsweise sind die Nutzungsdaten oft überhaupt nicht verfügbar.

Im Fall von Kurzzeitprognosen auf Basis vordefinierter Regeln wählt man eine generische Methode, welche die Verwaltung der Ressourcen ermöglicht. Für gemäßigte Veränderungen im Lastverhalten funktioniert dieser Ansatz sehr gut. Allerdings versagt er für groß Veränderungen in kleinen Zeitintervallen, da die dynamische Reorganisation viel Zeit benötigt [105].

Heutige Systeme bieten die Möglichkeit der Überwachung durch die Sammlung von Performanzdaten, um die Leistung des Systems im Allgemeinen zu bestimmen. Alam et al. [4] klassifizieren dafür zwei Gruppen von 'Performance Countern'. Eine Klasse zeigt an, dass in baldiger Zukunft Probleme auftreten werden. Die andere lässt auf bereits bestehende Überlast schließen.

Nahende Probleme ergeben sich durch eine hohe Anfragemenge in einer Warteschlange, lange Bearbeitungszeiten, maximale CPU Auslastung und geringe Abarbeitung pro Sekunde. Wenn andererseits viele Applikationen oder Prozesse wieder neu gestartet werden müssen und eine hohe Fehleranzahl auftritt, besteht Grund zur Annahme, dass dem System nicht genügend Ressourcen zur effizienten Verarbeitung zur Verfügung stehen. Allerdings muss die Auswertung der Performanzdaten für eine angemessene Ressourcenzuschaltung jeweils manuell gelöst werden. Um diesen hohen Aufwand zu umgehen, muss die Systemregulierung automatisiert werden.

Dafür stellen Miers et al. [125] eine Platform-as-a-Service-Lösung vor, die Dienstanbietern erweiterte Features bietet, ohne dass sie tiefer liegende technische Details der Cloud kennen müssen. Die grundlegende Idee des sogenannten 'Trade Wind' ist, den Dienstanbietern Möglichkeiten zu bieten, high-level Parameter für ihre Bedürfnisse zu konfigurieren. Die Dienste sind auf tiefer Ebene optimierbar. Außerdem kann das Verhalten der Cloud bezüglich Ressourcennutzung sowie Kosten definiert werden.

Zur weiteren Verbesserung der Skalierbarkeit und Flexibilität wird von Yazier et al. [186] eine multikriterielle Entscheidungsanalyse durchgeführt. Sie vernachlässigen allerdings den Status der einzelnen virtuellen Maschinen. Diese Zustandsüberwachung ist in Cloud-Systemen generell nötig.

Heutzutage unterstützen Cloud-Plattformen die Konfiguration der Ressourcenanzahl. Sie bieten zugleich eine Schnittstelle zur Einstellung über verschiedene

Portale. Beispielsweise schlagen De Assunção et al. [17] vor, Nutzeranfragen zunächst im heimischen Organisationscluster zu bearbeiten und bei Überlastsituationen die Anfragen in eine zugeschaltete externe Cloud zu übertragen, in der zusätzliche virtuelle Maschinen bereitgestellt werden. Dabei untersuchen sie zwar den Trade-off zwischen Verbesserungen der Performanz und steigenden Kosten, blenden allerdings Nutzungsgebühren aus.
Ein solcher Ansatz wird im Protokoll 'SNAP' verfolgt [39]. Es dient der Aushandlung von Service-Level-Agreements und anschließender Koordinierung von Ressourcen in verteilten Systemen auf Basis von virtuellen Maschinen.
In [63] erklären Galstyan et al. einen Algorithmus zur verteilten dynamischen Ressourceneinteilung ohne zentralen Kontrollmechanismus und ohne Einbezug von globalen Informationen durch lernende Komponenten. Dennoch befassen sie sich nicht mit der Implementierung eines Kommunikationsprotokolls, was nötig wäre, um die eigentlichen Agenten zu realisieren.

3.3 Monitoring-Lösungen im Cloud Computing

Systeme, die nach dem Cloud-Paradigma entworfen sind, müssen große Informationsmengen verwalten. Für die Bewältigung dieser Daten sind enorme Kapazitäten erforderlich. Um die nötige Ressourcenmenge konfigurieren und vor allem überwachen zu können, müssen entsprechende Dienste eingebunden werden. Neben den beschriebenen wissenschaftlichen Herangehensweisen gibt es auch kommerzielle Ansätze, die im Folgenden exemplarisch an den Lösungen von Amazon und Microsoft Azure erläutert werden.

Amazon CloudWatch

Amazon bietet zur Systemkontrolle durch den Nutzer das Produkt *CloudWatch*. Es stellt unterschiedliche Möglichkeiten bereit, die eigenen Amazon-Web-Service-Ressourcen und Anwendungen zu überwachen. Zu diesem Zweck sammelt das System Informationen über die Auslastung und Performanz des beobachteten Prozesses. Dafür werden Metriken verwendet, die als Standard verfügbar sind oder durch den Benutzer definiert werden.
Mittels *CloudWatch* sind Instanzen der *Elastic Compute Cloud* überwachbar. Auch der *Elastic Block Store*, der *Relational Database Service* sowie das *Elastic Load Balancing* und sogar benutzerdefinierte Anwendungsmetriken können von Kunden beobachtet werden [8].
Eine Metrik repräsentiert eine zeitlich geordnete Menge von Datenpunkten, die in der *CloudWatch* verankert wird. Die einzelnen Metriken werden durch den Namen, den Namensraum und durch eine oder mehrere Dimensionen eindeutig gekennzeichnet. Dabei wird eine Dimension durch ein Name/Wert-Paar präzise beschrieben. Die einzelnen Metriken besitzen spezifische Charakteristika.

Die Dimensionen stellen die Kategorien für diese Merkmale dar. Weiterhin ist jeder Datenpunkt bestimmt durch einen Zeitstempel und eine Messeinheit. Zu den Datenpunkten können Statistiken als geordnete Zeitreihendaten abgefragt werden. Sie sind anhand von identifizierten Namensräumen, dem Metriknamen, der Dimension oder der Einheit zu unterscheiden.
Die Metrik ist also die zu überwachende Variable. Die Datenpunkte sind die vom Messzeitpunkt abhängigen Variablenwerte. Die CPU-Auslastung einer einzelnen *EC2*-Instanz oder die Latenz eines *Elastic Load Balancers* sind Beispiele für Metriken [13].
Um einen reibungslosen Betriebsablauf zu gewährleisten, werden die definierten Metriken nachverfolgt und Prozesseinblicke gewährt. Durch die Analyse der Daten über die AWS-Management-Konsole kann man sich einen Gesamtüberblick verschaffen. Die Entscheidung, ob neue Instanzen hochzufahren oder unbenutzte freizugeben sind, wird erleichtert. Basierend auf dem überwachten Zustand der Cloud-Umgebung und durch die Definition geeigneter Metriken, kann der Nutzer verschiedene Automatismen anstoßen.
Anwendungen, die *CloudWatch* unterstützen, senden ihre Metriken direkt an den Service. Welche Metriken dabei kontrolliert und übermittelt werden, entscheidet der Nutzer [8]. Im Falle der *Elastic Compute Cloud* ist es beispielsweise sinnvoll, die CPU-Auslastung, die Lese- und Schreiboperatoren der Festplatten und die Netzwerktransferrate zu analysieren.
Insgesamt müssen so zwar keine eigenen Infrastrukturen zur Überwachung eingerichtet werden, doch liegt es am Nutzer, eine effiziente Logik zur Skalierung seines Systems umzusetzen und zu prüfen.

AzureWatch

Der Service *AzureWatch* ist von Paraleap Technologies entwickelt und wird von Microsoft unterstützt. Hierüber bietet Microsoft eine verbesserte Elastizitätsunterstützung. Die dynamische Anpassung der Compute-Instanzen richtet sich nach dem zeitlichen Bedarf der betrachteten Azure-Anwendung. Durch vom Nutzer definierte Regeln entscheidet das System automatisch über die Zuschaltung weiterer Ressourcen oder die Abschaltung ungenutzter Instanzen. Zur Definition und Konfiguration der Regeln bietet der Monitoring-Dienst von Azure ein Windows-basiertes Bedienfeld [133].
AzureWatch misst die Performanzmetriken, aggregiert sie über verschiedene Applikationen und analysiert diese Werte. Wenn die Analyse eine Übereinstimmung zwischen den Metriken und den vom Nutzer oder über Basisfunktionalitäten vordefinierten Regeln ergibt, wird eine Skalierungsaktion aufgerufen.
Dabei kann das Erfassen der Metriken entweder über die *AzureWatch*-Server erfolgen oder über ein kleines Agentenprogramm, das off-premise auf den Servern der Nutzer läuft. Die Datenspeicherung, Vereinigung und Regelevaluation erfolgt immer auf den *AzureWatch*-Servern in der Cloud. Über diese Rechner

kann der Nutzer nach der Analyse auch die historischen Daten per Bericht und Diagramm abrufen. So erhält er durch die Anzeige der durchschnittlichen Metriken auf verschiedenen Azure-Instanzen einen groben Überblick über die Performanz der Applikationen.

Das Hinzunehmen oder Abgeben von Azure-Instanzen basiert auf der Echtzeit-Bedarfsanalyse der aktuellsten Performanzwerte, dem vergangenen aggregierten Bedarf, dem Anteil des Anstiegs oder Abfalls im Bedarf, der Tageszeit, der Warteschlangenlänge und/oder den verschiedenen Zuständen. Wenn die definierten Regeln greifen, schickt *AzureWatch* einen einzelnen Alarm oder intervallartige Meldungen an den Nutzer. So kann beispielsweise ein Alarm ausgelöst werden, wenn eine Instanz nicht mehr erreichbar ist oder wenn trotz maximaler Ressourcenmenge noch eine Überlast im System besteht.

Die eingebauten Begrenzungen und Drossel-Kontrollmechanismen bewahren die Instanzen einerseits vor dem Überschreiten vordefinierter Bereiche und andererseits vor dem zu häufigen Skalieren. Dabei wird sämtliche Kommunikation mit den Azure-Servern über den Secure Sockets Layer (SSL) verschlüsselt, dem hybriden Verschlüsselungsprotokoll zur sicheren Datenübertragung im Internet [144]. Kontoinformationen, Performanzmetriken und Zertifikate werden in der Azure-Cloud hinter einer IP-Filter-Firewall gespeichert [134].

Wie Abbildung 3.1 zeigt, gliedert sich die grundlegende *AzureWatch*-Architektur in drei Teile. Die eigentliche Applikation stellt in diesem Zusammenhang eine Tabelle mit den Werten der Performance Counter sowie Warteschlangeninformationen via *Azure Storage Queues*. Die Instanz-Zustände wie 'Ready', 'Busy' oder 'Unresponsive' können per *Diagnostics Management*-API

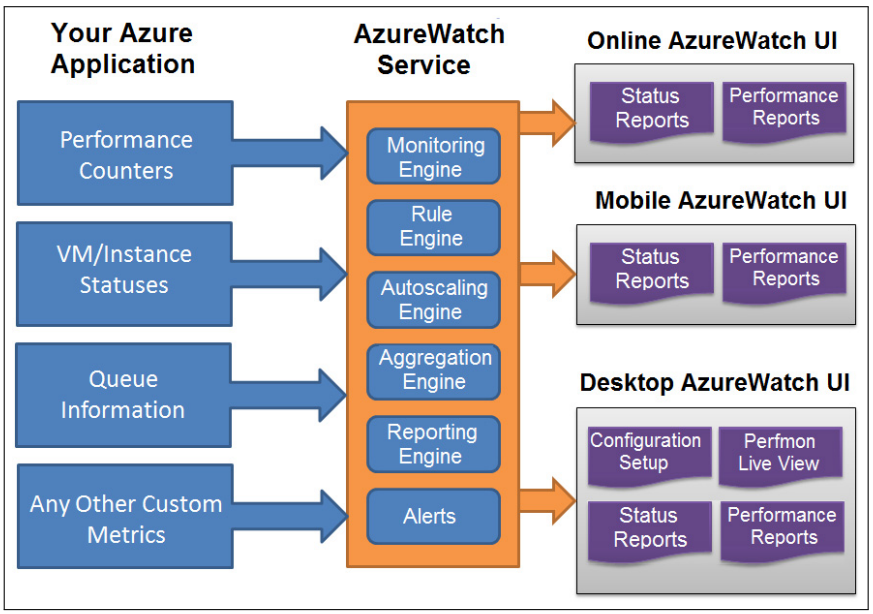


Abbildung 3.1: Die high-level Architektur von *AzureWatch* [132]

abgefragt werden. Außerdem kann jede beliebige Kundenmetrik über XML-Feeds zur Anwendungsüberwachung eingebunden werden.
Der *AzureWatch*-Service verarbeitet die gelieferten Informationen mithilfe von Regel-, Überwachungs-, Aggregations-, Berichterstattungs- und Warn-Einheiten und leitet sie an die Nutzerschnittstellen weiter. Diese Schnittstellen gliedern sich in drei Gruppen: Die Online- und die mobile Schnittstelle visualisieren ausschließlich Status- und Leistungsberichte. Der dritte Teil stellt die Desktop-Schnittstelle dar. Sie bietet zusätzlich ein Konfigurationssetup für spezielle Einstellungen. Eine *Perfmon Live*-Ansicht überprüft den Computer im laufenden Betrieb. Anschließend wird eine weitreichende Systemdiagnose erstellt. Per *Perfmon* stellen Windows Vista und Windows 7 also ein sehr mächtiges Reporting-Tool bereit, das bei der Ursachenforschung bei Leistungsproblemen hilfreich sein kann [132].

3.4 Zusammenfassung

Um alle Anfragen, die an große verteilte Systeme gestellt werden, bearbeiten zu können, wird eine enorme Menge an Ressourcen benötigt. Mit ausreichend umfangreicher Kapazität ist es theoretisch möglich, der umfassenden Belastung eines Systems zu begegnen. Die unüberschaubare Zahl an Instanzen führt jedoch dazu, dass intelligente Automatismen benötigt werden, um die laufenden Maschinen zu verwalten.
Ursprünglich entstand dafür das statische IT-Ressourcenmanagement. Bei solchen Lösungen wird vor dem Betrieb per Service-Level-Agreement festgelegt, welches Kapazitätsvolumen der betreibenden Umgebung zur Verfügung steht. Das Kapitel stellte einige Ansätze vor. Diese Art des Ressourcenmanagements kann durch eine dynamische Adaption verbessert werden. Die dazu beschriebenen Lösungen analysieren Häufigkeitsmuster vergangener Anfragenverläufe. Durch vorausschauende Prognosen ist so in der Zukunft besser auf eventuelle Engpässe zu reagieren. Um eine Möglichkeit zur Überwachung zu schaffen, kommen bei einigen Systemen Performanzwerte zum Einsatz.
Dabei tritt die Bedeutung der Virtualisierungs- und Cloud-Technologie in den Vordergrund. Theoretisch wird eine unendliche Instanzenmenge nutzbar, was ihre Kontrolle zusätzlich verkompliziert. Mit steigendem Interesse an Cloud-Systemen muss auch den technischen Aspekten der Lastvarianzen und ihrer Balancierung mehr Beachtung geschenkt werden als bisher.
Neben den wissenschaftlichen Vorschlägen wurden in diesem Kapitel auch kommerzielle Lösungen bekannter IT-Anbieter beleuchtet. Sie sollen zur Überwachung und Verwaltung der Ressourcen in verteilten Systemen beitragen. Allerdings zeigt sich, dass weder die wissenschaftlichen noch die unternehmerischen Herangehensweisen den heutigen komplexen und schnelllebigen Cloud-Umgebungen gerecht werden.

Die optimale Verteilung auftretender Last bezogen auf kurze Antwortzeiten und hohe Leistungsfähigkeit ist bis jetzt noch nicht befriedigend gelöst. Zur Gewährleistung einer effizienten Anfragenbearbeitung und Anwendungsskalierung müssen die verfügbaren Angebote stets vom Nutzer über Regeln und Metriken konfiguriert und zusätzlich kontrolliert werden.

Die durchgeführte Analyse bestehender Lösungen macht den enormen Bedarf nach intelligenteren Aufgabenverarbeitungen und automatisierten Allokationsmechanismen zur proaktiven Zuschaltung und Deaktivierung von Ressourcen sehr deutlich. Ein solches System wird im Rahmen dieser Arbeit entwickelt.

4 Konzeption eines Systems für effizientes Ressourcenmanagement

Cloud Computing hat sich inzwischen als wichtiger Bestandteil der IT-Welt etabliert. In einem Cloud-Umfeld kann der Nutzer flexibel über eine theoretisch unendliche Menge an Ressourcen verfügen. So können Software-as-a-Service-Anbieter ihren Benutzern lauffähige Anwendungen auf eigenen oder angemieteten Servern aus der Platform-as-a-Service- oder Infrastructure-as-a-Service-Ebene bereitstellen. Um dabei wirtschaftlich profitabel zu sein und gleichzeitig einen guten Dienst zu bieten, müssen sie zeitnah auf Lastanstiege durch die Zuschaltung von zusätzlicher Kapazität reagieren können.

Dies stellt eine erhebliche Herausforderung dar, weil die Bereitstellung zusätzlicher virtueller Maschinen auf physikalischen Ressourcen zu nicht vernachlässigbaren Verzögerungen führt [105]. Da außerdem der Umfang an Cloud-Diensten massiv ansteigen wird, muss in Zukunft verstärkt auf Ressourceneffizienz geachtet werden [27]. Wie das vorangegangene Kapitel zeigt, bieten Anbieter der Cloud-Technologie nur begrenzte Möglichkeiten zum dynamischen Management der bereitgestellten Betriebsmittel.

Hinzu kommt, dass verteilte Cloud-basierte Services typischerweise vorab nicht wissen, wann und wie viele Nutzer ihren Dienst in Anspruch nehmen. Folglich können sie nur schwer die zu erwartende Arbeitslast auf ihren Ressourcen prognostizieren. Im Folgenden wird synonym für Nutzer auch der Begriff Client verwendet. Obwohl die Lastprognose Probleme aufwirft, verhält sich die tatsächliche Inanspruchnahme der Dienste durch die Clients meist nach bestimmbaren Lastmustern. Die aus gesammelten Datensätzen extrahierten Informationen, können der Ressourcenverwaltung dienen.

Für ein solches Management der Cloud-Ressourcen gibt es jedoch noch keine standardisierte Technik. Deshalb ist eine manuelle Konfiguration für die Systemskalierung durch den Anwender unabdingbar. Die Entscheidung über das Starten oder Stoppen der benötigten Instanzen kann lediglich über vordefinierte Regeln realisiert werden. So müssen bestehende Angebote zum Ressourcenmanagement durch den Nutzer massiv erweitert werden, um verhandelte Service-Level-Agreements und Quality-of-Service zu kontrollieren und um Verwaltungsmechanismen zu schaffen [28].

Das zeigt die Bedeutung der Selbstregulierung ohne umfassende Systemkenntnisse für ein effizientes Ressourcenmanagement in Cloud-Umgebungen. Beim

Management der beteiligten Komponenten ergeben sich allerdings einige Herausforderungen. Das unterschiedliche Verhalten der Cloud-Systeme ist dabei zu bedachen. Außerdem ist die ökologische Verantwortung unserer Generation zu berücksichtigen. Diese angeführten Aspekte werden im Folgenden analysiert und ein Lösungsansatz in den Grundzügen vorgestellt.

4.1 Herausforderungen beim effizienten Ressourcenmanagement

Ein kritischer Punkt des Ressourcenmanagements ist die Kompromissfindung zwischen minimaler Anzahl physikalischer Kapazität aus Kostengründen und gleichzeitiger hochverfügbarer Diensterbringung mit Einhaltung zuvor vereinbarter SLAs. Auch stellt sich die Frage, wann Instanzen zur Kosteneinsparung wieder abzugeben sind. Hier gelten folgende Herausforderungen:

- **Ermittlung der benötigten Kapazitäten:** Es muss effizient und automatisiert festgestellt werden, ob und wie viele Ressourcen anzufordern sind. Zur optimalen Ressourcenauslastung müsste dafür der Bedarf bereits im Voraus bekannt sein. Vorab sind darüber aber standardmäßig keine Informationen verfügbar. Zudem ist die Generierung von Historien beim Diensterbringer häufig sehr aufwändig.

- **Extraktion der Ressourcenkosten:** Bei hohen Kosten für die Cloud-Kapazitäten ist eine eher zurückhaltende Strategie bezüglich der Zuschaltung sinnvoll. Bei niedrigen Kosten kann es durchaus praktikabel sein, Ressourcen im Übermaß bereitzustellen, um auch unerwartete und spontane Lastspitzen behandeln zu können. Die Ressourcenkosten variieren je nach Anbieter und können sich auch innerhalb eines Angebots zeitlich ändern. Somit ist es schwer, den Trade-off allgemein zu lösen. Es muss jeweils eine individuelle Lösung gefunden werden.

- **Wissen über das Bezahlmodell:** Da sich etliche Cloud-Anbieter etabliert haben, besteht unter ihnen ein enormer Konkurrenzdruck. Die Anbieter versuchen möglichst viele Kunden durch optimale Kosten-Nutzen-Relation an sich zu binden. Dafür bieten sie verschiedene Bezahlmodelle an, wobei Ressourcen häufig im ein-Stunden-Raster abgerechnet werden. Das bedeutet, auch wenn eine Instanz nur zwei Minuten belegt wird, ist die angefangene Stunde voll zu bezahlen. Dann sollte sich der Nutzer die Kapazität gleich die ganze Stunde vorhalten lassen, um Rückstau abzuarbeiten oder um Reserven für unerwartete Last aufzubauen.
  Die Definition des Zeitpunkts, wann Ressourcen abzustoßen wieder sind, muss äußerst genau über verlässliche Algorithmen bestimmt werden.

- **Provisionierungszeit der Ressourcen:** Wenn die Zeit zum Starten einer Instanz mehrere Minuten dauert, kann die Bereitstellung der Ressource bei Lastspitzen eventuell zu spät erfolgen. Insofern ist eher prognostizierend beziehungsweise „übersteuernd“ zu agieren.
  Die Provisionierungszeit, also die Dauer, bis die eigentlich benötigte Ressource durch den Anbieter bereitgestellt ist, variiert erheblich und ist nicht als Standard festgelegt.

- **Erfüllung von vereinbarten Service-Level-Agreements:** Bei der Diensterbringung werden SLAs oft zu starr, das heißt ohne dynamische Kontrollmöglichkeit, gehandhabt. Dadurch ist der Anbieter in seiner Handlungsfreiheit eingeschränkt. So agiert er oft uneffektiv oder muss entstehende Kosten an den Nutzern weitergeben.
  Der Parameter des Quality-of-Service misst dabei die Dienstqualität. Zum Beispiel kann mit Transaktionen pro Sekunde, allgemeinem Durchsatz, Verfügbarkeit oder Latenzzeiten eingeschätzt werden, wie gut der Dienst die Anforderungen des Clients erfüllt.
  Verhandelte SLAs sollen also im besten Fall dynamisch während der eigentlichen Diensterbringung angepasst werden. Zudem müssen eventuell verletzte SLAs kompensierbar sein. Bei Neuverhandlungen der Nutzungsbedingungen zwischen Client und Anbieter soll der entstehende Overhead möglichst minimal sein. Andernfalls ist die Zusatzbelastung durch ständige SLA-Änderungen nicht tragbar.

- **Bedarfsoptimierung von physikalischen IT-Ressourcen:** Aufgrund der rasant wachsenden Dienstlandschaft, der wirtschaftlichen Randbedingungen und im Sinne von umweltverträglicher Nachhaltigkeit müssen Cloud-Dienste mit *minimaler* Kapazität erbracht werden.
  Außerdem sind Veränderungen im physikalischen Ressourcenbestand mit längeren Ladezeiten und Zusatzaufwand behaftet. Deshalb ist es für die effiziente Diensterbringung ein zweites wichtiges Kriterium, mit möglichst *konstanter* Ressourcenanzahl auszukommen.

- **Denial-of-Service-Fehlalarme durch große Lastspitzen:** In Public Clouds müssen zwecks Sicherheit und Stabilität der Diensterbringung sogenannte Intrusion-Detection-Systeme vor den Lastbalancierer geschaltet werden. Diese sind herkömmlich IP-Adressen-basiert.
  Geht nun eine intensive Cloud-Dienstnutzung von einer bestimmten IP-Adresse aus, wird dies häufig fälschlicherweise als Denial-of-Service-Attacke erkannt und die zugehörige IP-Adresse unnötig blockiert. Das führt für Anwender zum Dienstausfall [87]. Bei Algorithmen zur Lastverteilung ist dieser Aspekt zu berücksichtigen.

4.2 Identifikation von Lastmustern

Um sämtliche erwartete Arbeitslast abzuarbeiten, ist eine möglichst genaue Vorhersage nötig. So kann man schon vorab Ressourcenanforderungen behandeln, beziehungsweise wenigstens in angemessenem Zeitraum auf bereits steigendes Arbeitslastaufkommen reagieren. Zur Angabe des zukünftigen Bedarfs müssen deshalb Lastmuster aus zuvor gesammelten Datensätzen oder externen Quellen extrahiert werden.

Bei Diensten, die per Nutzerinteraktion gesteuert werden, ist es schwierig, das künftige Benutzerverhalten exakt anzugeben. Durch abschätzende Analysen können trotzdem häufig richtige Voraussagen realisiert werden. Die verschiedenen Faktoren und beeinflussenden Parameter gilt es zu bestimmen.

Wie in [155] beschrieben, tauchen im täglichen Leben regelmäßig Laststrukturen mit verschiedenen Wiedererkennungswerten auf. Ein Ausschnitt der Lastmuster ist in Abbildung 4.1 dargestellt. Hier sind sogenannte *On-Off-Situationen* zu nennen, in denen entweder eine relativ konstante, hohe Last zu verzeichnen ist oder der Dienst gar nicht genutzt wird und die Ressourcen im Idle-Modus laufen, was Abbildung 4.1a zeigt.

Gerade bei gut anlaufenden Startup-Unternehmen lässt sich ein stetiges, im Idealfall sogar *exponentielles Wachstum* beobachten, wenn die Resonanz auf das Unternehmensangebot steigt – siehe Abbildung 4.1b.

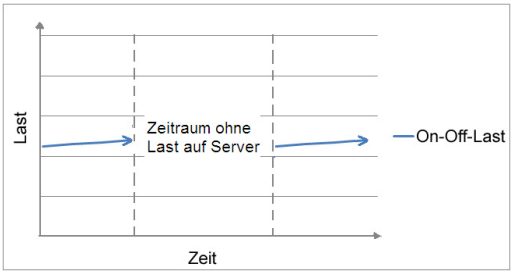
Weiterhin finden sich Muster, in denen die durchschnittliche Belastung meist konstant ist, mit *hervorstechenden Lastspitzen*. Basierend auf Echtzeitdaten ist zum Beispiel anzunehmen, dass Lastspitzen auf Servern, die Steuerverwaltungsprogramme bereitstellen, im Mai jeden Jahres zu verzeichnen sind, da zu dieser Zeit die jährliche Steuererklärung fällig ist – siehe Abbildung 4.1c.

Auf Servern, die Foto-Sharing-Plattformen bedienen, sind laut der Alexa Web Information Company [5] Lastspitzen eher um März/April und im September zu erkennen, wie Abbildung 4.1d verdeutlicht. Das spiegelt den Umstand wieder, dass die meisten Menschen Fotos nach dem Urlaub hochladen, um ihre Ferienerinnerungen mit ihren Freunden zu teilen.

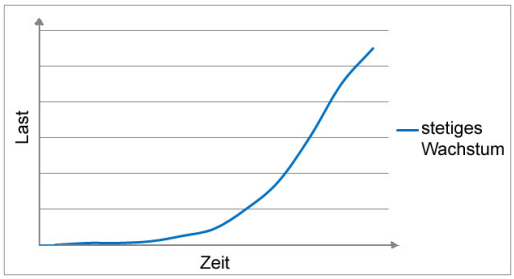
Ferner lassen sich auch Prozesse identifizieren, die einem täglichen Muster folgen, wie in Anwendungsfällen der Elektromobilität. So sind in Hauptverkehrszeiten enorme Datenmengen zu erwarten, die an den Management-Servern auflaufen. Da in diesen Zeiten viele Menschen zur Arbeit oder von ihr nach Hause fahren, produzieren ihre Autos unzählige Telemetriedaten über den technischen Autozustand. Diese werden automatisch an zentrale Server übermittelt, dort verarbeitet und abgelegt.

Durch solche Regelmäßigkeiten im Lastverhalten lassen sich spezifische Historien erstellen und proaktive Aktionen durchführen. Mithilfe der Prognose kann die Anzahl der vermutlich nötigen Ressourcen vor der eigentlichen Verwendung gestartet werden. Sie sind dann bei einer Lastzunahme bereits funktionstüchtig und einsatzbereit. Insgesamt beschreiben diese Beispiele also eher wiederkehrende *zyklische Muster*. Das erleichtert eine Generierung von Historien.

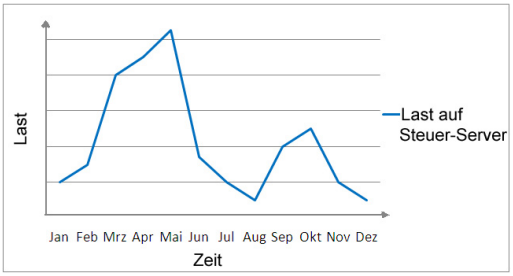
Nichtsdestotrotz gibt es Fälle, in denen es enorm schwierig ist, exakte und offensichtliche Muster zu identifizieren, zum Beispiel für News-Seiten auf Internet-Servern. Abbildung 4.1e verdeutlicht, dass diese Daten einem unerwarteten Schema folgen, da sich interessante Geschehnisse in der Welt nach reinem Zufallsprinzip ereignen. Dennoch machte der amerikanische Professor Leskovec in seiner Keynote auf der Extended Semantic Web Konferenz 2011 die Aussage, dass Lastspitzen auf Blog-Seiten typischerweise 2,5 Stunden nach Lastausschlägen auf News-Servern zu verzeichnen sind [96].
So lassen sich relative Modelle erstellen, die Varianzen und Steigungen berücksichtigen und die kurzzeitige Vorhersage der künftigen Last nach relevanten



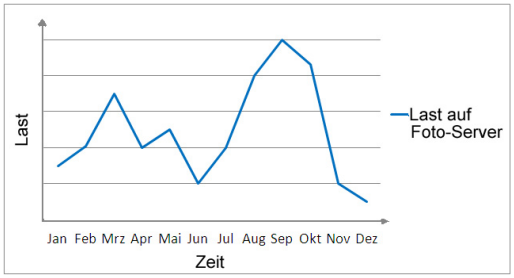
(a) Last bei Batch-Job-Server



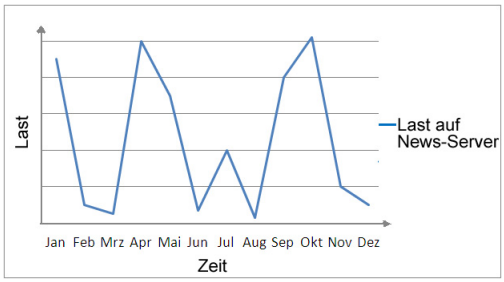
(b) Last bei exponentiellem Wachstum



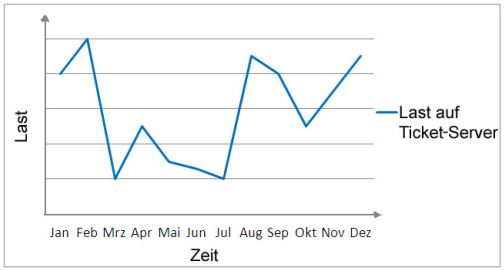
(c) Last auf Steuer-Server



(d) Last auf Foto-Sharing-Server



(e) Last auf News-Server



(f) Last auf Ticket-Server

Abbildung 4.1: Denkbare Lastmuster in Cloud-Anwendungsszenarien [154]

Begebenheiten ermöglichen. Der weitere Verlauf auf den Servern verhält sich auf diese Weise, wenn auch mit unterschiedlicher Intensität.
Daneben existieren *Ereignisse, die genau planbar sind*, obwohl sie keinem zyklischen Muster folgen, wie das Lastverhalten auf Ticket-Servern in Abbildung 4.1f. Events wie Konzerte, bedeutende Fußballspiele oder andere große und seltene Ereignisse sind meist bereits vor dem eigentlichen Geschehen bekannt. Deshalb können Informationen über beispielsweise Ort oder Zeitpunkt des Ereignisses zur präzisen Lastvorhersage hinzugezogen werden. Damit ergibt sich die weitere Chance, Korrelationen zwischen Mustern zu erkennen und sie zur Lastkurvenbestimmung zu nutzen.
Die Prognose des künftig zu erwartenden Lastprofils kann auch durch den Client selbst erfolgen. Sie wird dann an den zugehörigen Server kommuniziert und verspricht auf diese Weise Entlastung.
Entsprechend dieser Erkenntnisse können unter Berücksichtigung der identifizierten Lastmuster Ressourcen rechtzeitig zugeschaltet werden.

4.3 Ökonomische und ökologische Aspekte des Cloud-Ressourcenmanagements

Cloud Computing verspricht schnelle und beliebige Ressourcenzuschaltung für eine breite Masse an Nutzern. Dies ist durchaus wünschenswert, doch erwächst damit auch eine nicht zu vernachlässigende ökologische Verantwortung. Kurzfristiges, unbedachtes Hinzuschalten einer mangelhaft abgeschätzten Ressourcenanzahl verursacht nicht nur Kosten für das Unternehmen, sondern führt ebenso zu höheren Schadstoffemissionen und Energieverschwendung. Dabei wird oft verdrängt, dass ein Rechner im Leerlauf etwa zwei Drittel der Energie verbraucht, die ein ausgelasteter Server benötigt [15]. So zeigen Rechenzentren von Google mit ihren unaufhörlich laufenden Servern den gleichen Energieverbrauch wie eine Stadt der Größe von Los Angeles [28].
Laut Berechnungen aus dem Jahr 2008 wird angenommen, dass Rechenzentren 0,5 % des weltweiten Gesamtenergieverbrauchs ausmachen und dieser Wert bis 2020 enorm ansteigen wird [58]. Im Jahr 2011 belief sich der Stromverbrauch von deutschen Rechenzentren laut dem Borderstep Institut auf 9,7 Terawattstunden, was einem Anteil von 1,8 % des Gesamtstromverbrauchs in Deutschland entspricht [80]. Dies führt offensichtlich zu enormen Kosten.
Nach physikalischen Grundlagen errechnen sich die Gesamtkosten für den Stromverbrauch aus der elektrischen Arbeit multipliziert mit dem Preis für die Ressourcen [149]. Damit ergibt sich die folgende Formel zur Berechnung der jährlichen Stromkosten K in Euro einer Firma für a Computerarbeitsplätze im Dauerbetrieb mit bestimmter elektrischer Anschlussleistung e in Kilowatt und festgelegtem Preis k pro Kilowattstunde.

$$K = a \cdot e \cdot 24 \ \frac{\text{h}}{\text{Tag}} \cdot 365 \ \frac{\text{Tage}}{\text{Jahr}} \cdot k \tag{4.1}$$

Ein voll funktionsfähiger Desktop PC benötigt zur Erbringung sämtlicher Leistung neben der CPU auch Grafikkarten, Software und Peripheriegeräte wie Drucker [89]. Aktuellen Schätzungen zufolge kann für Desktop PCs im Schnitt von einer elektrischen Anschlussleistung mit 400 Watt ausgegangen werden [38, 48, 76, 111]. Mit Formel (4.1) ergibt sich bei 1.000 Arbeitsplätzen zu einem Preis von 20 Cent pro Kilowattstunde für die Stromkosten aller PCs:

$$K_{PCs} = 1.000 \cdot 0,4 \text{ kW} \cdot 24 \ \frac{\text{h}}{\text{Tag}} \cdot 365 \ \frac{\text{Tage}}{\text{Jahr}} \cdot 0,2 \ \frac{€}{\text{kWh}} = 700.800 \ \frac{€}{\text{Jahr}}$$

Das Pay-per-Use-Modell der Cloud verspricht eine Reduktion solcher Kosten, solange die Kosten der Cloud-Rechnereinheiten geringer sind als die Anschaffungskosten eigener Hardware.
Da man grundlegend aber zusätzlich mit einrechnen muss, wie lange Lastspitzen andauern, macht nach Weinmann [179] der Cloud-Einsatz auch Sinn, falls die Rechnereinheiten der Cloud teurer sind als die eigene Beschaffung. Das ist vor allem dann der Fall, wenn die Lastspitzen wesentlich höher und kürzer ausfallen als die durchschnittlichen Bedarfskurven im Vergleich zum Kostenunterschied zwischen der on-demand und der dedizierten Variante. Der Grund dafür ist, dass die grundsätzliche Kapazität der dedizierten Lösung an den Lastspitzen ausgerichtet werden muss, wohingegen im on-demand Fall die Kosten proportional zum Durchschnittsverbrauch sind. Auf ein längeres Intervall gesehen, sind so in der Cloud die Gesamtkosten im Schnitt geringer.
Verschiedene Analysen verdeutlichen außerdem, dass zur Kostenbestimmung mehrere Faktoren eine Rolle spielen. Nach Hohenstein et al. [81] ist nicht nur entscheidend, wie sich der Verlauf der Arbeitslast verhält, sondern auch welche Cloud-Architektur zur Verarbeitung der Anfragen zugrunde liegt.
Mit simulierten Szenarien decken die Autoren enorme Kostenunterschiede für drei unterschiedliche Architekturen auf. Zum einen werden Systeme betrachtet, die auf das Rollenprinzip der Windows Azure Platform mit Web- oder Worker-Rollen aufbauen. Der zweite Fall realisiert die Lastverarbeitung mit Warteschlangen. Schließlich wird ein direkter Datenbankzugriff durch den Nutzer erlaubt. Die Kostenunterschiede liegen sogar im mehrstelligen Bereich. Da ihr Ausmaß aber von Frequenz und Größe der Daten abhängt, ist von Fall zu Fall zu entscheiden, welche Architektur die Anwendung am besten bedient.
In Cloud-Umgebungen spielt außerdem das Prinzip der Multimandantenfähigkeit eine große Rolle. Multimandantenfähige Applikationen unterstützen zeitgleich verschiedene Mandanten unterschiedlicher Organisationen. Dabei wer-

den die in der Cloud bereitgestellten Instanzen von den Nutzern der Mandanten geteilt. Beispielsweise können Firmen wie Siemens und BWM ein CRM-System des Anbieters Salesforce nutzen, das auf denselben Maschinen läuft.
Die beiden Unternehmen werden als Mandanten und ihre Mitarbeiter als Nutzer der Anwendung bezeichnet. Im Fall der gleichzeitigen Verwendung des Systems stellt sich dann die Frage, welche der beteiligten Parteien wie viel für die Nutzung der Anwendung zahlen muss.
Abbildung 4.2 repräsentiert den Geldfluss unter den beteiligten Komponenten. Die Infrastruktur- und Plattform-Betreiber stellen den Software-as-a-Service-Anbietern die nötige Kapazität zum Anwendungsbetrieb. Diese Applikation wird wiederum gegen Gebühr von verschiedenen Mandanten verwendet. Sie stellt ihren Nutzern die Funktionalitäten bereit.
Wie von Schwanengel et al. [153] erläutert, ist es folglich für SaaS-Anbieter zum profitablen Betrieb von enormer Bedeutung, dass die Einnahmen von den Nutzern der einzelnen Mandanten die ausgehenden Kosten für die Nutzung der PaaS- oder IaaS-Ressourcen decken.
Generell hängen die Betriebskosten von der Summe benötigter Kapazität ab. Für Cloud-Umgebungen verspricht die Aufgaben-Konsolidierung vieler virtueller Maschinen auf wenige physikalische Server eine Reduzierung der Gesamtressourcenmenge. Allerdings sind die heute gängigen Methoden nicht ausreichend, um tatsächlich ökologisch verträglich zu sein [189].
Die Ergebnisse zum Trade-off zwischen Energieverbrauch und Performanz in [52] zeigen das besonders drastisch. Für bestimmte Lasten lässt sich durch eine fünfprozentige Reduzierung der Prozessorleistung eine 20-prozentige Energieeinsparung realisieren. So führt die Verwendung schwächerer Computer zur geringen Erhöhung der Applikationsausführdauer von 57 Minuten auf 60 Minuten, dabei werden aber auch fast zwei Millionen US-Dollar eingespart. Dieses Wissen sollte beim Entwurf eines Managementsystems von Cloud-Ressourcen mit einfließen, um im Sinne umweltverträglicher Nachhaltigkeit zu handeln.

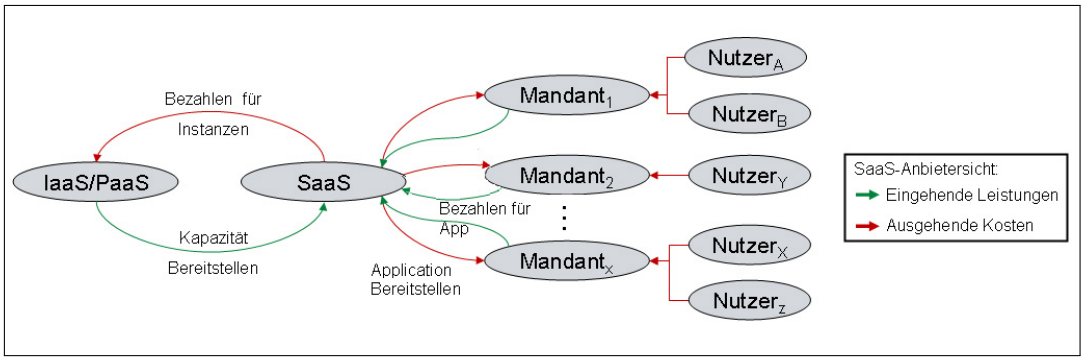


Abbildung 4.2: Geld- und Leistungsfluss zwischen IaaS-, PaaS- und SaaS-Anbietern und deren Nutzern

4.4 Lösungsansatz zum Ressourcenmanagement

Bei der Reaktion auf Lastveränderungen sind nach ihrer Erkennung generell verschiedene Strategien möglich. So ist bei erwarteten oder periodisch wiederkehrenden Lastspitzen meist eine frühzeitige Ressourcenanforderung empfehlenswert. Dahingegen ist ein völliger Verzicht auf jegliche Reaktion denkbar, wenn kein entsprechendes SLA existiert oder eine lange Provisionierungszeit eine zu späte Reaktion zur Folge hätte. Andererseits ist eine spontane Ressourcenanforderung bei schneller Provisionierungszeit sinnvoll – außer die Kosten für die Ressourcen sind zu hoch. Bei langsamer Ressourcenprovisionierung und/oder hohen Kosten ist eher abzuwarten, wie sich die Last entwickelt. Gegebenenfalls besteht die Chance, aufgestaute Last im Nachhinein nach einmaliger Ressourcenanforderung abzuarbeiten. Als letzte Möglichkeit können Ressourcen auch kontinuierlich angefordert werden.

Welche Strategie gewählt wird, ist also vor allem abhängig vom Lastvolumen, den Ressourcenkosten und der benötigten Zeit für die Bereitstellung. Um möglichst genau Ressourcen zu provisionieren, werden in dieser Arbeit verschiedene Verfahren entwickelt. Sie sind in den folgenden Kapiteln genauer erörtert.

Grundsätzlich muss die Ressourcenmanagement-Lösung die beschriebenen Aspekte zur Erkennung von Lastmustern sowie zu ökologischen und ökonomischen Herausforderungen vereinen und den Kapazitätenbedarf angemessen decken. Dabei sind neben der Dauer, ab wann und wie lange die Mittel benötigt werden, ebenso die entstehenden Kosten für die Instanzen zu berücksichtigen. Dies wird auch durch die Lokalität der gestarteten Ressourcen beeinflusst. Der Fokus liegt folglich auf der Identifikation einer geeigneten Reaktionsstrategie und der Entwicklung eines Verfahrens zum Ressourcenmanagement, das diese Anforderungen angemessen und automatisiert umsetzt.

In der Ressourcenverwaltung gibt es grundlegend zwei Wege, die nötige Instanzenmenge zu bestimmen. Zum einen kann man Ressourcen aktiv in die Bereitstellungsprozesse einbinden. Die einfachste Methode zur Umsetzung besteht darin, von den Nutzern eine Reservierung durchführen zu lassen. So geben sie selbst an, wann sie wie viele Instanzen belegen. Gerade in industriellen Szenarien ist das gut zu realisieren, da die anfragenden Clients meist selbst Rechner und keine Menschen sind. Maschinen können durchaus mit dem Zusatzaufwand der Reservierung belastet werden, was bei menschlichen Nutzern nicht ohne weiteres möglich ist. Außerdem sind die Clients durch Sicherheitsmechanismen auf ihre Vertrauenswürdigkeit überprüfbar.

Werden die Nutzungsreservierungen bei einem Lastmanager gesammelt, kann er sich auf bekanntgegebene Last vorbereiten und unerwartete Spitzen glätten. Außerdem ist mit der Bedarfsangabe durch die Nutzer ihre eindeutige Identifikation verbunden. Das Intrusion-Detection-System kann also vorab über die vermehrte Nutzung durch einen Client informiert werden und blockiert

den Vielnutzer nicht ungewollt. Somit lassen sich falsche Warnungen bezüglich Denial-of-Service-Attacken verhindern. Das steigert die Systemeffizienz.

Mit dem Wissen über das Gesamtsystem kann der Lastmanager die Anfragen besser einplanen. Da diese optimierte Ausführungsfolge alle Nutzer mit konstanterer Ressourcenmenge bedient, minimieren sich die Gesamtkosten. Zudem ist bei kurzzeitiger Überlast der Nutzer im Rahmen ausgehandelter SLAs zeitlich verzögerbar, um in der Zwischenzeit neue Ressourcen anzufordern. Der Systemkomponenten-Überblick in Abbildung 4.3 macht deutlich, dass ein entsprechendes Protokoll benötigt wird, um ein solches Ressourcenmanagement umzusetzen. Die konkrete Lösung findet sich in Kapitel 5.

Da in Cloud-Systemen häufig komplexe Abhängigkeiten unter den Betriebskomponenten bestehen, ist es oft schwer, die Auswirkungen steigender Arbeitslast bis in die untersten Schichten des Systems abzuschätzen. Hier können Engpässe auf tiefster Ebene entstehen, obwohl die höchste Anwendungsschicht scheinbar keine Probleme bei der Bearbeitung der Anfragen hat. Langfristig büßt der gesamte Dienst allerdings an Performanz ein. Die Ressourcenauslastung jeder einzelnen Schicht muss deshalb dokumentiert werden.

Um diesen enormen zusätzlichen Verwaltungsaufwand zu verhindern und außerdem die Effizienz der Instanzenverwaltung zu steigern, müssen Ressourcenabhängigkeiten analysiert werden. Ohne die Notwendigkeit der Auslastungsüberwachung jeder Ressource, ist dann mit adäquater Relationsabbildung das gesamte Cloud-System skalierbar. Genau dies ermöglicht die in Kapitel 6 beschriebene Protokollerweiterung. Der Lastmanager kann Cloud-Systeme so kalibrieren, dass ein Graph die Beziehungen der Ressourcen speichert. Auf dessen Basis werden mithilfe eines Regelverfahrens alle Ressourcen skaliert.

Die weitere Möglichkeit im Bereitstellungsprozess von Ressourcen besteht darin, die Cloud-Instanzen als passive Komponenten zu verstehen. Sie müssen dann überwacht werden, damit der Lastmanager Ressourenmängel erkennen und aktiv durch Maschinenzuschaltung für ausreichende Kapazitäten sorgen

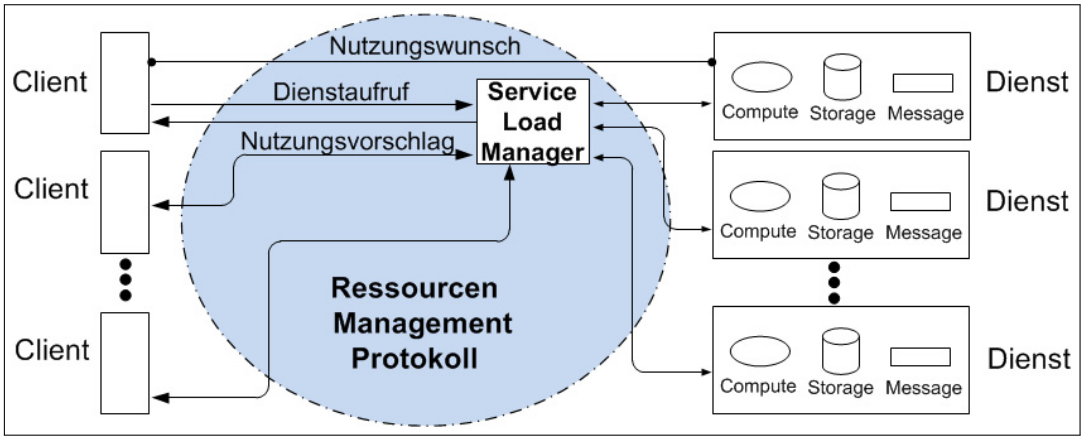


Abbildung 4.3: Ablauf zwischen anfragenden Clients und Cloud-Diensten

kann. Dafür müssen Historien erstellt und mithilfe dieser gesammelten Daten Vorhersagen bezüglich der weiteren Entwicklung der Arbeitslast auf den bereitgestellten Cloud-Servern getroffen werden. Dann kann durch diese Informationen errechnet werden, ob die bereits zur Verfügung gestellten Kapazitäten ausreichen, um das prognostizierte Anfragevolumen zu bearbeiten.

Falls die bestehenden Rechner nicht alle Anfragen bedienen können, erkennt dies der Lastmanager vor dem tatsächlich auftretenden Engpass. Neue Ressourcen werden dann frühzeitig angefordert, um einer Überlastung rechtzeitig entgegenzuwirken. Zur schnellen Behandlung des stetig wechselnden Arbeitslastverhaltens lassen sich zudem Informationen aus sozialen Netzwerken für Prognosen nutzen. Dort stellen die Nutzer Daten über ihr Interesse und ihre Aufenthaltsorte jedem frei zur Verfügung. Sind diese Auskünfte extrahiert, kann von dem kommunizierten gesteigerten Nutzerinteresse an einer Anwendung auf ein Anwachsen in der Lastkurve bezüglich der korrespondierenden Applikationsressourcen geschlossen werden. Mit dieser Vorausrechnung ist rechtzeitig auf einen erhöhten Ressourcenbedarf zu reagieren.

Der zweite Teil der Überlegung ist, von den mitgeteilten Aufenthaltsorten der Nutzer auf die Lokalität des Hauptbedarfs an Ressourcen zu schließen. Dann werden eher dort Ressourcen gestartet, um unnötige Latenzen durch langes Routen an entfernte Rechner zu verhindern. Dabei befinden wir uns auf einer höheren Abstraktionsebene und der Nutzer ist nicht bewusst in die Entscheidung involviert, wann und wo Instanzen gestartet werden. Vielmehr werden die notwendigen Daten aus von ihm veröffentlichten Informationen im sozialen Netzwerk abgeleitet. Ein Beitrag zur Realisierung dieses Lösungswegs wird in Kapitel 7 ausführlich beschrieben.

4.5 Zusammenfassung

Das komplexe Zusammenspiel der drei Ebenen des Cloud Computings, Infrastructure-as-a-Service, Platform-as-a-Service und Software-as-a-Service, stellt einige Herausforderungen an seine Nutzer und Betreiber. Es müssen der Ressourcenbedarf, anfallende Gebühren, Bereitstellungsdauer, Verträge und vieles mehr beachtet und umgesetzt werden, um ein effizientes Handeln auf allen beteiligten Schichten zu ermöglichen.

Die Herleitung von Arbeitslastmustern aus bestehenden Datensätzen ist oft hilfreich, um Änderungen im Lastverhalten besser vorzubereiten. Diese Profile unterteilen sich in On-Off-Situationen, exponentielles Wachstum, hervorstechende Lastspitzen, zyklische Muster und Ereignisse, die genau planbar sind. Nach der Mustererkennung ist es im Prinzip möglich, die zur Bearbeitung aller Nutzeranfragen nötige Ressourcenmenge rechtzeitig bereitzustellen. Dies ist allerdings noch von keinem Cloud-Anbieter voll automatisiert umgesetzt.

Um ein Verständnis über den Forschungsstand der Green IT und für die ökologischen und ökonomischen Aspekte des Cloud Computings zu schaffen, muss

man sich klar machen, welche Kosten im Cloud-Betrieb anfallen. Außerdem muss überlegt werden, welche Architektur für das System zu wählen ist und wie die Gebühren im multimandantenfähigen Cloud-Umfeld umgelegt werden können. Dieses Kapitel stellte einen Einblick in das umfangreiche Thema dar und führte die verfolgten Ansätze ein.

Bei der Interpretation der in diesem Kapitel angeführten Beispiele ergibt sich, dass der Zuschaltung und Abschaltung der Ressourcen große Aufmerksamkeit geschenkt werden muss. Dem kann man sich von verschiedenen Seiten näheren. Entweder können die beteiligten Komponenten aktiv Ressourcen reservieren oder sie müssen überwacht werden, um ihren Bedarf zu decken. Hierfür muss man zum einen erkennen, *wann wie viele* Ressourcen hinzugeschaltet werden sollen. Zum anderen muss die Entscheidung getroffen werden, *wo* diese Ressourcen hoch- oder heruntergefahren werden sollen.

Zur Lösung wird ein Protokoll zum Ressourcenmanagement entwickelt, das auch Abhängigkeiten der Instanzen bewältigen kann. Über einen umfassenden Algorithmus wird mittels sozialer Netzwerkdaten die automatisierte Adaption an die Last und den Ort mit dem Hauptbedarf ermöglicht.

5 Ein Protokoll zur Ressourcenverwaltung

Im vorausgehenden Kapitel wurden Überlegungen zur Ressourcenbereitstellung angestellt. Dabei zeigt sich der Bedarf zur besseren Steuerung der Anzahl benötigter Kapazitäten zu einem ausgewählten Zeitpunkt. Es wird unter anderem eine Herausforderung genannt, die im Cloud-Bereich besonders schwer zu lösen ist: Denial-of-Service-Attacken. Diese beabsichtigten, schadhaften Dienstüberflutungen sind in Systemen, die nach dem standardmäßigen Best-Effort-Modell des Internets handeln, leicht realisierbar. Bei dieser Architektur versucht jede Ressource unabhängig von ihrem Belegungsstatus, immer alle Service-Anfragen zu beantworten. Das bedeutet, selbst wenn die Server bereits vollständig ausgelastet sind, nehmen sie stets neu eintreffende Requests an und vernachlässigen so die bereits begonnenen Jobs [187].

Als Konsequenz kann der Betrieb zum Erliegen gebracht werden, indem ein Angreifer der Umgebung in schneller Folge sinnfreie Aufrufe schickt. Die Systemkomponenten werden damit fortlaufend von der Abarbeitung echter Anfragen abgehalten. Da dies zu erheblichen Performanzeinbußen führt, ist demnach ein reines Best-Effort-Modell ungeeignet für Cloud-Systeme.

Ungeeignet sind außerdem Algorithmen, die einzig auf Basis konfigurierter Schwellwerte die Instanzenskalierung umsetzen. Sie sind nicht in der Lage zu erkennen, hinter welchen Anfragen tatsächlich ein echter Nutzer steckt und welche Dienstanforderungen nur zur Sabotage geschickt werden. In diesem Fall werden durch die heutigen einfachen regelbasierten Skalierungsmechanismen immer mehr Ressourcen bereitgestellt. Bis die Attacke erkannt ist, laufen so beim Systemanbieter immer höhere Kosten auf.

Ein solcher Vorgang ist von vornherein zu verhindern. Um dabei möglichst genau die gewünschte Kapazität zu bieten, besteht die Möglichkeit, die Cloud-Komponenten aktiv in den Prozess der Bereitstellung einzubinden. In diesem Kapitel wird der implementierte Mechanismus vorgestellt, der dynamisches automatisiertes Arbeitslastmanagement für Cloud-Dienste realisiert. Das entworfene Protokoll zur Ressourcenverwaltung definiert durch einen Service-Load-Manager die Abarbeitungsreihenfolge und die benötigte Maschinenanzahl. Dies beruht auf der Bedarfsreservierung durch die Dienstnutzer. Der Schedule kann durch den Dienst beeinflusst werden, indem der Cloud-Service die Clients im Rahmen der zuvor vereinbarten Service-Level-Agreements umsortiert.

Im Folgenden wird zunächst auf die heutige Technik der Autoskalierung durch einfache Regeln eingegangen und erläutert, warum die bestehenden Mecha-

nismen nicht ausreichen. Als adäquate Lösung dafür wird das entwickelte Reservierungs- und Feedback-basierte Protokoll im Detail vorgestellt. Das Protokoll wird mathematisch evaluiert und in Simulationen getestet. Wie aus der Beschreibung der Ergebnisse ersichtlich, ist die Nutzung des Protokolls vielversprechend und reduziert Antwortzeiten sowie Verwurfsraten.

5.1 Verwandte Arbeiten zur automatischen Ressourcenskalierung

Die Skalierung verteilter Ressourcen ist seit vielen Jahren Forschungsgegenstand. Grundlegend kann ein Cloud-Anbieter zur Verfügbarkeitsplanung das Ressourcenvolumen entweder an der Durchschnittsbelegung oder anhand entstehender Spitzenlasten anpassen [15]. Im ersten Fall reicht im Standardablauf eine kleinere Menge aus. Da wenige Instanzen ungenutzt bleiben, sind die Kosten meist gering. Andererseits müssen bei extremen Lastanstiegen Anfragen verworfenen werden, da zu geringe Kapazitäten bereitstehen. Diese Problematik kann umgangen werden, wenn die Ressourcenmenge über die Höchstlast konfiguriert ist. Allerdings entstehen so deutlich höhere Kosten. Um eine praktikable Lösung zu finden, wird Autoskalierung eingesetzt. Die nötige Anzahl an Instanzen wird anhand der anliegenden Last automatisch bestimmt [16].
Durch die Konfiguration von Performanzmetriken und Schwellwerten können die Ressourcen der Nutzer selbstständig skalieren. Dies erfolgt über sogenannte Trigger, also definierte Auslöser. Ein Trigger könnte zum Beispiel festlegen, dass bei der Performanzmessung im Falle einer länger als vier Minuten andauernden Überschreitung der CPU-Auslastung von 50 %, automatisch eine zusätzliche Instanz startet. Meist muss zur Erstellung solcher Trigger eine Historie erzeugt werden.
Urgaonkar et al. [173] schlagen dafür einen ‘Workload Predictor’ vor, der sowohl prädiktiv wie auch reaktiv in ihrem dynamischen Bereitstellungssystem für mehrschichtige Internetanwendungen eingesetzt wird. Der prädiktive Mechanismus erkennt durch stündliche Aufzeichnungen uhrzeitbedingte und saisonale Trends. Der minütlich angestoßene reaktive Algorithmus vergleicht die momentane Session-Rate mit der vorhergesagten und korrigiert den Verlauf bei Abweichungen. Dadurch wird zwar der Durchsatz erhöht, doch ist die Auslastung sehr gering, da in diesem Algorithmus je physikalischer Ressource nur eine virtuelle Maschine laufen kann. Außerdem werden weder verschiedene VM-Typen unterstützt noch die Gesamtkosten berücksichtigt.
Padala et al. [131] stellen ein kontrolltheoretisches Blackbox-System vor, das mithilfe von Feedbacks mehrschichtige Anwendungen dynamisch verwaltet. Das System erstellt periodisch einen angemessenen Allokationsplan aller Maschinen, wobei die Instanzennutzung als Blackbox-Eingabe dient. Auch Lim et al. [99] haben ein Feedback-Prinzip entworfen, welches das Ressourcenmanagement durch proportionales Thresholding automatisiert. Die

Performanzmetriken sind in einem flexiblen Bereich festgelegt und beziehen sich auf die durchschnittliche CPU-Nutzung. So wird die minimale Anzahl an Ressourcen zur Verfügung gestellt, welche die Nutzeranforderungen noch erfüllt. Der Ansatz ist allerdings in dieser einfachen Form zu ungenau, da zwar erkannt wird, wann, jedoch nicht wo neue Ressourcen erforderlich sind.

Neben reaktiven Verfahren gibt es auch Algorithmen, die mit der Vorhersage von Arbeitslast agieren. Beispielsweise prognostiziert der vorausschauende Ressourcenallokationsmechanismus von Roy et al. [146] zur Kostenoptimierung die aufkommende Last mit der 'Autoregressive Moving Average Method', einer modellprädiktiven Regelung. Dabei wird das Nutzerverhalten durch einen Graphen mit Log-Informationen modelliert, um über die Gesamtzahl der Nutzer die zu erwartende Durchschnittslast zu berechnen. Auch Chieu et al. [35] skalieren dynamisch mithilfe von Schwellwerten und periodischen, statistischen Trend-Berechnungen über festgelegte Regeln. Dies führt zwar zu kurzen Antwortzeiten und hoher Zuverlässigkeit, geht dabei allerdings von einschichtigen Anwendungen aus. Diese sind im Cloud-Umfeld unüblich.

Einen heuristisch geprägten Ansatz verfolgen Mao et al. [106], indem die Plangenerierung für den Instanzenstart zum Optimierungsproblem erklärt wird. Die Lösung erfolgt über Integer-Programming und verwendet Metriken auf Anwendungsebene. Den Autoren zufolge spiegelt die dabei gemessene Job-Antwortzeit die Performanzanforderungen der Nutzer besser wider. Das führt zu präzisen Skalierungsinstruktionen. Zur Kostenminimierung sollen alle Anfragen vor Ende der spezifizierten Deadline bedient sein. Der Nutzer muss aufwändig die minimale Instanzenanzahl angeben, die er zur Diensterbringung verlangt. Bei Budgetüberschreitungen muss er aktiv entscheiden, ob zusätzliche Instanzen gestartet oder die Bearbeitungsfristen verlängert werden.

Insgesamt reflektieren also bestehende Lösungen mit ihren Messgrößen die Systemnutzung zwar recht gut, doch gilt es für den Nutzer als enorme Herausforderung leistungsfähige Metriken und passende Grenzwerte zu definieren. Erschwerend kommt hinzu, dass der Cloud-Dienst meist nicht im Voraus abschätzen kann, wann welche Anfragemengen zu bewältigen sind. Obwohl die exakte Prognose schwer durchführbar ist, können häufig wiederkehrende Muster erkannt werden. Gerade im Machine-to-Machine-Bereich folgt die Anfragenbearbeitung oft strikten Zeitplänen. Damit ist es in vielen Szenarien möglich, den Ressourcenbedarf vorab zu bestimmen. Nichtsdestotrotz werden fatalerweise in den meisten Fällen die benötigte Zeit zum Instanzenstart und das Finden der optimalen Maschinenlokalität nicht berücksichtigt. Zudem führt die stündliche Abrechnung der genutzten Kapazität zu enorm hohen Kosten, wenn Maximallast-Situationen beispielsweise nur zehn Minuten andauern. Dafür wäre ein angepasstes Bezahlmodell erforderlich.

Bei den aktuell bereitgestellten Techniken besteht demnach erhebliches Verbesserungspotenzial. Dort setzt das folgende Ressourcenmanagement-Protokoll an, das teilweise in [156] veröffentlicht ist.

5.2 Ressourcenmanagement mittels Reservierung und Feedback

Durch ein neu gestaltetes dynamisches Ressourcenmanagement mittels Reservierung und Feedback ist das Lastverhalten effizienter zu organisieren. Der Algorithmus basiert einerseits auf der Anmeldung des geschätzten Instanzenbedarfs durch die Clients. Andererseits können die Cloud-Services über eine Vermittlungskomponente auf das letztendliche Dienstnutzungsverhalten regulierend einwirken. Gleichzeitig muss die Bearbeitung der Anfragen den vereinbarten SLAs gemäß erfolgen. Der Dienstnutzer repräsentiert dabei eine tiefere Ebene, indem er selbst ein Rechner ist. Er fordert dann den bereitgestellten Cloud-Service an, um eine eigene Dienstleistung zu schaffen.

Zwischen die Nutzer und Dienste wird im Protokoll der Service-Load-Manager als eben dieser Vermittler geschaltet. Abbildung 5.1 stellt diese Beziehung dar. Dabei wird die Lastmanagement-Komponente auf Client-Seite adaptiert. Die Nutzer reservieren nun proaktiv ihre nötige Kapazität beim Manager. Dieser kann daraufhin vorab die erforderliche Instanzenanzahl für die Bewältigung der gesamten Last ermitteln. Der aggregierte Bedarf wird schließlich bei der Ressourceneinteilung der Dienste berücksichtigt, um auf bekannte Lastspitzen vorbereitet zu sein oder auch um unerwartete Ausschläge zu glätten.

Der Dienst kann zudem die Clients in ihrer eigentlichen Angebotsnutzung beeinflussen. Er ist in der Lage, Anfragen im vom Benutzer erlaubten Intervall beliebig einzuplanen. Bestimmte Requests können so im Rahmen der SLAs zeitlich verzögert werden, um kurzfristige teure Skalierungsprozesse zu verhindern. Der Anwender wird durch eine Feedback-Nachricht über den ihm zugeteilten Nutzungszeitraum informiert.

Durch das Feedback wird ihm auch eine mögliche Systemüberlastung mitgeteilt. Eine erneute Anfrage des Clients würde unnötig weitere Last generieren. Deshalb wartet er ab, bis er vom Manager über die Bereitstellung zusätzlicher Kapazität benachrichtigt wird.

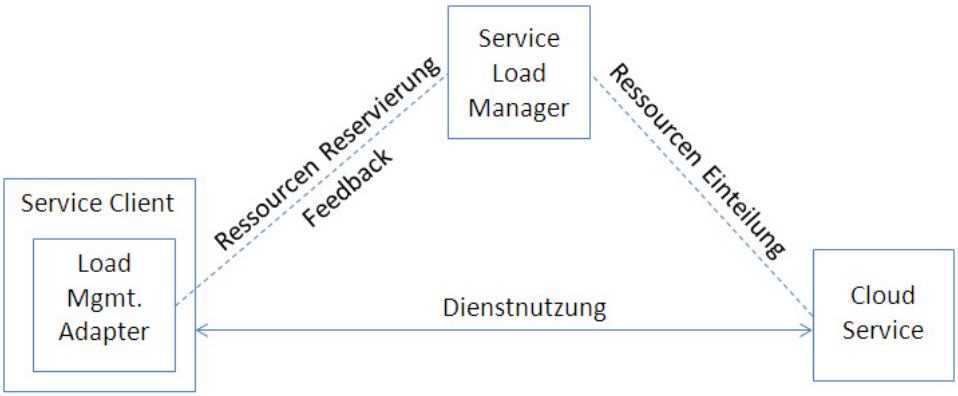


Abbildung 5.1: Beziehung zwischen den beteiligten System-Komponenten

Auf diese Weise lässt sich die Deckung des Bedarfs aller Nutzer bei der Dienst-erbringung verbessern. Gleichermaßen können die benötigten physikalischen Ressourcen durch die höchstmögliche Auslastung effizient verwendet werden.
Dabei können neben den Nutzern, die das Ressourcenmanagement-Protokoll implementieren, auch Clients bedient werden, die diesen Ablauf nicht unterstützen. Der Unterschied in der Behandlung besteht darin, dass bei kurzfristiger Überlast die protokollunterstützenden Kunden bevorzugt werden.
Die Interaktion zwischen Client und Service erfolgt also indirekt über das Protokoll. Abbildung 5.2 zeigt mit der indirekten Interaktion zwischen Client und Service, dass verschiedene Kriterien zur Qualitätssicherung des Ablaufs eine bedeutende Rolle spielen. Die ausgehandelten Service-Level-Agreements sind ein zentraler Aspekt der Protokollgüte. Sie werden anhand der Quality-of-Service-Parameter der vom Dienst bereitgestellten Ressourcen überprüft. Bei guten Resultaten führen sie während der Dienstinstanzennutzung zu einem hohen Grad an Zufriedenheit beim Client.
Grundsätzlich besteht jedes Protokoll aus drei entscheidenden Segmenten: einem Informationsmodell, einem Kommunikationsmodell und einem Aktivitätsmodell [161]. Im Informationsmodell werden die ausgetauschten Nachrichtendaten und das dafür verwendete Format festgelegt. Das Kommunikationsmodell beschreibt den Ablauf zwischen den beteiligten Komponenten. Schließlich definiert das Aktivitätsmodell, welche Aktionen in den Systemelementen nach dem Empfang einer Information ausgeführt werden. Diese Protokollbestandteile werden in den kommenden Unterkapiteln detailliert erklärt.

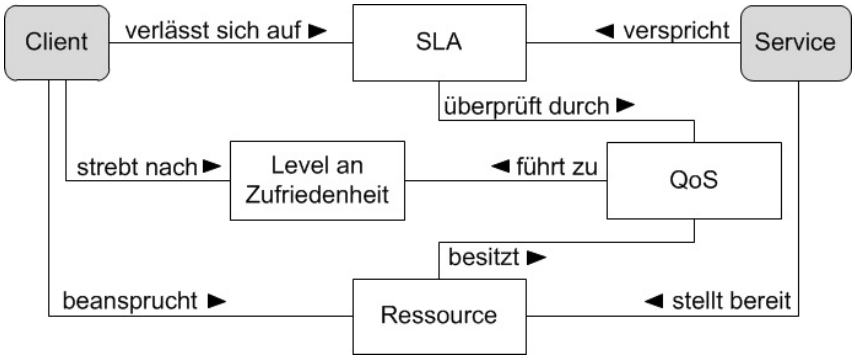


Abbildung 5.2: Darstellung der Interaktion zwischen Client und Service

5.2.1 Informationsmodell des Reservierungs- und Feedback-basierten Protokolls

Um die Anfragen der richtigen Komponente zuzuordnen, muss das Protokoll in der Lage sein, Clients und Dienste voneinander zu unterscheiden. Bei Überlastung soll sichergestellt sein, dass die bedeutendsten Requests nicht verworfen werden. Deshalb müssen mit der Befolgung des Ablaufs die Dienstanfragen

priorisierbar sein. Außerdem ist zu gewährleisten, dass niederpriore Nutzer nicht beliebig oft hinten angestellt und letztlich nie bedient werden. Dafür wird im Informationsmodell festgehalten, dass sich die ausgetauschten Daten über das folgende Nachrichtenformat definieren.

- **Client-ID:** Zur eindeutigen Nutzerdifferenzierung trägt der Client-Identifikator bei. Bei häufiger Dienstanfrage durch denselben Client wird seine Identität mit den in einer Datenbank gespeicherten Identifikationsnummern der Nutzer verglichen. Existiert die Nummer des Vielnutzers in der Datenbank, kann er als vertrauenswürdig eingestuft werden, da er vor seiner Ablage in der Datenbank durch den Service-Load-Manager überprüft wurde. Dadurch sind Fehlinterpretationen des Intrusion-Detection-Systems vermeidbar, die zu einem Denial-of-Service-Alarm und anschließenden Dienstausfall führen.
- **Manager-ID:** In Systemen mit großer Anfragedichte kann es vorkommen, dass ein einzelner Service-Load-Manager nicht ausreicht, um die korrekte Gesamtverwaltung aller Komponenten zu gewährleisten. Aus diesem Grund ist der Manager replizierbar. Die einzelnen Verwalter können über die Manager-Identifikatoren unterschieden werden.
- **Service-ID:** Welcher Dienst explizit vom Client angesprochen wird, ist über den Service-Identifikator gekennzeichnet.
- **SLA-ID:** Die Service-Level-Agreements, die zwischen Client und Dienst ausgehandelt sind, müssen bezüglich ihrer Relevanz der Client-Ressourcenanfragen gewichtet werden können. Deshalb sind sie in drei Priorisierungskategorien unterteilt.

  a. Der SLA-Identifikator 0 steht dafür, dass der Nutzer weder das entwickelte Protokoll unterstützt noch irgendwelche anderen SLA-Garantien mit dem angesprochenen Dienst festgelegt hat.
  b. Durch einen Service-Level-Rang mit dem Identifikator 1 wird die Protokollunterstützung des Clients signalisiert. Diese Zusicherung gilt allerdings für niederpriore Anfragen mit geringer Dienstgüte.
  c. Mit einer SLA-Wertigkeit von 2 wird sichergestellt, dass Clients dieser Kategorie im Vergleich zu niedrig priorisierten bevorzugt bedient werden. Sie unterstützen das Protokoll und verfügen außerdem über die höchste Absicherung an Service-Level-Agreements.

- **Resource-Demand:** Wie viele Ressourcen vom Client benötigt werden, gibt der vom Nutzer festgelegte Ressourcenbedarf-Parameter an.
- **Start-Time:** Der Zeitpunkt des Beginns der Ressourcennutzung wird über das Startzeit-Attribut bestimmt.
- **Duration-of-Usage:** Die Intensität der Maschinenbelegung durch einen Client ist über den Wert der Nutzungsdauer ermittelbar.

- **Shifting-Proposal:** Im Falle einer Überlastung können Clients mit niedriger SLA-Wertigkeit zeitlich durch den Service-Load-Manager verzögert werden. Dafür unterbreitet er dem Client einen Verzögerungsvorschlag, bis die neu gestarteten Ressourcen zur Verfügung stehen.
- **Deadline:** Außerdem wird vom Client festgelegt, bis wann seine Anfrage spätestens bearbeitet sein muss. Diese Frist bestimmt, wie oft der Nutzer-Request maximal verzögert werden darf.

5.2.2 Kommunikationsmodell des Reservierungs- und Feedback-basierten Protokolls

Dienste im Cloud-Bereich fallen meist in die Kategorien der Kalkulations-, Speicher- oder Nachrichtenanwendungen. Das Kommunikationsmodell sieht vor, dass sich diese Dienste beim Service-Load-Manager registrieren. Der Manager speichert sie mit zugehöriger *Service-ID*. Anschließend werden die ausgehandelten Nutzungsbedingungen mit ihrem Rang als *SLA-IDs* festgehalten. Der Service-Load-Manager ist nun bereit für Anfragen von Nutzern. Der Protokollablauf wird in Abbildung 5.3 veranschaulicht.

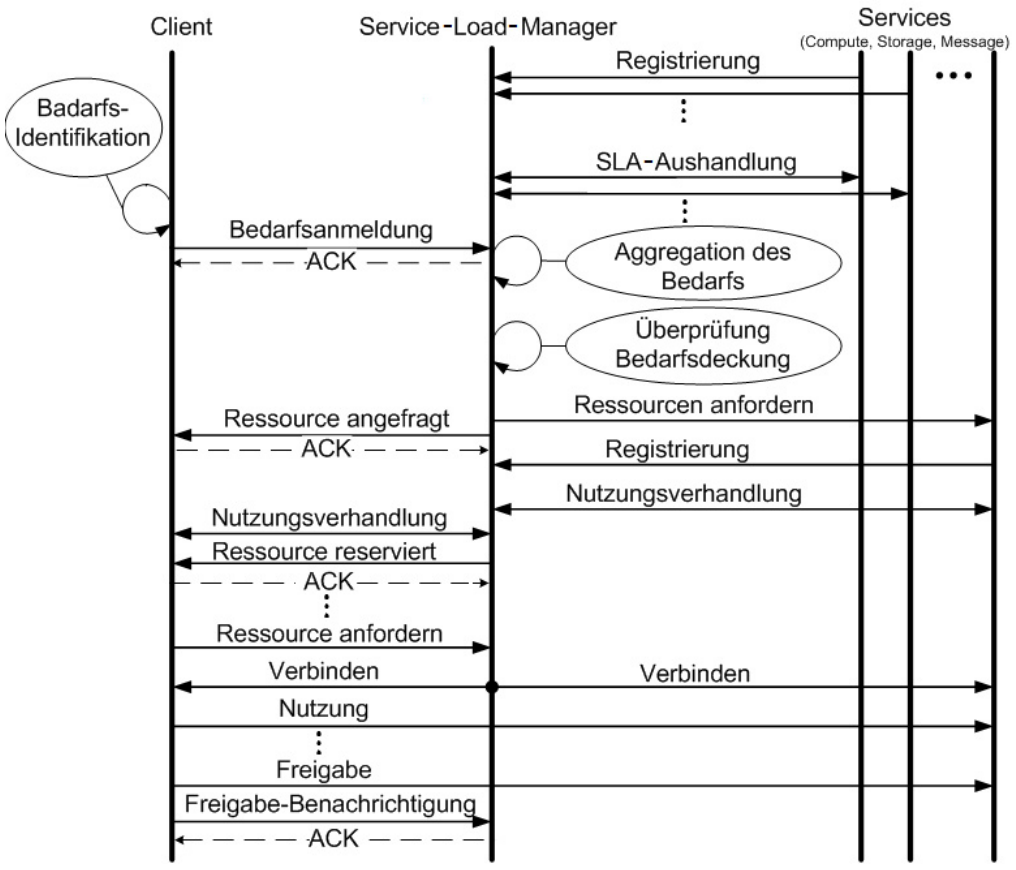


Abbildung 5.3: Protokollablauf bei Überlastung des Systems

Der Client schätzt zunächst seinen Bedarf und meldet diesen *Resource-Demand* inklusive der *Start-Time*, seiner *Duration-of-Usage* und der *Deadline* dem Service-Load-Manager. Dieser legt die empfangenen Werte zusammen mit der *Client-ID* ab. Alle Anfragen werden beim Service-Load-Manager aggregiert. Außerdem überprüft der Manager, ob der Gesamtbedarf mit den bereits registrierten Ressourcen zu decken ist.
Wenn zu viele Nutzungsanforderungen zum gleichen Zeitpunkt starten, kann es zu einem Ressourcenengpass kommen. Sollten die Instanzen nicht ausreichen, versucht der Manager den Schedule neu zu planen. Darüber werden die Nutzer per Feedback informiert. Indem im Schedule – wo möglich – die Anfragen um das *Shifting-Proposal* aufgeschoben werden, können häufig alle Clients mit der bestehenden Kapazität bearbeitet werden.
Allerdings können nach der Verzögerung im anschließenden Zeitintervall bei vielen Nutzern die jeweiligen *Deadlines* erreicht werden. Beim Auftreten dieser nicht bewältigbaren Lastspitzen fragt der Service-Load-Manager zusätzliche Maschinen bei den Cloud-Services an. Der Client wird über den Neustart benachrichtigt. Sobald die Ressourcen bereitstehen, werden die Nutzungsparameter verhandelt. Da die Kapazität zum ausgehandelten Zeitpunkt für den Nutzer reserviert ist, kann er sie entsprechend der gespeicherten Rahmenbedingungen anfordern.
Nachdem der Client mit dem Dienst verbunden ist, nutzt er ihn für die angemeldete *Duration-of-Usage* und gibt ihn danach wieder frei. Der Service-Load-Manager kann nun die frei gewordenen Ressourcen wieder in der Erstellung des Schedules für die Maschinenbelegung mit einbeziehen.

5.2.3 Aktivitätsmodell des Reservierungs- und Feedback-basierten Protokolls

Neben den Registrierungsvorgängen beim Service-Load-Manager dienen die internen Aktivitäten der Services grundlegend der Bereitstellung ihrer Fähigkeiten. Um einen Plan zur Ressourcennutzung zu generieren, muss der Dienst die Instanzensteuerung vom Service-Load-Manager berücksichtigen. Deshalb übergibt ihm der Service-Load-Manager die durch den Client gemeldeten Nutzungszeiten und mögliche Aufschubintervalle. Der Ausführplan ist durch diese Absprache optimierbar. So wird unnötige Ressourcenvergeudung vermieden und eine höhere Auslastung erreicht.
Im Vergleich zu den Cloud-Diensten gestalten sich die Abläufe innerhalb des Clients und Service-Load-Managers bei der Protokollbefolgung wesentlich komplexer. Die folgenden Darstellungen dienen dem besseren Einblick in die Aktivitäten dieser beiden Komponenten.
Das Ablaufdiagramm 5.4 zeigt, dass der Client nach der Bedarfsidentifizierung überprüft, ob die nötigen Ressourcen bereits für ihn reserviert sind. Wenn dies nicht zutrifft, kann er die Kapazität direkt beim Service-Load-Manager

einfordern, solange seine Behandlung zeitkritisch und er aufgrund einer hohen *SLA-ID* dazu berechtigt ist. Sind genügend Ressourcen frei, wird er mit dem Dienst verbunden. Anschließend gibt er den Service wieder frei und informiert den Service-Load-Manager darüber.

Soll Kapazität vorgemerkt werden, untersucht der Client, ob für diesen Fall bereits Lastmuster aus der Vergangenheit bestehen. Bei positiver Rückmeldung wird die Reservierung zugesagt. Wurde zu früherer Zeit noch keine Mustererkennung durchgeführt, folgt die Prüfung, ob dies aufgrund gesammelter Daten möglich ist. Dann kann das extrahierte Muster gespeichert werden und die anschließende Buchung erfolgen. Dieser Vorgang ist mit der Reservierungsbestätigung abgeschlossen.

Nun kann der Client die reservierten Instanzen zum vereinbarten Zeitpunkt anfordern und den Dienst dementsprechend nutzen und freigeben. Da das dynamische Ressourcenmanagement vom Client optional nutzbar ist, kann die

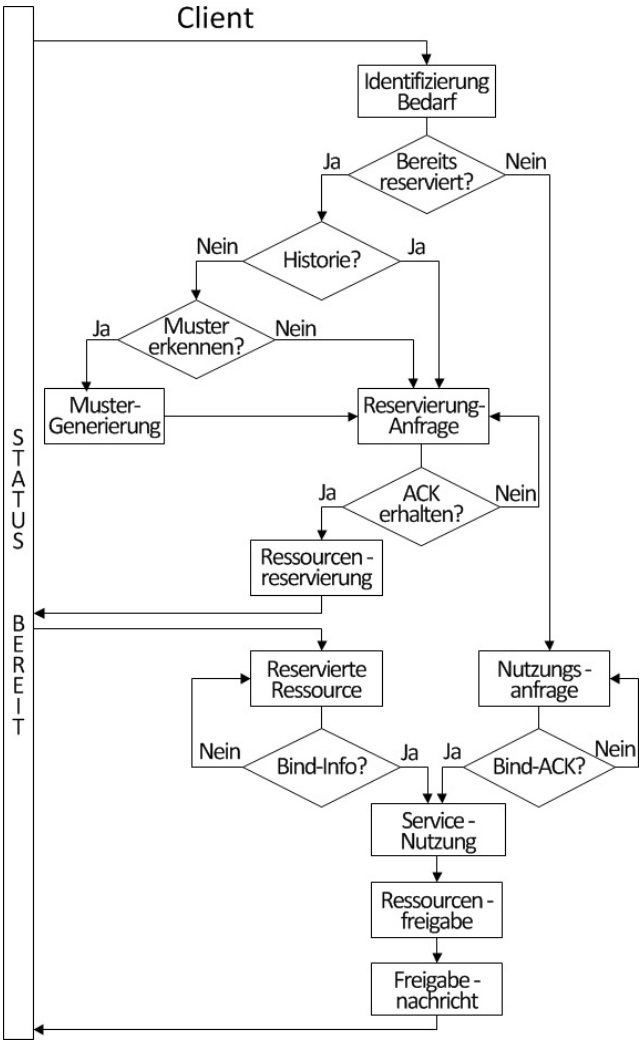


Abbildung 5.4: Internen Aktivitäten im Client

Kompatibilität des Gesamtsystems zu bestehenden Nutzungsanfragen bereits registrierter Nutzer gewährleistet werden.
Das Aktionsmodell des Service-Load-Managers wird in Abbildung 5.5 dargestellt. Geht beim Manager eine Anfrage zur Dienstregistrierung ein, kontrolliert er, ob sich eine zugehörige *Service-ID* bereits im Register befindet. Liegt eine Neuregistrierung des Dienstes vor, wird sein Identifikator abgelegt. Daraufhin kann ein Verbindungsaufbau beziehungsweise die Aushandlung der Nutzungsbedingungen mit dem Dienst erfolgen. Jede eingehende Anfrage wird vom Service-Load-Manager in einem Pool aggregiert, der den gesamten Bedarf widerspiegelt. Wenn die Deckung des Gesamtbedarfs möglich ist, können die Nutzungsbedingungen mit dem Client verhandelt werden. Die Ressourcen werden für den Client reserviert, sobald eine Übereinkunft erzielt ist.
Falls die bestehende Ressourcenmenge nicht ausreicht, werden neue Maschinen

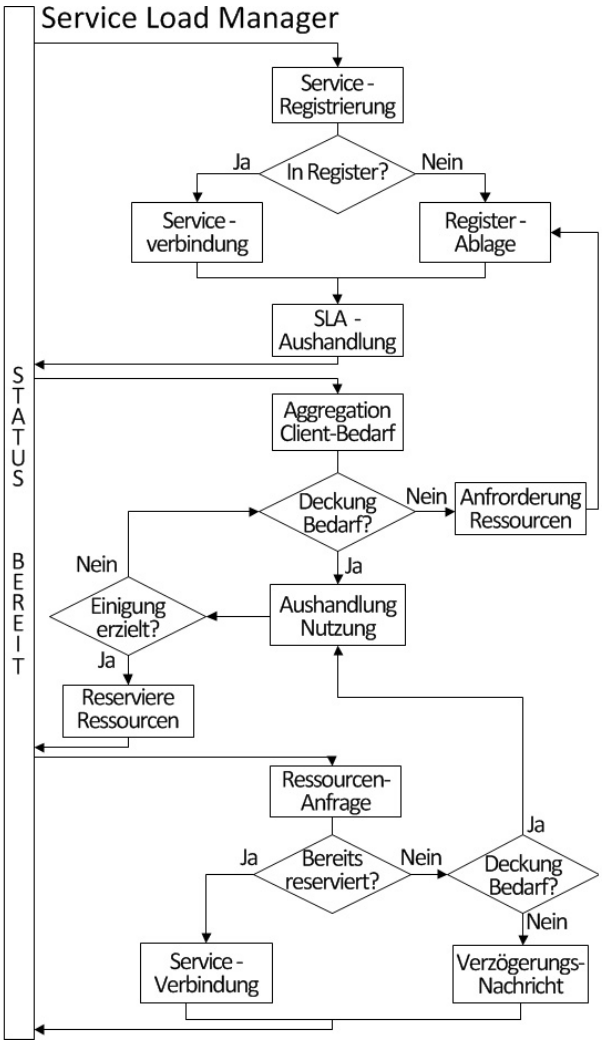


Abbildung 5.5: Internen Aktivitäten im Service-Load-Manager

angefordert. Der Prozess wird beim Eintreffen der Zusatzkapazität mit dem Registrierungsschritt wieder aufgenommen.
Nach einer Ressourcenanfrage durch den Client verifiziert der Service-Load-Manager, ob es eine korrespondierende Reservierung gibt. Bei einem solchen Nachweis verbindet der Manager den Client mit dem Dienst.
Im anderen Fall wird der Client bei einer Systemüberlastung verzögert, um die Instanzen den protokollunterstützenden Nutzern mit Reservierung bereitzustellen. Dieser Client erhält die Information, dass seine Anfrage im Rahmen des von ihm bestimmten Verschiebungsintervalls umsortiert wird. Innerhalb der gesetzten *Deadline* ist garantiert, dass der Nutzer den benötigten Dienst zugesprochen bekommt, und zwar sobald bestehende Kapazitäten freigegeben oder neue virtuelle Maschinen gestartet sind.
Da in Cloud-Systemen die Anzahl an Teilnehmern beliebig wachsen kann, muss auch die Skalierbarkeit des Service-Load-Managers sicher gestellt sein. Dies wird durch die selbstständige Replizierung der Komponente erreicht. Die Manager unterscheiden sich dann über ihre *Manager-ID*.

5.3 Entwurf der System-Architektur

Für die Umsetzung des Protokolls wird eine passende Architektur zur Systemimplementierung benötigt. Wie im vorangegangenen Kapitel erläutert, beeinflussen im Cloud-Bereich viele Faktoren die Wahl einer Entwicklungsumgebung. Folglich ist in diesem Umfeld eine solche Entscheidung höchst komplex und die konkrete Realisierung stellt eine Herausforderung dar.
Bei der Betrachtung der drei marktführenden Cloud-Anbieter Microsoft, Google und Amazon fällt auf, dass sie wegen ihrer unterschiedlichen infrastrukturellen und programmatischen Grundlage schwer zu vergleichen sind.
Wie in Kapitel 2.3 erläutert, bietet die Google App Engine nur eine eingeschränkte Laufzeitumgebung, ohne Sockets oder Schreibmöglichkeiten in Dateien. Der Nutzer erhält auch lediglich Zugriff auf ein nicht-relationales Datenbankmanagementsystem. Allerdings sind die angebotenen Kapazitäten nahezu unbegrenzt. So skalieren eigene Anwendungen in maximalem Umfang.
Mit der Lösung sind günstig skalierbare Anwendungen für Netzwerke umzusetzen, doch ist die Programmierfreiheit für die Zwecke des entworfenen Ressourcenmanagement-Protokolls zu eingeschränkt. Da der Zugriff auf tiefer liegende Dienstschichten erschwert ist, kann das Modell der Google App Engine schnell ausgeschlossen werden.
Amazon Web Services unterstützen eine Lösung, die dem Kunden lediglich eine Implementierungsbasis in Form von Hardware und Speicher bereitstellt. Infolgedessen muss ein Nutzer bei der Anwendungsentwicklung zuerst die komplette Programmierumgebung inklusive Betriebssystem und Entwicklungssoftware selbst einrichten. Während der Implementierung müssen alle Schichten durch

ihn gänzlich eigenständig überwacht und auf dem neuesten Sicherheitsstand gehalten werden. Das birgt viele Gefahrenpunkte.
Trotz guter Kontrollmöglichkeiten und Flexibilität auf unterster Infrastrukturschicht eignet sich Amazon nicht für das umgesetzte Verfahren. Man benötigt umfassende Kenntnisse über den Ausbau zugrundeliegender Rechenzentren. Das resultiert in einem zu hohen Aufwand für die Systemadministration.
Microsoft Azure bietet eine etwas eingeschränktere Cloud-Umgebung an, da sie erst auf der Plattform-Ebene aufsetzt. Obwohl dabei weniger Kontrolle über die zugrundeliegende Infrastruktur gegeben ist, eröffnet sich ein breites Feld zur Anwendungsentwicklung. Es müssen mehrere Ebenen des OSI-Schichtenmodells implementiert werden. Obwohl dies zusätzlichen Programmieraufwand erfordert, ist dieser im Vergleich zum völlig neu aufzusetzenden System bei Amazon vertretbar.
So stellte sich für die Umsetzung des Ressourcenmanagement-Protokolls Microsoft Azure als bester Lösungsansatz heraus. Diese Plattform bietet das gewünschte Maß an Freiheit und gleichzeitig vordefinierte, erprobte Tools und Features. Damit lassen sich ideal Ergebnisse zur Performanzmessung mit und ohne Nutzung des Protokolls erzeugen.
Abbildung 5.6 gibt einen Überblick zur Systemarchitektur der entworfenen Lösung. Dabei werden die Dienste über das nicht besonders mächtige Azure-Managementsystem überwacht und verwaltet. Die Performance Counter speichern hierfür Werte zu Reaktionszeiten und CPU-Auslastung.
Darauf setzt das entwickelte Protokoll auf. Seine Implementierung ermöglicht die automatische Skalierung der Instanzen und überprüft, ob der Systemzustand akzeptabel ist. So sind die gemessenen Resultate mit und ohne Protokollnutzung vergleichbar. Um sicherzustellen, dass zudem die unverfälschte, tatsächliche Ende-zu-Ende-Antwortzeit gemessen wird, übergibt auch der Client seine Messung an die Messkomponente.
Die Grundidee des Protokolls ist auf Basis der Microsoft *Windows Azure Platform* umgesetzt. Grundsätzlich stehen auf dieser Web-Rollen und Worker-Rollen zur Verfügung. Mithilfe von Internet-Information-Services, kurz IIS, und ASP.NET kann die Web-Rolle kundenspezifisch zur Programmierung

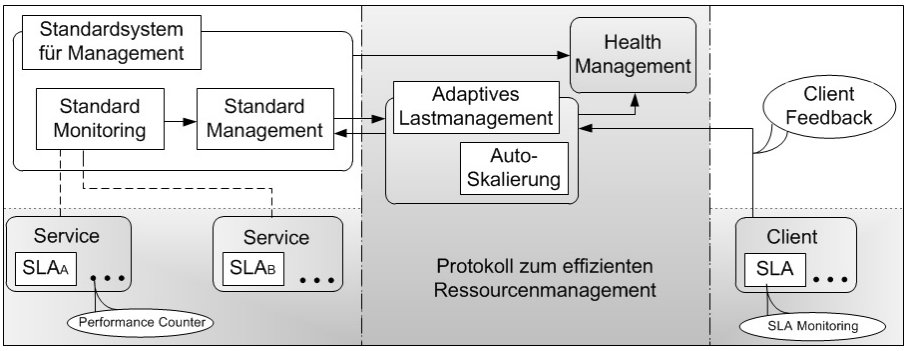


Abbildung 5.6: Grobgranulare Architektur der Ressourcenmanagement-Lösung

von Internet-Anwendungen angepasst werden. So lässt sich eine Web-basierte Nutzeroberfläche für die in der Cloud befindliche Applikation bereitstellen. Allerdings ist eine Web-Rolle nicht für umfangreichere Prozesse praktikabel. Genau für diese Verarbeitung von umfassenden oder auch periodischen Vorgängen im Hintergrund ist die Worker-Rolle besser geeignet [123].

Die Architektur des Systems ist nach dem Vorschlag von Schwanengel et al. realisiert [157]. Der erste Schritt in Abbildung 5.7 veranschaulicht die Anmeldung zur Ressourcenbelegung des Clients per https-Request. Im Hintergrund legt die Web-Rolle diese Aufforderung im zweiten Schritt in einer Job Queue, also in der Warteschlange der zu bearbeitenden Aufgaben, ab.

Der dritte Schritt speichert die nötigen Informationen im internen Speicher. Den Auftrag zur Verarbeitung erhält die Worker-Rolle dann im vierten Schritt aus der Job Queue. Darauf holt sich in Schritt fünf die Worker-Rolle die Requests aus dem Speicher und archiviert die Parameter dort wieder.

Um reservierte Ressourcen einzufordern, schickt der Client eine Anfrage per https an die Web-Rolle. Diese zieht sich die bearbeiteten Werte aus dem Speicher und übergibt sie dem Client in Schritt sechs und sieben.

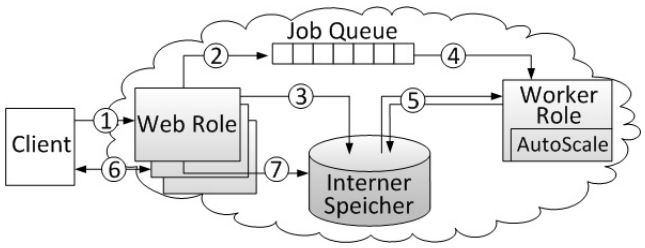


Abbildung 5.7: Umsetzung auf der Microsoft Azure Platform [157]

5.4 Mathematische Analyse des Protokolls

Zur Bestimmung der Güte des entworfenen Protokolls wird im Folgenden eine abstrakte mathematische Analyse durchgeführt. Grundsätzlich passt sich die Arbeitslast kaum der aktuellen Auslastung der Maschinen an. Um dennoch wenigstens ein gewisses Maß an flexibler und effizienter Instanzenverwaltung zu bieten, stellen die meisten Cloud-Anbieter verschiedene Ressourcenlevel zur Verfügung. Innerhalb eines Levels ist der Client bei der Ressourcennutzung an die dort fest definierte Menge virtueller Maschinen gebunden.

Das Bereitstellen zusätzlicher Kapazität ist nur durch die Anforderung eines höheren Levels mit einer größeren Anzahl an Ressourcen möglich. Da dies zu erheblichen Verzögerungen führt, ist eine Adaption an einen sich ändernden Instanzenbedarf in kürzester Zeit fast unmöglich. Bis das höhere Level die nötige Kapazität bereitstellt, steigt die Verwurfsrate der Anfragen innerhalb des aktuellen Ressourcenlevels auf einen inakzeptablen Umfang [156]. Diese Situation wird in Abbildung 5.8 visualisiert.

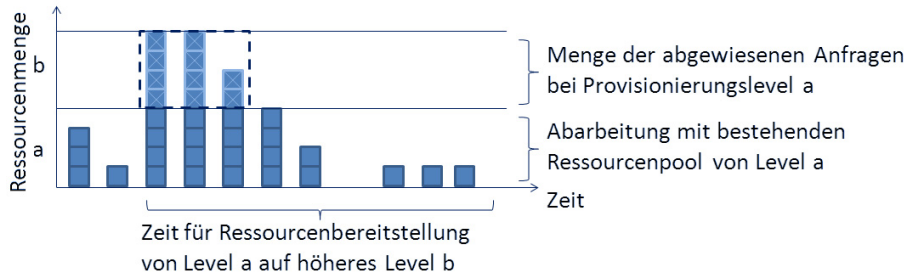


Abbildung 5.8: Ressourcenmanagement ohne Adaption

Nun wird ein Graph über die diskreten Werte der spezifischen Instanzenanzahl eines Ressourcenlevels gelegt. Durch daraus abgeleitete Formeln ist die Funktionstüchtigkeit des Protokolls mathematischen zu beweisen. Im Folgenden identifiziert das Symbol f indexabhängig unterschiedliche Funktionen. Hierbei kann die Gesamtlastfunktion f_{Last} mit Formel (5.1) zum Zeitpunkt t angegeben werden als Summe aller Requests der einzelnen Clients $j = \{1 \dots N\}$. Wie viele Clients pro Ressource behandelt werden können, ist durch die grundlegende Hardware bestimmt und festgelegt als Integer-Konstante ε. So beschreibt die Kapazitätsfunktion f_{Kap} in Formel (5.2), welches Lastvolumen im gesamten System mit $i = \{0 \dots M\}$ Instanzen verarbeitet werden kann.

$$f_{Last}(t) = \sum_{j=1}^{N} request_j(t) \tag{5.1}$$

$$f_{Kap}(t) = \sum_{i=0}^{M} \lceil \varepsilon \rceil_i \tag{5.2}$$

Pro Ressourcenlevel ist die verfügbare Kapazität als Konstante r_x gespeichert. Mit dieser Menge ist ein bestimmtes Lastvolumen zu bearbeiten. Die möglichen Fälle werden über Formel (5.3) abgebildet. $f_{Level}(t)$ stellt demnach eine Stufenfunktion dar, die sich an die variierenden Lastsituationen anpasst.
Falls die mit Formel (5.1) gegebene Anfragelast kleiner gleich der Ressourcenanzahl r_x ist, können alle Requests beantwortet werden. $f_{x_0}(t)$ drückt aus, dass keine weiteren Instanzen benötigt werden. Falls die Last je Ressource größer ist als der Wert r_x, reichen die bereitgestellten Ressourcen des Provisionierungslevels nicht aus, um alle Clients zu bedienen. Folglich springt die Funktion zum Ausdruck $f_{+x}(t)$. Wenn andererseits die angefragte Kapazitätenzahl unter den definierten Schwellwert ε fällt, kann das Ressourcenlevel zur Kosteneinsparung reduziert werden. Die Funktion $f_{-x}(t)$ verringert die Anzahl der Ressourcen.

$$f_{Level}(t) = \begin{cases} f_{+x}(t), & \text{wenn } r_x > \varepsilon \\ f_{x_0}(t), & \text{wenn } r_x \leq f_{Last}(t) \ \& \ r_x = \varepsilon \\ f_{-x}(t), & \text{wenn } r_x \ll \varepsilon \end{cases} \tag{5.3}$$

Um für den letzten Fall die Menge an Ressourcen im Idle-Modus zu minimieren, muss die Anzahl unterbeschäftigter Ressourcen berechnet werden. Mit den Formeln (5.1) und (5.2) wird dies über die Differenz zwischen der Summe aller bearbeitbarer Anfragen und der Gesamtlast ermittelt.

$$\begin{aligned} f_{\,Idle}(t) &= f_{\,Kap}(t) - f_{\,Last}(t) \\ &= \sum_{i=0}^{M} \lceil \varepsilon \rceil_i - \sum_{j=1}^{N} request_j(t) \end{aligned} \tag{5.4}$$

Ein negativer Ergebniswert dieser Formel signalisiert, dass alle Kapazitäten genutzt werden. Wenn der Ausdruck allerdings ein positives Resultat erzeugt, existieren unterbeschäftigte Instanzen. Dies erzeugt unnötige Kosten. Deshalb gilt die Formel (5.5) als Zielfunktion. Das dargestellte Integral muss gegen ein Minimum konvergieren, um die Auslastung der Ressourcen in der Cloud-Umgebung zu erhöhen und Kosten zu sparen.

$$\int_0^t f_{\,Idle}(t)\ dt \longrightarrow min \tag{5.5}$$

Die Gesamtauslastung des Systems zum Zeitpunkt t lässt sich ausdrücken mit Formel (5.6). Sie entspricht der Summe aller Last $f_{\,Last}(t)$ mit $k = \{0 \dots M\}$ Ausführschritten, die über die Stufenfunktion $f_{\,Level}(t)$ bestimmt sind. Die Verzögerung, die beim Sprung in ein höheres Ressourcenlevel aufgrund der Provisionierungsdauer weiterer Ressourcen entsteht, wird über den δ-Wert festgehalten. Als Vorbedingung gilt, dass der diesbezügliche Aufschub nicht größer sein darf als die per Service-Level-Agreement definierte akzeptierte Wartezeit des Clients. Es muss folglich $\delta < t_{Deadline}$ gelten.

$$f_{\,Sum}(t) = \sum_{k=0}^{M} f_{\,k}(t, f_{\,Level}(t), f_{\,Last}(t), \delta) \tag{5.6}$$

Während hoher Arbeitslast kann der Service-Load-Manager anhand des Protokolls Anfragen eine gewisse Zeit ans Ende der Abarbeitungsschlage sortieren. Dies erfolgt, solange die Service-Level-Agreements mit dem Nutzer eine Verzögerung erlauben. Dadurch können kurzzeitige Lastspitzen geglättet werden. Außerdem kann der Cloud-Anbieter innerhalb dieses gewonnenen Zeitintervalls weitere virtuelle Maschinen starten, falls die Kapazitäten auch mit dem Aufschub zum späteren Zeitpunkt nicht ausreichen.
Dieser Vorgang wird als Feedback vom Manager an den Client übermittelt. Der Nutzer muss sich also nicht fortlaufend über den Status seiner Abarbeitung erkundigen. Bis der Client über den erfolgreichen Instanzenstart informiert wird, kann er sich anderen Aufgaben widmen. Folglich kann mit dem Protokoll die

Zeit, bis zusätzliche Ressourcen bereitstehen, bestmöglich überbrückt werden. In Abbildung 5.9 ist dieser positive Einfluss des Feedbacks erkennbar.

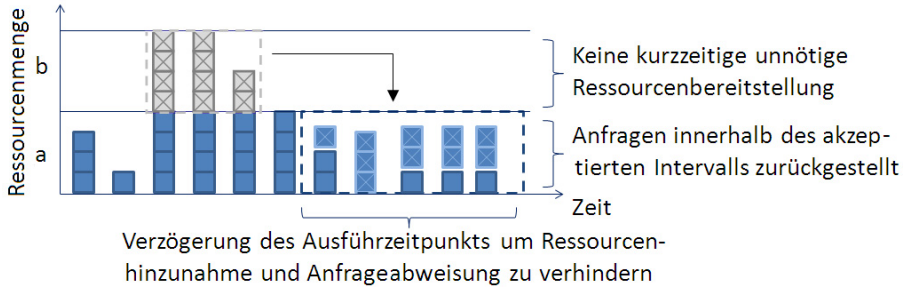


Abbildung 5.9: Ressourcenmanagement mit Adaption durch Verzögerung

5.5 Experimenteller Aufbau und Ergebnisse zum Protokoll

Um die vorangegangene mathematische Analyse mit realen Werten zu untermauern, werden verschiedene Szenarien mit Cloud-Dienstnutzung näher untersucht. Die Ergebnisse sind zum Teil in [157] publiziert.

Nach Ye [187] werden zur Bestimmung von realistischen Performanzwerten üblicherweise die drei Metriken *Timeliness*, *Precision* und *Accuracy* verwendet. Frei übersetzt bedeutet das 'Aktualität', 'Präzision' und 'Genauigkeit'. Faktisch meint *Timeliness*, wie schnell auf eine Eingabe eine Ausgabe erfolgt, wohingegen *Precision* die Quantität und *Accuracy* die Qualität der Ausgabe als Reaktion auf eine Eingabe misst.

Zeitverzögerung, Schwankungen und Antwortzeit dienen meist zur Festlegung der *Timeliness*. Messungen, die sich auf Bandbreite und Durchsatz beziehen, werden zum Attribut der *Precision* gezählt. Fehlerraten umschreiben die *Accuracy*-Metrik [188]. Bei der in dieser Arbeit entworfenen Testumgebung kann davon ausgegangen werden, dass Attribute der *Precision* wie die Bandbreite unveränderbar sind und das Protokoll hierauf keinen Einfluss hat.

Deshalb werden zur Ermittlung der Performanz des Protokolls in jedem Szenario die Attribute *Timeliness* und *Accuracy* analysiert. Genauer gesagt, wird für den *Timeliness*-Parameter die Antwortzeit als Messwert zur zeitlichen Aktualität betrachtet. Andererseits dient die *Accuracy* über die Fehlerrate, also die Anzahl abgelehnter Anfragen, als Messwert der Genauigkeit.

Im Folgenden werden drei Szenarien beleuchtet. Im ersten Szenario wird auf den Standardfall des periodischen gleichzeitigen File-Downloads vieler Clients eingegangen. Das zweite Szenario geht vom stetigen Herunterladen relativ kleiner Daten im Hintergrund aus. Dieser Vorgang ist bei vielen Smartphone-

Applikationen gängige Praxis. Der dritte Fall befasst sich mit ereignisgetriebener Nachrichtenverbreitung. Dies kommt aus Energiezwecken gewöhnlich in Sensorfeldern zum Tragen. Pro Szenario werden zwei Durchläufe je einmal mit und ohne Nutzung des Protokolls ausgeführt und die Messwerte verglichen.

5.5.1 Szenario 1: Periodischer File-Download

Als möglicher Anwendungsfall zur Evaluation des Ressourcenmanagement-Protokolls wird ein Szenario mit einem Service zur Viruserkennung untersucht. In einer täglichen Routine erhält der Diagnosedienst Updates für alle elektronischen Betriebsgeräte von großen Firmen mit bis zu 10.000 Angestellten. Die Größe der fiktiven Unternehmen reicht von mindestens 50 Mitarbeitern, über 100, 500, 1.000, 5.000, bis zu 10.000 Beschäftigten. Jeder Computer der Firmenangehörigen entspricht einem Client.

Im Szenarioablauf beginnen alle Clients die Download-Nachfrage über neue Sicherheitsaktualisierungen ihres Virus Scanners jeden Tag annähernd zum selben Zeitpunkt. Der Dienst erhält damit zeitgleich eine sehr große Menge Anfragen zur Prüfung auf neue Updates. Infolgedessen erreicht der bereitstellende Server schnell seine Grenzen und ist nicht in der Lage, rechtzeitig alle Anfragen zu beantworten.

Ohne Protokollnutzung werden die Anfragen sukzessive in einer Warteschlange eingereiht. Da die Clients in dieser Standardbearbeitung kein Feedback über diesen Vorgang erhalten, bleibt unklar, ob der Download-Request den Server überhaupt erreicht hat. Wenn beim Client innerhalb eines definierten Zeitintervalls keine Antwort eingeht, muss er vermuten, dass seine Nachfrage während des Transfers verloren gegangen ist und den Server nie erreicht hat. Häufig ist dies allerdings schlechtweg nicht zutreffend. Der Dienst ist lediglich zu überlastet, um in der ihm vorgegebenen Zeit alle Requests zu bedienen.

In diesem Fall fragt der Client erneut beim Server nach einem Download-Slot. Dieser Umstand resultiert in einer zusätzlichen Beanspruchung des Verbindungskanals. Daraufhin muss der Dienst prüfen, ob die Anfrage des Clients bereits eingegangen ist und noch in der Warteschlange liegt. In einfacheren Versionen solcher Dienste werden meist die überflüssigen duplizierten Anfragen kurzerhand ebenfalls in der Warteschlange eingereiht und schließlich doppelt bearbeitet. Dadurch läuft am Server unnötige Zusatzlast auf.

Wenn nun der Client den Download vor dem Ablauf der zuvor festgelegten Frist nicht erfolgreich beenden kann, wird lediglich eine knappe Fehlermeldung zurückgegeben. So hat der Client keine Kenntnis darüber, warum der Download fehlgeschlagen ist. Die Internet-Verbindung kann beispielsweise unterbrochen oder ein Computer-interner Fehler entstanden sein. Insgesamt sind ohne Protokollnutzung immense Latenzen, also lange Wartezeiten zu beobachten. Die Anzahl abgelehnter Anfragen steigt ebenfalls stark an. Folglich führt dieser Standardfall zu einem sehr unbefriedigenden Status der gesamten Systemfunktionalität. Das Protokoll verspricht dafür eine enorme Verbesserung.

Nach den Ergebnissen von Ye et al. [188] sind in Überlastsituationen verschieden lange Wartezeiten bis zur Beantwortung von Requests zu ermitteln. Die Größe der Anfragenachrichten liegt dabei bei 12 Mbit. Diese Zeitintervalle, bis die Gesamtbearbeitung erfolgt, reichen hierbei von 600 Sekunden bis zu 4.000 Sekunden, was zu einem Mittelwert von 36,6 Minuten Wartezeit führt.

Darauf aufbauend lassen sich ohne Nutzung des Protokolls die in Tabelle 5.1 erfassten Ergebnisse zu den Wartezeiten ableiten. Während für 50 Mitarbeiter die Bearbeitungsdauer für alle Anfragen noch bei 10 Minuten liegt, gibt es bei 500 Clients einen Anstieg auf bereits 30 Minuten und mit 1.000 Systemteilnehmern muss 40 Minuten gewartet werden. Die resultierenden Antwortzeiten reichen bis zu 67 Minuten bei 10.000 Mitarbeitern. Dies entspricht einer durchschnittlichen Wartezeit t_{warte} von 30,57 Minuten bei einer Standardabweichung σ von ca. 25 Minuten und einem Median m von 30 Minuten.

Mit dem in dieser Arbeit vorgestellten Protokoll lassen sich wesentlich bessere Antwortzeiten in der Anfragenbearbeitung ermitteln: für 50 Mitarbeiter liegt die Wartezeit bei 2 Minuten und mit 100 Angestellten bei nur 3 Minuten. Bei 500 Clients werden 6 Minuten und bei 1.000 Nutzern 10 Minuten gemessen. Die maximale Bearbeitungszeit aller Clients ergibt lediglich 18 Minuten bei 10.000 Anfragen. Bei einer Standardabweichung σ von 6,85 und einem Median m von 6 beträgt die Durchschnittswartezeit t_{warte} nur 7,71 Minuten.

Daraus wird deutlich, dass die Antwortzeiten für alle Clients im System bei der Nutzung des Protokolls circa viermal besser sind als ohne. Diese Differenz nimmt bei steigendem Anfragevolumen kontinuierlich zu. Abbildung 5.10 stellt die Ergebnisse mit und ohne Protokollabwicklung graphisch gegenüber.

Anzahl Clients	Antwortzeit ohne Protokoll (min)	Antwortzeit mit Protokoll (min)	Verworfene Anfragen ohne Protokoll	Verworfene Anfragen mit Protokoll
50	10	2	2	0
100	12	3	5	1
500	30	6	15	5
1.000	40	10	30	10
5.000	55	15	80	30
10.000	67	18	150	40
σ	24,89	6,85	55,82	16,17
m	30	6	15	5
$\varnothing t_{\text{warte}}$	30,57	7,71		
$\varnothing$#drops			40,29	12,29
$\varnothing$Verbes.	25 %		30 %	

Tabelle 5.1: Simulationsparameter Szenario 1 mit periodischem File-Download

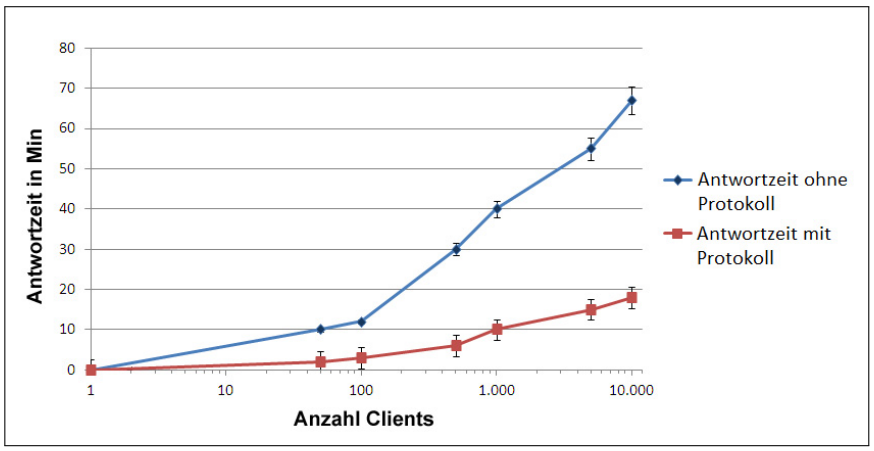


Abbildung 5.10: Antwortzeiten mit und ohne Protokoll beim File-Download

Die Standardumgebung ohne die Verwendung des Ressourcenmanagement-Protokolls ist nur in der Lage, eine bestimmte Menge an Anfragen zu verarbeiten. Bei Überschreitung des Schwellwerts müssen die überzähligen Requests in einer Warteschlange eingereiht werden. Effektiv müssen in diesem Szenario mit steigender Unternehmensgröße 30, 50, 350, 700, 4.200 und 8.500 Client-Anfragen zurückgestellt werden. Wenn nun das Volumen der wartenden Anfragen eine Zahl von 85 Prozent übersteigt, beginnt das System Anfragen abzulehnen, um seine Funktionsfähigkeit zu erhalten.

Abbildung 5.11 visualisiert die Verwurfsraten. Sie illustriert, dass das System bei 50 Clients 2 Anfragen verwirft. Bei wachsender Größe von 100 bis 5.000 Requests beginnt die Umgebung nacheinander 5, 15, 30 und 80 Anfragen abzulehnen. Mit 10.000 Nutzern beläuft sich die Verwurfsrate auf 150 Anfragen. Dies führt zu der Tatsache, dass ohne Nutzung des Ressourcenmanagement-

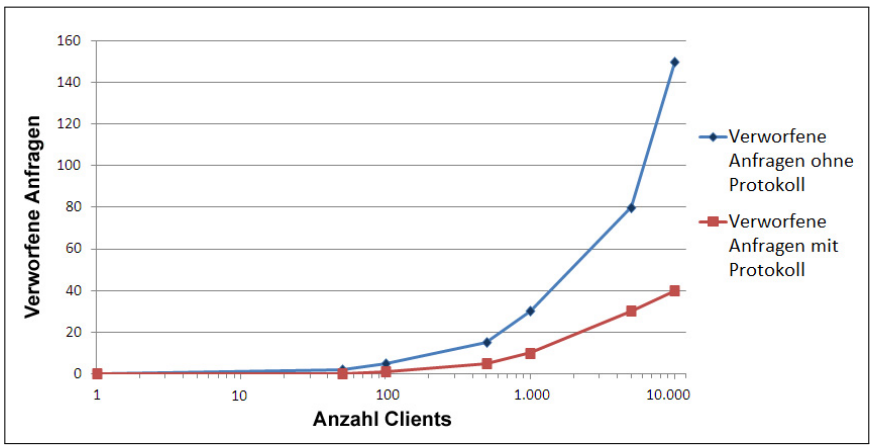


Abbildung 5.11: Verwurfsraten mit und ohne Protokoll beim File-Download

Protokolls die Anzahl der durchschnittlich verworfenen Anfragen $\varnothing\#drops$ bei 40 liegt mit einer Standardabweichung σ von 55,82 und einem Medianwert m von 15 Verwürfen. So steigt das Volumen an abgelehnten Requests bei steigender Client-Menge schon beinahe exponentiell.
Durch die Protokollimplementierung ergeben sich weitaus bessere Ergebnisse. Bei einer Menge von 50 Anfragen verwirft das System keine Aufforderung. Lediglich eine Anfrage wird bei insgesamt 100 gestellten Requests verworfen. Von 500 übermittelten Anfragen werden nur 5 verworfen. 10 Verwürfe gibt es bei 1.000 Anfragen, 30 bei einer Menge von 5.000 und lediglich 40 Anfragen bei einer Größe von 10.000. Um ein stabiles System aufrecht zu erhalten, müssen also durchschnittlich nur knapp 12 Anfragen verworfen werden bei einer Standardabweichung σ von 16,17 und einem Median m von 5.
Vergleicht man diese Werte mit den Ergebnissen für den Fall, dass das Protokoll nicht eingesetzt wird, lässt sich eine Verbesserung von ca. 30 % ableiten. In den Simulationsläufen wächst bei diesem Szenario die Menge verworfener Anfragen nur linear bei steigender Anzahl von Clients.

5.5.2 Szenario 2: Permanenter Daten-Download im Hintergrund

Das zweite Szenario entspricht dem heutigen Trend, dass ein sehr großer Teil der „Always-on“-Generation ein Smartphone besitzt und damit permanent im Internet online sein will [92]. Um den Nutzern einen unauffälligen und dennoch voll personalisierten Betrieb der Applikation auf diesen Endgeräten zu garantieren, benötigen die Anwendungen eine extreme Menge an Informationen. Dies impliziert das kontinuierliche periodische Herunterladen der notwendigen Daten, was durch die Applikationen als Hintergrund-Tätigkeit durchgeführt wird. Der Datenumfang solcher Downloads bewegt sich meist im unteren Megabyte-Bereich um 1 MB [93]. Üblicherweise können diese Zeitintervalle sehr flexibel angepasst werden, wenn die angesprochene Ressource aktuell an einen anderen Nutzerinteraktionsprozess gebunden ist.
Dieser Anwendungsfall befasst sich mit Großstädten, die unterschiedliche Einwohnerzahlen aufweisen. Dabei wird die steigende Anzahl von 1.000, 5.000, 10.000, 100.000, 1.000.000 und 3.000.000 Smartphones angenommen. Dies ist trotz der Größe eine realistische Menge, betrachtet man den Umstand, dass heutzutage Geschäftsleute oft zwei oder gar mehr Smartphones für den privaten Gebrauch und für Geschäftszwecke besitzen.
In Tabelle 5.2 sind die Ergebnisse zu den Gesamtantwortzeiten und den Verwurfsraten des Szenarios mit permanentem Download überblicksartig dargestellt. Zu verzeichnen ist, dass die Bearbeitungszeit, bis alle Anfragen im System abgehandelt sind, bei 1.000 Clients ohne Verwendung des Protokolls einen hohen Wert von 20 Minuten aufweist.

Anzahl Clients	Antwortzeit ohne Protokoll (min)	Antwortzeit mit Protokoll (min)	Verworfene Anfragen ohne Protokoll	Verworfene Anfragen mit Protokoll
1.000	20	10	6	2
5.000	32	13	20	5
10.000	50	15	30	7
100.000	65	25	50	15
1.000.000	78	30	65	20
3.000.000	90	37	75	25
σ	32,58	12,74	28,99	9,54
m	50	15	30	7
$\varnothing t_{\text{warte}}$	47,57	18,57		
$\varnothing$#drops			35,14	10,57
$\varnothing$Verbes.	39 %		30 %	

Tabelle 5.2: Simulationsparameter bei permanentem Daten-Download

Wenn das Protokoll verwendet wird, halbiert sich die Antwortzeit auf 10 Minuten. Die Systemantwortzeit aller Anfragen steigt für 5.000 Clients auf 32 Minuten ohne Protokoll und auf nur 13 Minuten bei seiner Verwendung.
Im aufgezeichneten Kurvenverlauf der Antwortzeiten in Abbildung 5.12 ist ein klarer Bruch im Falle der Nichtnutzung des Protokolls zu erkennen. Da das für die Simulationen verwendete Rechenzentrum die steigende Anfragemenge mit den dort bereitgestellten Kapazitäten nicht mehr bewältigen konnte, musste ab einer Request-Menge von 10.000 ein weiteres Rechenzentrum angefragt werden. Das damit nötige Aufsetzen der neuen Umgebung und die verlängerte Routing-Strecke der Anfragen verursachen eine zusätzliche Verzögerung. Das erklärt

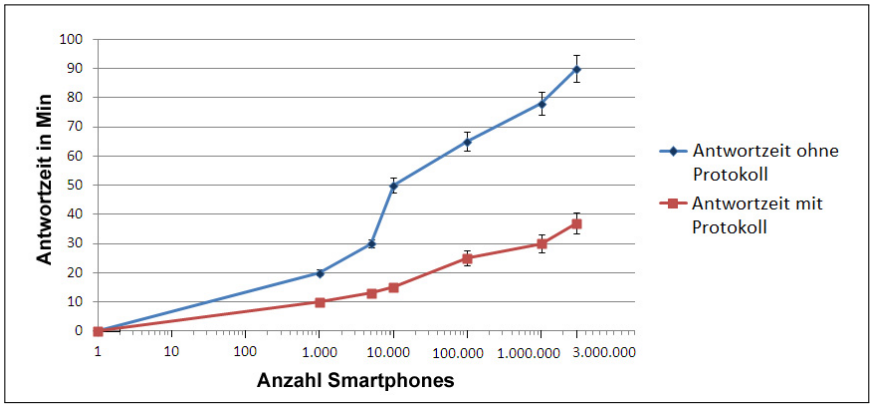


Abbildung 5.12: Antwortzeiten in Szenario 2 mit und ohne Protokoll

den Knick in den Graphen. Ohne Protokollnutzung werden die Messwerte bei dieser Menge sehr unbefriedigend, da die Antwortzeit 50 Minuten beträgt.
Bei 100.000 Endgeräten wird bei der Bearbeitung bereits die Stundenmarke überschritten. Dieser Wert steigt drastisch weiter bis zu einer gesamten Abarbeitungsdauer von 90 Minuten bei 3.000.000 Systemnutzern. Diese Ergebnisse ohne Protokollnutzung führen zu einer durchschnittlichen Wartezeit von 47,57 Minuten bis alle Clients bearbeitet sind mit der Standardabweichung von 32 Minuten und einem Median von 50 Minuten.
Dem steht für 100.000 Anfragen bei Verwendung des Protokolls eine Antwortzeit von nur 25 Minuten gegenüber bis alle Requests behandelt sind. Auch ist der Anstieg der gesamten Bearbeitungszeit auf 37 Minuten mit wachsender Anfragemenge bis 3.000.000 Requests relativ gering. Insgesamt beläuft sich die Antwortzeit bei Protokollverwendung auf lediglich 18,57 Minuten im Schnitt mit einer Standardabweichung von etwas über 12 Minuten und einem Median von 15. Die durchschnittliche Bearbeitungszeit aller Anfragen ist in dem protokollimplementierenden System um fast 40 % reduziert.
Ein ähnliches Resultat wird bei der Betrachtung der jeweiligen Verwurfsraten des Szenarios erkennbar. Sie sind in Abbildung 5.13 erfasst. Die Vorteile des Protokolls treten bereits ab einer Anzahl von 1.000 Nutzern klar zu Tage. Wenn das Protokoll nicht genutzt wird, steigt ab dem dortigen Wert von 6 Request-Ablehnungen die Rate auf 30 Verwürfe bei 10.000 Smartphones und auf bis zu 75 Anfragenverwürfe bei 3.000.000 Clients.
Der Reihe nach werden 630, 3.000, 6.350, 77.700, 800.000 und 2.500.000 Requests in der Warteschlange zur Wiederbearbeitung eingereiht. Durchschnittlich erfolgen in diesem Anwendungsfall über 35 Anfragenverwürfe.
Im Fall mit Protokollvermittlung werden bei 1.000 Clients nur 2 Anfragen verworfen. Dieser Wert steigt mit wachsender Nutzeranzahl nur sehr langsam auf 5, 7, 15 und 20 Verwürfe an. Bei 3.000.000 Nutzern im System werden

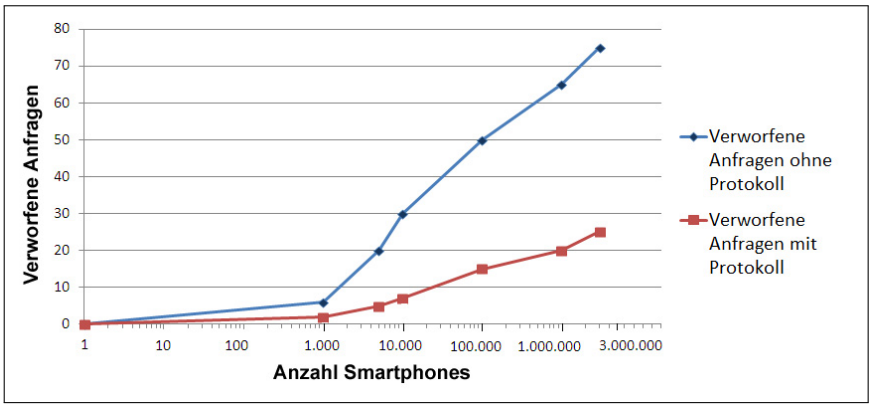


Abbildung 5.13: Verwurfsraten in Szenario 2 mit und ohne Protokoll

lediglich 25 Anfragen verworfen. Somit wird mit der Nutzung des Protokolls eine Verbesserung der Verwurfsrate von 30 Prozent realisierbar.
In diesem Anwendungsfall bietet das vorgestellte Protokoll eine enorme Leistungssteigerung hinsichtlich der Antwortzeiten und Verwurfszahlen, da die Datenaktualisierung im Hintergrund erfolgt und nicht zeitkritisch ist. Damit ist die Bedienung der Clients nicht an ein starres Zeitintervall gebunden. Der Service-Load-Manager kann so den Bedienzeitpunkt der Anfragen besonders effizient in der Abarbeitung einplanen. Die frühzeitige Reservierung reduziert folglich unnötige Wartezeiten und hohe Verwurfsraten für die parallel arbeitenden Clients erheblich.

5.5.3 Szenario 3: Ereignisgetriebene Nachrichtenverbreitung

Das dritte Szenario befasst sich mit einer Sensornetzwerk-Umgebung, in der sehr kleine Nachrichten nur dann verbreitet werden, wenn ein Ereignis erkannt wird. Diese Vorkommnisse sind kaum vorhersagbar. Zudem variiert die benötigte Anzahl an Ressourcen enorm. Sobald ein nahe gelegener Sensor ein Ereignis detektiert, produziert er eine entsprechende Nachricht. Diese Information wird synchron von Sensorknoten zu Sensorknoten in Richtung der Kontrollstation weitergeleitet [85]. Tabelle 5.3 umfasst die Ergebnisse der ereignisgetriebenen Nachrichtenverbreitung in Sensornetzen.
Aufbauend auf den Beobachtungen und Ergebnissen von Gandham et al. [64] betrachten die nun beschriebenen Simulationen ein Sensorfeld. Die Größe beträgt 30 mal 30 Meter. Je Simulationsdurchlauf werden der Reihe nach 30, 50, 100 und schließlich 200 Knoten im Feld platziert. Drei zufällig angeordnete Kontrollstationen verarbeiten die eingehenden Nachrichten der Sensoren.

Anzahl Clients	Antwortzeit ohne Protokoll (min)	Antwortzeit mit Protokoll (min)	Verworfene Anfragen ohne Protokoll	Verworfene Anfragen mit Protokoll
30	2	1	5	2
50	5	2	8	3
100	12	6	15	8
200	15	8	20	10
σ	6,46	3,44	7,96	4,22
m	5	2	8	3
$\varnothing t_{warte}$	6,8	3,4		
$\varnothing \#drops$			9,6	4,6
$\varnothing$Verbes.	50 %		48 %	

Tabelle 5.3: Simulationswerte bei ereignisgetriebener Nachrichtenverbreitung

Aus Gründen der Energieeffizienz und der Kostenersparnis wird in Sensorknoten häufig keine Komponente verbaut, die eine Ablauflogik implementiert. Da nur die reine Information ohne jegliche Interpretation verschickt wird, ist die Nachricht sehr klein. Konkret wird die Nachrichtengröße auf 200 bit gesetzt. Weiterhin wird das Zeitlimit, bis eine Instanz bedient werden muss, auf vier Minuten gesetzt. Bei Überschreitung des Schwellwertes wird der Sensorknoten zur Verlängerung seiner Lebensdauer heruntergefahren.

Wenn das Protokoll nicht verwendet wird, ergeben sich bei 30 Sensoren ca. 2 Minuten Antwortzeit, bis alle Nachrichten von den Basisstationen bearbeitet sind. Mit 50 Knoten beträgt die Verarbeitung aller Clients 5 Minuten.

Auch mit steigender Knotenanzahl im Sensorfeld werden selbst ohne Nutzung des Protokolls die notwendigen Wartezeiten bei der Bearbeitung der Sensornachrichten nicht viel größer. So besteht bei 100 Knoten eine 12-minütige Wartezeit. Bei 200 Sensoren ergibt sich eine 15-minütige Antwortzeit. Dabei ist eine durchschnittliche Wartezeit $\varnothing t_{warte}$ von 6,8 Minuten bei einer Standardabweichung σ von ca. 6 Minuten und einem Median m von 5 Minuten zu beobachten.

Im Gegensatz dazu zeigt sich, sobald das Protokoll eingeschaltet ist, dass 30 Knoten nur 1 Minute und 50 Knoten 2 Minuten warten müssen. Bei 100 Knoten erhöht sich die Antwortzeit auf 6 Minuten und bei 200 Knoten auf 8 Minuten. Die durchschnittliche Wartezeit $\varnothing t_{warte}$ beträgt 3,4 Minuten mit einer Standardabweichung σ von ca. 3 Minuten und einem Median m von 2 Minuten. Abbildung 5.14 visualisiert diese Ergebnisse für die unterschiedliche Anzahl an Knoten im Sensornetzwerk, indem es die Fälle der Verwendung des Protokolls und seiner Nichtnutzung gegenüberstellt.

Insgesamt ist erkennbar, dass bei Nichtverwendung des Protokolls die Antwortzeiten bei steigender Knotenanzahl auch im Szenario des Sensornetzwerks immer schlechter werden. Allerdings passen sich die Ergebnisse bei einer Feldgröße von ca. 50 Knoten relativ an. Zudem ist das Ausmaß weniger signifikant im Vergleich zu den Messungen der vorherigen Anwendungsfälle. Das liegt vor

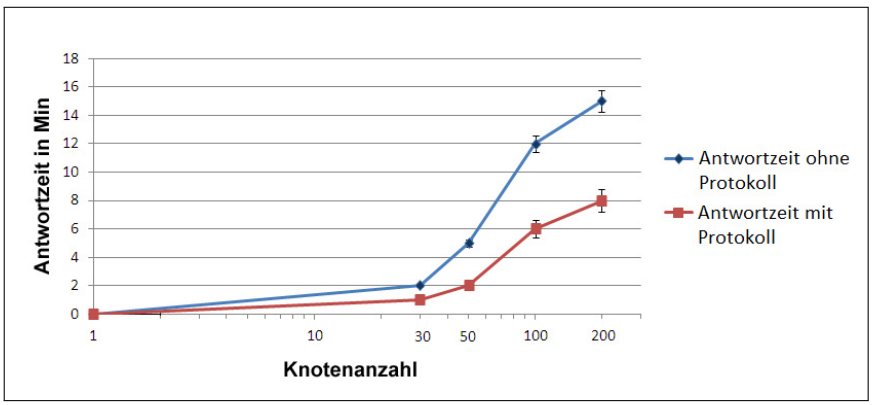


Abbildung 5.14: Antwortzeiten in Sensornetzen mit und ohne Protokoll

allem am Umstand der begrenzten Aktivzeit der Sensoren durch ihr automatisches Herunterfahren nach vier Minuten.
Die Verwurfsrate ist in Abbildung 5.15 ersichtlich. Sie demonstriert eine Differenz zwischen den Abläufen mit Protokollunterstützung und denen, wenn das Protokoll nicht genutzt wird. Wie in der Grafik repräsentiert, ist die Menge an abgelehnten Anfragen im Fall ohne die Verwendung des Protokolls fast doppelt so hoch wie mit.
Nichtsdestotrotz ergibt sich auch im Systemprozess ohne das Protokoll eine relativ geringe Anzahl an Verwürfen. Ursächlich dafür ist der begrenzte Radius der Sensoren. Lediglich Knoten in direkter Nähe eines Ereignisses erkennen die Veränderung und kommunizieren sie im Netz von Sensor zu Sensor weiter zur Kontrollstation. Infolgedessen ist die Menge an sendenden Sensorknoten im Durchschnitt meist relativ gering.
Der Reihe nach werden 18, 30, 70 und 150 Anfragen in die Warteschlange eingereiht. Ohne Protokoll werden dann bei 30 Knoten 5 Anfragen verworfen. Bei 50 Knoten gibt es 8 Verwürfe, bei 100 Knoten finden sich 15 Verwürfe und bei 200 Knoten entstehen 20 Verwürfe. Dabei zeichnet sich eine durchschnittliche Verwurfsrate $\varnothing\#drops$ von 9,6 ab mit einer Standardabweichung σ von ca. 8 und einem Median m von 4 abgelehnten Anfragen.
Bei Benutzung des Protokolls werden dagegen mit 30 Knoten nur 2 Anfragen und mit 50 Sensoren nur 3 Requests verworfen. Wenn das Netzwerk 100 Knoten verwaltet, ergeben sich 8 Verwürfe. Bei 200 Knoten entstehen 10 Verwürfe. Die durchschnittliche Verwurfsrate $\varnothing\#drops$ liegt bei nur 4,6 mit der Standardabweichung σ von 4 und dem Median m von 3. Für diesen Anwendungsfall wird mit dem Protokoll eine fast 50-prozentige Reduzierung der Verarbeitungszeit und der Anzahl an Verwürfen erreicht.
Basierend auf diesen Messungen ist in allen Simulationen eine signifikante Verbesserung der Antwortzeiten und der Verwurfsraten im System zu erkennen.

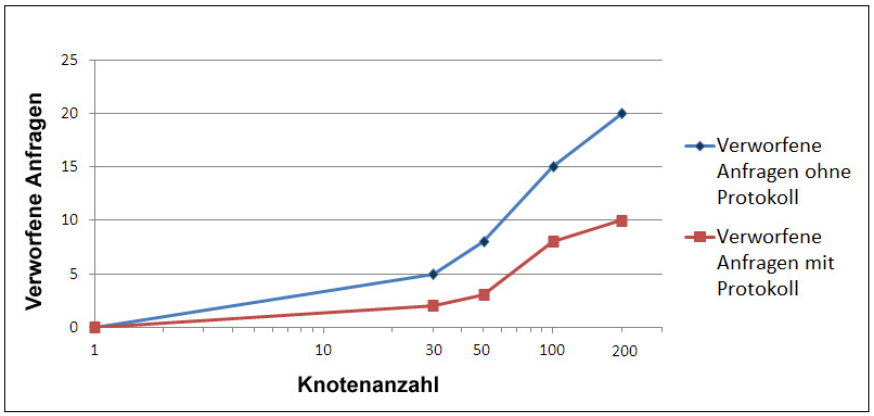


Abbildung 5.15: Verwurfsraten in Sensornetzen mit und ohne Protokoll

5.6 Zusammenfassung

In diesem Kapitel wird deutlich, dass ein Ressourcenmanagement-System im Cloud Computing unterschiedliches Dienstverhalten bewältigen muss. Problematisch ist dabei meist die präzise Vorhersage der Service-Aufforderung beim Dienstanbieter. Prognosen über die Menge und den Zeitpunkt der eigentlichen Client-Anfragen erzeugen im Endeffekt häufig mehr Last als schließlich die tatsächliche Dienstbeanspruchung. Hier besteht ein enormer Bedarf an einer nicht-komplexen schnellen Lösung.

Durch die Reservierung und das Feedback über den entwickelten Protokollablauf ist es möglich, ein solches leichtgewichtiges Ressourcenmanagement zu realisieren. Die Protokollbestandteile sind im Informations-, Kommunikations- und Aktivitätsmodell klar definiert. Darüber ist festgelegt, wie welche Informationen ausgetauscht werden und wie sich der Ablauf in und zwischen den beteiligten Komponenten verhält.

Mithilfe des Protokolls kann der Dienst den Client in der eigentlichen Service-Nutzung beeinflussen, indem die Anfragen bidirektional und vereinigt abgehandelt werden. Durch die Verzögerung des Protokolls können einige Skalierungsprozesse vermieden werden. Die Provisionierung aller Nutzer kann mit konstanterer Ressourcenmenge erfolgen. Infolgedessen sind Lastspitzen ohne hohe Varianzen ausgleichbar. Die Kapazitätenauslastung ist so optimierbar. Ein Vorteil der somit reduzierten Anzahl an Ressourcen sind die minimierten Kosten auf Service-Seite bei der Diensterbringung.

Aus der Evaluation des Protokolls zum Cloud-Ressourcenmanagement wird ersichtlich, dass basierend auf der Reservierung durch den Client selbst und dem Feedback durch den Service-Load-Manager die Performanzwerte gesteigert werden. Aufgrund der Zeiteinsparungen bei der Reduzierung der Ressourcenallokation werden bessere Antwortzeiten erzielt. Außerdem sinkt mit dem Feedback die Anzahl der Timeouts auf Client-Seite, was zu geringeren Werten in der Verwurfsrate führt.

6 Proaktives automatisches Instanzenmanagement abhängiger Ressourcen

Im standardmäßigen Cloud-Ressourcenmanagement erfolgt die Festlegung der Menge zu startender und stoppender Maschinen über vorkonfigurierte Schwellwerte und primitive Metriken. Steigen Größe und Komplexität der Systeme, wird eine wachsende Zahl an Regeln benötigt. Sind zu viele Kontrollwerte im Management definiert, entstehen schnell Konflikte. Dann löst eine Regelausführung ein Ereignis aus, das wiederum den nächsten Punkt triggert. Das führt zu einer nicht-überwachbaren Handlungskaskade.

Zudem weisen Cloud-Ressourcen diffizile Abhängigkeiten untereinander auf. Ursächlich dafür ist zum einen der vielschichtige Aufbau der zugrundeliegenden Infrastruktur. Dabei werden häufig verschiedene Services der gleichen Ebene parallel genutzt. Andererseits erhöht sich die Unübersichtlichkeit durch die Vereinigung mehrerer einzelner Komponenten zu einem ineinandergreifenden Gesamtsystem [192]. Zur Service-Erbringung kommen explizit External-Facing-Ressourcen, wie Web-Server, und intern abhängige Instanzen zum Einsatz. Zusätzlich werden die Maschinen zwischen den Diensten geteilt [33].

Meist sind diese Abhängigkeiten bei Systemanpassungen nicht einkalkuliert. Um Konflikte durch übereiltes Herunterfahren von unausgelasteten, für das Scheduling essentiellen Ressourcen zu vermeiden, verwenden gängige Skalierungsmethoden zurückhaltende, kostenintensive Strategien. Analog werden bei minimaler Schwellwertüberschreitung voreilig Instanzen zugeschaltet.

Zur Lösung des Problems wird das entwickelte Protokoll erweitert. So ist eine proaktive automatische Verwaltung von abhängigen Cloud-Ressourcen möglich. Zunächst wird auf verwandte Arbeiten zu Skalierung und Scheduling eingegangen. Des Weiteren wird die grundlegende Systemstruktur basierend auf der Lastspitzen-Glättung durch das Protokoll erklärt.

Ein Überblick zu dem folgenden Algorithmus ist in [158] publiziert. Die Methodenumsetzung findet mithilfe eines Selbstkalibrierungsmechanismus vor dem Regelbetrieb statt. Darauf aufbauend führt der Service-Load-Manager während der Komponentenaktivitäten ein Ressourcenregelverfahren durch, das einen optimierten Scheduling-Ablauf realisiert. Die Evaluationsergebnisse der abstrakten Analyse und der Simulationen zeigen, dass sich mit diesem Ansatz die Lastverläufe automatisch überwachen lassen. So kann sich das Cloud-System selbstständig auf ein Auslastungsoptimum skalieren.

6.1 Verwandte Arbeiten zu Skalierung und Scheduling bei abhängigen Ressourcen

Der folgende Abschnitt widmet sich wissenschaftlichen Erkenntnissen des Lastmanagements unter Berücksichtigung von Ressourcenabhängigkeiten. Die bereits in Kapitel 3 beschriebenen Arbeiten beziehen beispielsweise Signaturen, vordefinierte Ausführreihenfolgen oder Plangenerierung in die Kapazitätenverwaltung ein [22, 32, 70]. Dabei wird allerdings enormes Potenzial für Kosteneinsparungen und Performanzgewinne vergeudet. Zudem findet die Aushandlung und Verwirklichung von SLAs meist unzureichende Beachtung. Folglich ist eine tiefer gehende Untersuchung dieses Forschungsfelds nötig.

Zur Bestimmung einer performanten Anzahl erforderlicher virtueller Maschinen definieren Cardosa et al. [30] sogenannte Min-, Max- und Share-Parameter. Der Min-Parameter stellt sicher, dass die Ressourcen beim Start mindestens diese Anzahl an Instanzen aus der verteilten Umgebung zugewiesen bekommen. Dagegen legt der Max-Parameter fest, dass eine Maschine nicht mehr virtuelle und physikalische Kapazität als die darin präzisierte Menge erhält. Der Share-Parameter bezieht sich auf den Pool frei verfügbarer Instanzen. Dieser Wert erleichtert dem Scheduler die Festlegung der Anfragenverteilung auf die Maschinen. In diesem Umfeld bleiben jedoch die Beziehungen zwischen den beteiligten Systemkomponenten unbeachtet.

Der von Dornemann et al. [46] vorgeschlagene Ansatz ermöglicht ebenfalls die Anpassung des Instanzenumfangs. Dabei werden jedoch weder Abhängigkeiten noch SLAs zwischen den Ressourcen berücksichtigt. Die Architektur skaliert zwar bei Lastspitzen für Cloud-Arbeitsabläufe. Doch führt die Unkenntnis über die gegenseitigen Beeinflussungen zu ungleichmäßiger Auslastung der genutzten Maschinen und Performanzeinbußen.

Folglich stellt die Identifikation von Abhängigkeiten zwischen den zu bearbeitenden Aufgaben eine besondere Herausforderung dar. Zu Zwecken der Scheduling-Optimierung ist dies seit langer Zeit Forschungsgegenstand. Meist wird auf die mathematische Grundlage der gerichteten azyklischen Graphen gesetzt. Sie dienen der Modellierung von Beziehungen unter den Teilkomponenten [95]. Die Graphen müssen allerdings häufig von zentraler Stelle erstellt werden, da nur so der Gesamtüberblick über das System besteht. Diese Vorgabe durch einen solchen Primärknoten behindert wiederum die Elastizität und Umsetzung schnelllebiger Änderungen der Cloud-Umgebung.

Takahashi et al. [169] referenzieren diesbezüglich Probleme, die in den vielschichtigen Umgebungen des Cloud Computings auftreten. Sie konzentrieren sich auf Schwierigkeiten, die durch den Ausfall einer von anderen Komponenten genutzten Instanz bedingt sind. Dafür erstellen sie ähnlich zu dem hier verfolgten Ansatz Abhängigkeitsgraphen. Allerdings bieten die Autoren ein höchst unflexibles Modul der Fehlertoleranz, da zur Kontrolle und Organisation der Ressourcen und ihrer Abhängigkeiten ein Administrator nötig ist.

Das ‘SWAP’-System von Zheng et al. [193] setzt ebenfalls auf die automatische Erkennung von Abhängigkeiten. Um entdeckte Interferenzen zwischen verschiedenen Prozessen im Ausführungsplan zu berücksichtigen, wird der aufgezeichnete Systemverlauf herangezogen. Die Autoren beschränken den Scheduler jedoch auf eine zuvor festgelegte Menge an Ressourcen. Dadurch sind Skalierungsmechanismen nicht umsetzbar.

Daneben eignen sich die von Malawski et al. [103] analysierten Algorithmen zur optimalen Ressourcenbereitstellung für a priori bekannte und vielfältige Aufgabenstellungen. Gleichwohl wird wiederum die Dynamik des sich ständig ändernden Ressourcenvolumens im Cloud Computing nicht beachtet.

Die von Mao et al. [104] entwickelte automatisch skalierende Scheduling-Methode verwirklicht den Ansatz, eingeordnete Jobs innerhalb einer vorgegebenen Zeitspanne zu beenden. Trotz der so ermöglichten Kostenreduktion müssen durchgehend die Laständerungen beobachtet werden. Zudem können in der Testumgebung der Autoren nach der Definition eines Ausführplans keine weiteren Aufgaben aufgenommen werden.

Weitere Arbeiten, wie der Feedback-basierte Ansatz von Steere et al. [168] und die von Dong et al. [45] vorgestellten Scheduling-Mechanismen des Grid Computings, konzentrieren sich dabei überwiegend auf Aspekte der Fehlertoleranz im Falle von Serverausfällen. Im Grid-Umfeld forschen auch Carretero et al. [2], um effiziente Ausführalgorithmen zu konzipieren. Sie stellen eine ausführliche analytische Studie zu genetischen Algorithmen vor. Der so entworfene Scheduler ist jedoch noch nicht in der nötigen dynamischen Form umgesetzt.

Wu et al. [181] adaptieren die Strategie des *Random*-Schedulings für Cloud-Umgebungen. Diese hierarchische Verteilungsmethode berücksichtigt statische Service-Levels, um die dynamische Zuordnung zufällig ausgewählter Anfragen auf freie Kapazität durchzuführen. Das soll Abarbeitungskosten minimieren. Hierbei kommen ebenfalls gerichtete azyklische Graphen zum Einsatz. Allerdings ist der Aufwand dieses Erstellungsmechanismus zu massiv, um tatsächlich effiziente Ressourcenverwaltung umsetzen zu können.

Nachteilig ist durchwegs die eingeschränkte Ressourcenverwaltung der herkömmlichen Management-Systeme. Die Cloud-Instanzen werden von den marktführenden Anbietern, wie Amazon, Microsoft und Google, in verschiedenen Typen angeboten. Die Kategorien stellen stufenweise unterschiedlich mächtige Kapazitäten bereit. Sie reichen beispielsweise von sehr schwacher Ausprägung mit geteiltem Kern und wenig Arbeitsspeicher bis zu extra großem Umfang mit acht eigenen Kernen und umfassendem Arbeitsspeicher. Bei Lastanstiegen ist es allerdings lediglich möglich, Ressourcen des gleichen Typs zu starten. Ist eine Architektur für einen Instanzentyp mittlerer Größe ausgelegt, werden folglich auch bei minimalem Lastzuwachs, wo kleine günstigere Komponenten ausreichen würden, nur mittelgroße Maschinen gestartet [122].

Hinzu kommt, dass die definierten Regeln zur Skalierung in der Praxis statisch sind. Dadurch werden Modifikationen und Anpassungen des Systems außerordentlich aufwändig. Bei Veränderungen der Ressourcenauslastung ist der

gesamte Cloud-Dienst vom Netz abzukoppeln. Anschließend müssen die Skalierungsdefinitionen entsprechend der Neuerungen reprogrammiert werden [143]. Das offenbart die enorme Bedeutung der Selbstregulierung ohne tief gehende Systemkenntnisse für ein effizientes Cloud-Ressourcenmanagement.
Insgesamt benötigen die meisten Lösungen einen Administrator, der alle Ressourcen und ihre Abhängigkeiten überwacht. Die so vorherrschende Management-Grundlage ist nicht ausreichend adaptiv und die einzelnen Instanzenzustände bleiben unberücksichtigt. Ein Mechanismus zur vollkommen automatischen Instanzenskalierung in Cloud-Systemen existiert insofern noch nicht. Die Herausforderung besteht also darin, mit geringem Aufwand die voneinander abhängenden Cloud-Services günstig zu verwalten und gleichzeitig die ausgehandelten Service-Level-Agreements umzusetzen. Im Rahmen dieser Arbeit wird eine Methode entwickelt, die ein solches automatisiert skalierendes Maschinenumfeld mit einem optimalen Scheduling-Ablauf realisiert.

6.2 Lastspitzen-Glättung per Ressourcenmanagement-Protokoll

Eine Erweiterung des Protokolls umfasst die Maßnahme, den Instanzenbedarf für abhängige Komponenten der Cloud-Umgebung zu berücksichtigen. Dabei heißt ein Service A abhängig von Service B, wenn er auf dessen Ausführung angewiesen ist. Basierend auf dem bereits beschriebenen Verfahren vermittelt der Service-Load-Manager nach wie vor zwischen den Cloud-Services und ihren Clients. Dies dient der besseren Auslastung des bereitgestellten Ressourcenvolumens. Der Protokollablauf sieht vor, dass die Nutzer weiterhin ihren Bedarf reservieren. Durch die Aggregation aller eingehenden Anfragen kann der Manager ein zu Abbildung 6.1 ähnliches Lastgebirge schaffen.
Durch das Ressourcenmanagement-Protokoll werden gewisse Requests aufgeschoben, um Überschreitungen der verfügbaren Kapazität und damit verbundene Anfragenverwürfe zu vermeiden. Aufgrund dieser Verzögerung von einwilligenden Clients ist eine geringere Zahl unausgelasteter Maschinen zu erreichen.

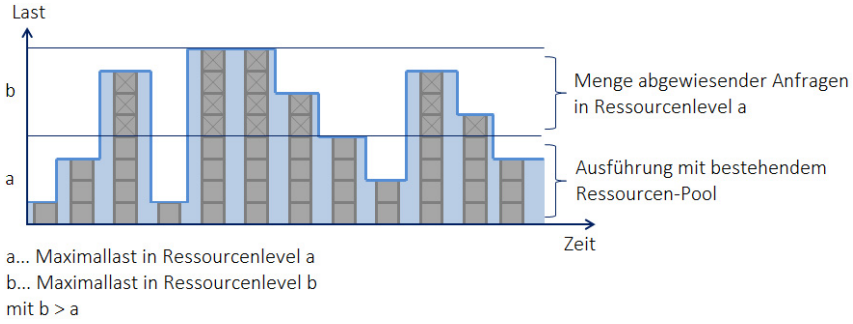


Abbildung 6.1: Lastgebirge aggregierter Client-Anfragen ohne Verzögerung

Hinzu kommt, dass zur Deckung des gesamten Nutzerbedarfs seltener skaliert werden muss. Infolgedessen sind Lastspitzen zu glätten. Gleichzeitig ist eine höhere Systemauslastung erreichbar – siehe Abbildung 6.2.

Offensichtlich kann jedoch nicht auf alle Skalierungsprozesse verzichtet werden. Damit in diesem Fall der weitere Verlauf schnellstmöglich abgewickelt wird, ist eine maschinell autonome Anpassung der Systemgröße erforderlich. Für die automatisierte Entscheidung darüber, wie viele und vor allem welche virtuellen Ressourcen hoch- oder heruntergefahren werden sollen, benötigt das System Wissen über die Abhängigkeiten unter ihnen. Genauer gesagt ist die Bestimmung der Verhältnisse der laufenden Maschinen zwischen den einzelnen Ebenen notwendig. Daraufhin können bei Laständerungen die Instanzen eines Dienstes mit ihren abhängigen Ressourcen als Einheit skaliert werden, ohne die Auslastung jeder spezifischen Komponente überwachen zu müssen.

Um das nötige Abhängigkeitsmodell zu erstellen, wird zuerst global ermittelt, welche Services mit welchen anderen in Verbindung stehen. Dafür teilt jeder Dienst dem Service-Load-Manager bei der Registrierung seine Relationen zu anderen Komponenten mit. Diese Verbindungen speichert der Service-Load-Manager, wobei er für jeden Service bestimmte Attributwerte belegt. Zum einen wird die übermittelte minimale und maximale Instanzenanzahl pro skalierbarer Ressource als Parameter *Min/Max # Instanzen* festgehalten. Andererseits wird über die Werte *Inst. Start-/Stoppzeit* die Zeitspanne erfasst, die der Service zum Starten und Stoppen neuer Instanzen benötigt.

Die folgende vereinfachte Schilderung dient dem leichteren Verständnis der typischen vielschichtigen und komplexen Service-Architekturen des Cloud-Umfelds. Man stelle sich eine Dienstumgebung vor, die aus Instanzen einiger Web- und Worker-Rollen besteht. Zur persistenten Speicherung von Informationen werden Datenbank-Rollen eingesetzt. Komponenten desselben Typs können in Ressourcengruppen zusammengefasst werden. Faktisch erfolgt die Instanzenskalierung über die Replikation der einzelnen Rollen.

Ein Beispielaufbau umfasst zwei anfragende Web-Rollen, zwei abhängige Worker-Rollen und eine geteilte Datenbank. Die Umgebung ist in Abbildung

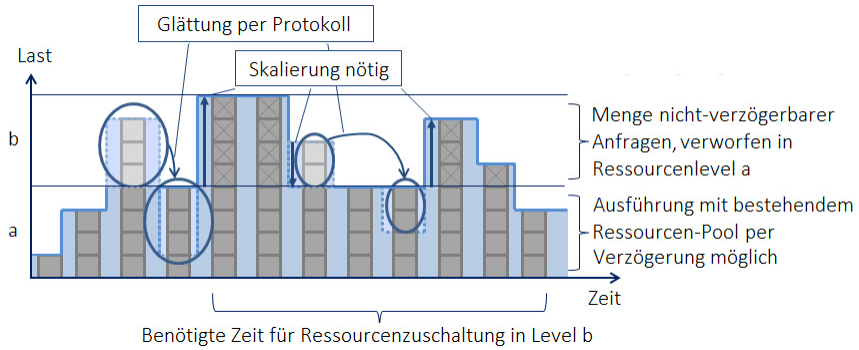


Abbildung 6.2: Lastgebirge aggregierter Client-Anfragen mit Verzögerung und Andeutung von Skalierungsprozessen

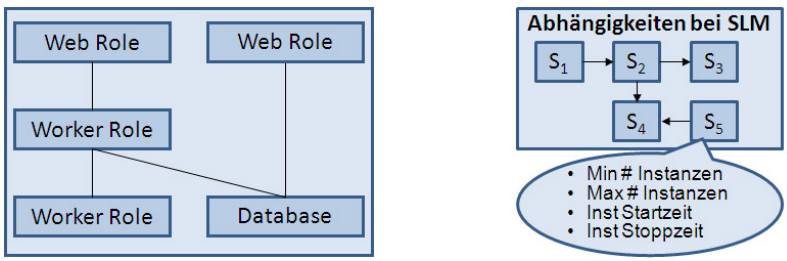
6.3a skizziert. Die beim Service-Load-Manager gespeicherte Abstraktion in Abbildung 6.3b besteht demnach aus den fünf verschiedenen Services S_1 bis S_5. Dabei gibt die Web-Rolle S_1 bei ihrer Registrierung eine Abhängigkeit zur Worker-Rolle S_2 an. Der Service S_2 übermittelt dem Manager seine Verbindung zu der anderen Worker-Rolle S_3 sowie zur geteilten Datenbank S_4. Eine weitere Web-Rolle S_5 deklariert ebenfalls eine Abhängigkeit zu S_4.

Um alle beim Service-Load-Manager gelisteten Abhängigkeiten zu verwalten, nutzt der vorgestellte Ansatz gerichtete azyklische Graphen der mathematischen Graphentheorie [43]. Deren Definition ergibt sich durch $G = (V, E)$, wobei V die Menge aller v_i Knoten und E die Menge der e_i gerichteten Kanten darstellt. Nach dem Vorschlag von Kwok et al. [95] repräsentieren die Graphknoten die Tätigkeiten der Systemkomponenten. Das Gewicht w eines Knoten n_i steht für die korrespondierenden Berechnungskosten $w(n_i) > 0$. Eine Kante zwischen zwei Knoten n_i und n_j ist durch das Tupel (n_i, n_j) gegeben. Ihr ist ein Kostenwert von $c(n_i, n_j) = \frac{w(n_i)}{w(n_j)}$ zugeordnet.

Aufbauend auf den klassischen gerichteten azyklischen Graphen entsprechen die beim Service-Load-Manager registrierten Cloud-Dienste den Graphknoten. Die so als Knoten dargestellten Datenbank-, Web- oder Worker-Rollen führen schrittweise die eigentliche Anfragenbearbeitung auf ihrer Ebene aus. Ihr jeweiliges Knotengewicht w bezeichnet die Anzahl n_i der laufenden Instanzen je Rolle und wird als Kostenwert $w(n_i)$ gespeichert. Eine über (n_i, n_j)-gegebene Kante bestimmt das Verhältnis zwischen den zwei zugehörigen Knoten. Ihre Relation wird als Gewicht $c(n_i, n_j) = \frac{w(n_i)}{w(n_j)}$ gespeichert.

Das Beispielszenario weist eine Knotenmenge von $V = \{S_1, ..., S_5\}$ auf. Das Kantengewicht ergibt sich entsprechend der Relationen zwischen den korrespondierenden Rollen. Wenn beispielsweise die Web-Rolle S_1 mit vier Instanzen läuft und die darunterliegende Worker-Rolle S_2 zwei Maschinen zur Anfragenbearbeitung beansprucht, ist ihr Verhältnis aktiver Elemente vier zu zwei.

Damit sind die grundlegenden Gegebenheiten der behandelten Systemumgebung skizziert. Der daran anknüpfende automatisierte Skalierungsansatz mit optimiertem Scheduling für abhängige Ressourcen besteht aus zwei Prozeduren. Zunächst wird ein Selbstkalibrierungsmechanismus zur Identifikati-



(a) Serviceumgebung (b) Abstraktion

Abbildung 6.3: Beispielhafte Dienstumgebung und gespeicherte Abstraktion

on der Abhängigkeiten zwischen den involvierten Komponenten durchgeführt. Dies erfolgt durch den Service-Load-Manager vor der eigentlichen Aufnahme des Dienstbetriebs. Anschließend findet im Standardbetrieb über den Service-Load-Manager ein Ressourcenregelverfahren statt. Diese zwei bedeutenden Lösungselemente werden nun konkretisiert.

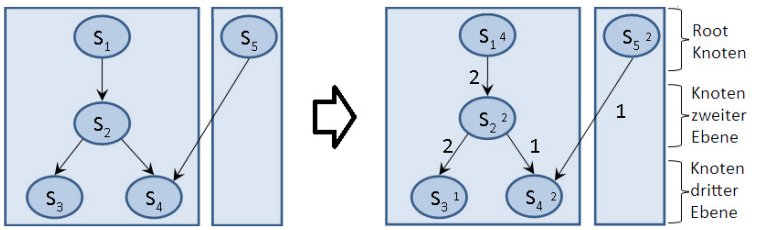
6.3 Selbstkalibrierungsmechanismus durch den Service-Load-Manager

Ein Ziel der Protokollerweiterung ist es, Cloud-Dienste als Gesamtkomplex zu skalieren, ohne die Auslastungswerte jeder einzelnen belegten Instanz überwachen zu müssen. Wie in Abbildung 6.4a illustriert, baut der Service-Load-Manager dafür einen gerichteten azyklischen Graphen auf, der die gemeldeten Abhängigkeiten zwischen den Ressourcen modelliert.

Nach der Registrierungsphase der Cloud-Services vervollständigt sich der grundlegende Abhängigkeitsgraph schrittweise mit realen Werten. In Abbildung 6.4b ist angedeutet, wie diese Zahlen laufender Instanzen in den Knoten und die Verhältnisse zwischen den einzelnen Ebenen an den Kanten vermerkt werden. Dieser Prozess wird nun erläutert.

Um automatisiert die Verhältnisse der abhängigen Ressourcen zu bestimmen, übergibt der Cloud-Service dem Service-Load-Manager einen Zielwert je Instanz. Dies geschieht vor dem tatsächlichen Dienststart. Die Zielgröße definiert die durchschnittliche CPU- oder Speicher-Auslastung. Damit wird zum Beispiel festgelegt, dass die CPU-Auslastung nicht einen Wert von 70 % überschreiten soll. In heutigen Rechenzentren hingegen werden die Server-Kapazitäten nur zu 5 % bis 10 % genutzt [108, 141, 162]. Mit der Verwendung des vorgestellten Protokolls wird eine wesentlich höhere Auslastung erreichbar, da der zukünftige Bedarf effizienter eingeplant wird [157].

Der Service-Load-Manager legt für jeden Dienst den übermittelten Standardwert minimal erforderlicher Kapazität in der zugehörigen Konfigurationsdatei an. Anhand dieser Vorgabe wird die definierte Anzahl an Ressourcen gestartet. Beispielsweise gibt der Service S_1 vor, dass er immer mit mindestens



(a) Graphaufbau (b) Graphvervollständigung

Abbildung 6.4: Graph der Dienstumgebung beim Service-Load-Manager

zwei virtuellen Maschinen bedient werden muss. Das wird beim Manager im Minimalwert für diese Ressource als $w_{min}(S_1) = 2$ gespeichert.
Dann folgt die Generierung einer fiktiven Last auf dem ersten Knoten im Graphen, also dem Root-Knoten des Dienstes. Die Last wird aufgrund der Abhängigkeiten ebenso an dahinterliegende Knoten weitergegeben. Der Manager fährt solange Instanzen auf der Komponente hoch, bis der vordefinierte Zielwert erreicht ist. Die zugehörige Anzahl laufender Maschinen $w_{act}(n_i)$ wird im Graphknoten als sein Gewicht $w(n_i)$ gespeichert. Analog geht der Manager für alle abhängigen Ressourcen tieferer Ebenen vor.
Anhand der Liste der Instanzenanzahl je spezifischer Ebene kann das Verhältnis der Rollen zueinander bestimmt werden. Im Beispielszenario erhält die Web-Rolle S_1 vier Instanzen zur Zielwerterreichung, $w(S_1) = 4$. Die darunterliegende Worker-Rolle benötigt initial zur Lastbewältigung zwei Instanzen, $w(S_2) = 2$. Für ihr Verhältnis und somit für ihr Kantengewicht ergibt sich:

$$c(S_1, S_2) = \frac{w(S_1)}{w(S_2)} = \frac{4}{2} = 2 \tag{6.1}$$

Die Worker-Rolle greift auf einen weiteren Worker S_3 mit einem Gewicht von $w(S_3) = 2$ und auf die Datenbank S_4 mit $w(S_4) = 2$ zu. Somit ergeben sich die Verhältnisse $c(S_2, S_3) = \frac{2}{1} = 2$ und $c(S_2, S_4) = \frac{2}{2} = 1$. Auch das Verhältnis der anderen Web-Rolle S_5 zur geteilten Datenbank S_4 ist $c(S_5, S_4) = \frac{2}{2} = 1$.
Abschließend wird der Regel-Prozess gestoppt. Die benötigten Instanzen sind nun anhand des Abhängigkeitsgraphen skalierbar. Zur Überprüfung kann der Selbstkalibrierungsmechanismus periodisch angestoßen werden. Bei Änderungen der Systemumgebung werden gegebenenfalls die gesetzten Werte korrigiert.

6.4 Ressourcenregelverfahren des Managers

Unter Verwendung des Abhängigkeitsgraphen kann der Service-Load-Manager die Ressourcen während des Regelbetriebs folgendermaßen regulieren. Zuerst werden alle ‘Composite Services’, also alle zusammenhängenden Dienstinstanzen, auf eine virtuelle Ressource reduziert. Nach der Dienstregistrierung sind dem Manager die abhängigen Limitationen der Services bekannt. Diese Begrenzungen beziehen sich auf die minimale und maximale Anzahl aktiver Instanzen und auf das benötigte Zeitintervall zum Hoch- und Herunterfahren.
Falls geteilte Ressourcen existieren – wie beim Datenbankdienst S_4 – wird zusätzlich das maximale Summenlimit unter den abhängigen Maschinen definiert. Dann werden zunächst in jedem Regelschritt alle Dienstanfragen vom Service-Load-Manager gesammelt. Dieser prüft dann die Limits der direkt vom Client angesprochenen Root-Ressource. Abschließend kontrolliert der Manager zusätzlich die aggregierten Summenlimits der abhängigen Dienste.
Sind alle Limitationen im akzeptierten Rahmen, steuert der Manager die Rol-

len gemäß ihrer Beziehungen und der definierten Gewichtung aus. Dabei wird weder die durch den Service festgelegte minimale Instanzenanzahl unterschritten noch über die korrespondierende Maximalmenge hinausgegangen.
Im Falle einer Überschreitung der Summenlimits von geteilten Ressourcen ist eine Lastfreigabe pro Service-Schicht umzusetzen. Die Festlegung, welche Anfragen dabei zurückgestellt werden, basiert auf den vom Ressourcenmanagement-Protokoll definierten Dienstprioritäten. Auf ein und derselben Prioritätsebene wird die vorhandene Kapazität zu gleichen Teilen verteilt. Durch die Verzögerung der verschiebbaren Lasten von Anfragen mit niedriger Priorität ist eine Durchsatzmaximierung in der Abarbeitung erreichbar. Der Manager kann sogar Clients auf höherer Prioritätsstufe im Scheduling-Prozess umsortieren, solange dabei keine Konflikte entstehen. Wie schon Kapitel 5 klar macht, werden dadurch die niederprioren Client-Anfragen zumindest kurzfristig nicht verworfen [156].
Auch wenn kein Verstoß gegen die Summenlimits vorliegt, können dennoch die Limitgrenzen einzelner virtueller External-Facing-Ressourcen erreicht werden. Kommt dies vor, muss der Client informiert werden, um rechtzeitig gegenzusteuern. Dafür wird das Feedback vom Service-Load-Manager generiert und mithilfe des Protokolls vermittelt. Eine Lastreduktion durch Request-Ablehnung kann folglich oft vermieden werden. Auf diese Weise wird die anliegende Arbeitslast so aufgeteilt, dass die Abarbeitung aller Anfragen bei abhängigen Ressourcen erheblich besser umsetzbar ist.

6.5 Abstrakte Analyse der Protokollerweiterung beim Skalierungsprozess

Selbstverständlich beansprucht die Generierung des nötigten Abhängigkeitsgraphen eine gewisse Zeit. Allerdings ist diese Zeit vernachlässigbar, da sie nach der initialen Relationsidentifikation auch mit wachsender Service-Zahl nicht ins Gewicht fällt. Aufgrund des Umfelds nicht zeitkritischer Anwendungen ist diese geringe Zeitspanne tragbar.
Im eingangs vorgestellten Beispielszenario wird nach der Grapherstellung die Arbeitslast nun so erhöht, dass in Web-Rolle S_1 der Bedarf nach zwei weiteren Instanzen entsteht. So wandelt sich das aktuelle Knotengewicht dieses Dienstes in $w_{act}(S_1) = 6$. Infolgedessen entstehen unausgeglichene Gewichte an den Kanten zu den verbundenen Worker- und Datenbank-Rollen S_2, S_3 und S_4.
Das Verhältnis der Web-Rolle S_1 und der Worker-Rolle S_2 ist ursprünglich auf zwei konfiguriert – siehe Formel (6.1). Aufgrund des Lastanstiegs werden nun zusätzliche Instanzen im Root-Knoten gestartet. Das Verhältnis ist nun gleich drei, wie Formel (6.2) beweist.

$$c_{act}(S_1, S_2) = \frac{w_{act}(S_1)}{w_{act}(S_2)} = \frac{6}{2} = 3 \neq 2 = c(S_1, S_2) \quad \lightning \tag{6.2}$$

Folglich muss eine Hochregulierung gemäß dem Algorithmus stattfinden. Diesen Vorgang skizziert Abbildung 6.5. Konkret müssen die aktuell laufenden Maschinen der Worker-Rolle S_2 mit einer weiteren Instanz aufgestockt werden, um das gegebene Verhältnis von zwei zurückzuerlangen. Das Gewicht der Rolle S_2 ändert sich so in $w(S_2) = 3$. Da die Last gleichermaßen an den tiefer liegenden Datenbank-Service S_4 durchgereicht wird, startet auch auf dieser Ressource eine neue Instanz. Das Knotengewicht steigt auf $w(S_4) = 3$.
Obwohl die Arbeitslast ebenfalls an die Worker-Rolle S_3 auf der dritten Ebene transferiert wird, muss dort keine zusätzliche Instanz hochgefahren werden. Dabei wird eine eher zurückhaltende Allokationsstrategie verfolgt, um Kosten zu sparen. Deshalb ist mit den drei aktiven Instanzen von S_2 und der bereits laufenden virtuellen Maschine von S_3 das Verhältnis noch ausreichend. Bei Betrachtung von Formel (6.3) wird diese Tatsache offensichtlich. Das resultierende Kantengewicht ist kleiner als vier. Erst ab diesem Wert ist ein Hochskalieren erforderlich. So gilt weiterhin $w_{act}(S_3) = w(S_3) = 1$.

$$c_{act}(S_2, S_3) = \frac{w_{act}(S_2)}{w_{act}(S_3)} = \frac{3}{1} = 3 \ < \ 4 \qquad \checkmark \tag{6.3}$$

Zudem kann ebenfalls Web-Rolle S_5 ein äußerst gefragter Dienst sein. Aufgrund der steten Laststeigerung findet analog zum vorherigen Prozess eine sukzessive Hochskalierung des Datenbankdienstes S_4 über die Rollen S_5 und S_1 statt. Nach einer gewissen Zeit wird schließlich das interne maximale Limit an Instanzen erreicht. Diese Schranke dient der Kostenbeschränkung nach oben. Folglich begrenzt die Maximalmenge an Instanzen im Backend unmittelbar das maximal mögliche Ressourcenvolumen im Frontend.
Selbst wenn in den Root-Knoten S_1 und S_5 jeweils größere Maschinenzahlen zulässig sind, erfahren sie trotzdem Performanzeinbußen durch den begrenzten Pool an S_4-Kapazität. Diese Situation wird über die aggregierten Informationen der einzelnen Log-Dateien bemerkt und eine entsprechende Warnung dazu generiert. Eine ineffiziente Diensterbringung, die nur einen Teil des Gesamtsystems bedient, ist dann durch den Service-Load-Manager über eine angemessene Drosselung von Service S_5 zu unterbinden.

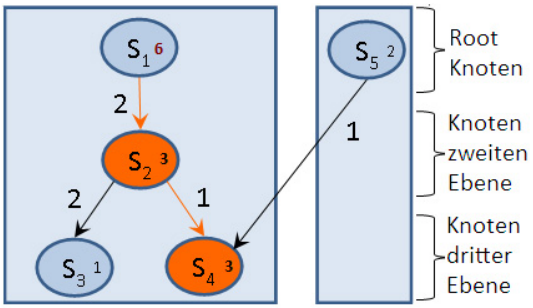


Abbildung 6.5: Skalierungsvorgang mittels Abhängigkeitsgraph

Auf diese Weise ist eine vorzeitige Erkennung von Kapazitätsengpässen möglich. Mithilfe der vordefinierten Ressourcenlimits kann der Service-Load-Manager also proaktiv Überlastung verhindern. Diese vorausschauende Identifikation von Flaschenhälsen ist auch für gemeinsam genutzte Instanzen anwendbar, indem die Berechnung über die akkumulierte Maschinenlast des Gesamtsystems erfolgt. Wird die Grenze der geteilten Ressource erreicht, beschließt der Service-Load-Manager anhand der zugehörigen Prioritäten eine alternative Ausführreihenfolge. Die Vordiagnose der Gesamtlimitationen ist demnach maßgeblich bedingt durch die Abhängigkeiten der Ressourcen.

Dieser Fakt wird bei Betrachtung des entworfenen Anwendungsfalls anschaulich. Web-Rolle S_1 gibt bei ihrer Registrierung an, dass mindestens zwei Instanzen $w_{min}(S_1) = 2$ und maximal zehn virtuelle Maschinen $w_{max}(S_1) = 10$ auf der Ressource laufen sollen. Die Worker-Rolle S_2 benötigt wiederum minimal eine Instanz $w_{min}(S_2) = 1$ und maximal vier $w_{max}(S_2) = 4$. Wie mit Formel (6.2) gezeigt, ist das aktuelle Abhängigkeitsverhältnis zwischen Service S_1 und dem dahinterlaufenden Dienst S_2 sechs zu drei, also $c(S_1, S_2) = 2$. Der Root-Knoten S_1 darf demgemäß beim Lastanstieg die Instanzen von S_2 fortlaufend verdoppeln, solange die Maximalwerte nicht erreicht sind. Zur Bestimmung des verfügbaren Kapazitätenumfangs ergeben sich die folgenden Ausdrücke.

$$w_{min_{real}}(S_1) = min\{w_{min}(S_1), [2 * w_{min}(S_2)]\} = min\{2, [2 * 1]\} = 2 \quad (6.4)$$

$$w_{max_{real}}(S_1) = min\{w_{max}(S_1), [2 * w_{max}(S_2)]\} = min\{10, [2 * 4]\} = 8 \quad (6.5)$$

Nach Formel (6.4) hat die virtuelle Ressource S_1 weiterhin eine Unterschranke von mindestens zwei laufenden Instanzen. Allerdings zeigt Formel (6.5), dass ihr Maximalgewicht effektiv nur bis acht Maschinen reicht. Ein Ausreizen der technisch erlaubten S_1-Obergrenze von zehn Instanzen schafft keinen Mehrwert, weil sich die Reglementierung auf tieferer Ebene nach oben auswirkt. Service S_1 ist also durch seine Abhängigkeit zu Rolle S_2 limitiert, da der Dienst S_2 weniger Kapazität als eigentlich von S_1 gefordert zur Verfügung stellt. Dies ist mit der Verwendung dieses Algorithmus vorzeitig feststellbar und im Scheduling berücksichtigbar.

Weiterhin ist es möglich, Annahmen zu machen, ab welchem Zeitpunkt eine zusätzlich beantragte Kapazität auf höchster Schicht frühestens einzufordern ist. Wie umfangreich das Startintervall für die Verfügbarkeit des gesamten Dienstes ausfällt, ist dabei über eine Maximumauswertung entlang des Abhängigkeitsgraphen berechenbar.

In diesem Szenario weist das Zuschalten neuer Instanzen der Web-Rollen S_1 und S_5 einen Zeitbedarf von drei Minuten auf. In den Worker-Rollen S_2 und S_3 beansprucht die Hochskalierung fünf Minuten. Zusätzliche Datenbankinstanzen benötigen für die Installation des Base-Image und der spezifischen Mandanten-Software insgesamt neun Minuten. Damit ergibt sich durch das Maximum der parallel startenden Maschinen eine neunminütige Verzögerung beim Hochregulieren des Gesamtsystems.

Das Herunterfahren der Maschinen und Löschen abgelegter Daten benötigt ebenfalls einige Zeit. Analog zur vorherigen Überlegung können auch für diesen Fall Aussagen getroffen werden, wann eine Ressource nach ihrer Freigabe durch einen Dienst vom Manager tatsächlich wieder einzuplanen ist.
Insgesamt minimiert sich durch den Einsatz der Protokollerweiterung der organisatorische Aufwand der Skalierung. Das senkt gleichzeitig die administrativen Kosten und verbessert das gesamte Ressourcenmanagement zusätzlich.

6.6 Ergebnisse der Protokollerweiterung

Nach der abstrakten Analyse des entwickelten Verfahrens wird nun auf die Ergebnisse der prototypischen Umsetzung eingegangen. Abbildung 6.6 präsentiert den Überblick über die Implementierung des beschriebenen Skalierungsalgorithmus. Auf dem externen Server links im Bild läuft der Service-Load-Manager. Über ihn wird eine fiktive Last auf dem ersten Graphknoten, *Rolle* 1, erzeugt. Diese Rolle fährt solange Instanzen I_1 bis I_n hoch, bis der Zielwert von 70-prozentiger CPU-Beanspruchung erreicht ist.
Dieser variabel festzusetzende Belastungswert ist mithilfe des Protokolls umsetzbar. Zum einen wird so die Systembelegung im Vergleich zum heute realen Auslastungsumfang um ein Vielfaches gesteigert. Andererseits ist trotzdem noch eine gewisse Reserve an freier Kapazität gewahrt. Falls viele nichtverzögerbare Anfragen eintreffen, kann also dennoch ihre zeitnahe Abarbeitung erfolgen. Die Anzahl laufender Instanzen wird mit ihrem durchschnittlichen Lastvolumen in einer internen Log-Datei abgelegt.
Gemäß dem vorgestellten Algorithmus wird die generierte Arbeitslast an jede nachfolgende Maschine, *Rolle* 2 bis *Rolle X*, weitergegeben. Dort findet jeweils

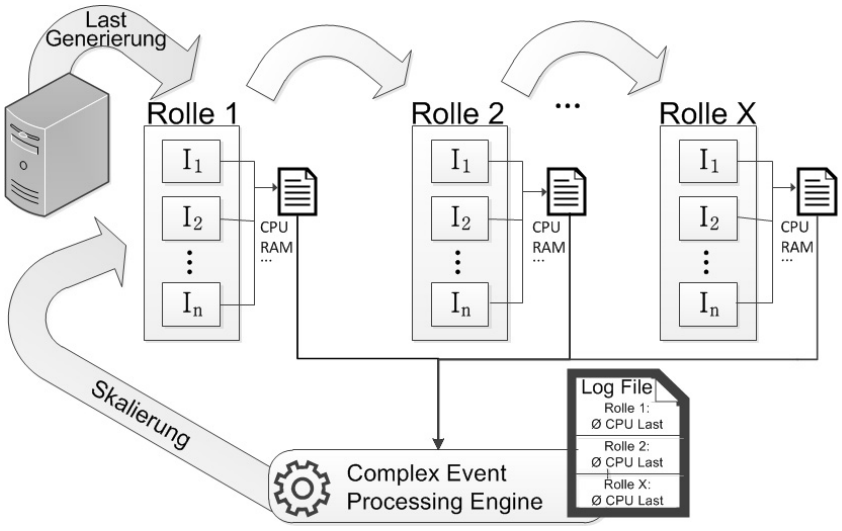


Abbildung 6.6: Implementierung des automatischen Skalierungsprozesses

der analoge Vorgang statt, um die Grundeinstellung des Gesamtsystems vorzunehmen. Die vervollständigten Log-Dateien werden an eine Complex Event Processing Engine übermittelt. Sie konsolidiert alle transferierten Werte und übergibt sie dem Service-Load-Manager.
Diese Daten können nun vom Manager in den Abhängigkeitsgraphen eingepflegt werden. Nach der Definition des Graphen kann der Manager die Ressourcen auf Basis ihrer Verhältnisse skalieren. So ist es nicht mehr nötig, alle Instanzennutzungen zu kontrollieren.
Der Service-Load-Manager setzt dabei das bereits erläuterte Scheduling-Verfahren des Ressourcenmanagement-Protokolls ein, um die Auslastung der Rollen auf gleichmäßige Lastverteilung hin zu optimieren. Infolgedessen können die nötigen Skalierungsvorgänge durch geschickte Umverteilung zusätzlich minimiert werden. Nachfolgend wird der Vorteil eines Einsatzes des entwickelten Scheduling-Algorithmus mit eindeutigen Zahlen belegt.

6.6.1 Evaluation des protokollspezifischen Scheduling-Verfahrens

Zur Bestimmung der Scheduling-Güte werden fünf verschiedene Szenarien simuliert. Die Basiskonfiguration dieser Anwendungsbereiche richtet sich nach den identifizierten Lastmustern des Cloud-Umfelds aus Abschnitt 4.2.
Außerdem fließen die Erkenntnisse von Misra et al. [126] bei der Ergebnisbestimmung mit ein. Durch präzise und reale Unternehmenswerte begründet, definieren die Autoren ähnliche Lastcharakteristika zu den in dieser Arbeit vorgestellten. Dafür beleuchten sie unterschiedliche Faktoren wie die verfügbare Server-Menge, die Verteilung der Server, die Nutzeranzahl und die erwirtschafteten Einnahmen. Anhand dessen wird eine Kategorisierung der betrachteten Cloud-Umgebungen vorgenommen. Der betrachtete Datenumfang reicht pro Monat von 100 Gigabyte bis 100 Terabyte. Dieser Datenfundus dient der grundsätzlichen Systemeinstellung der durchgeführten Simulationsläufe.
Die Simulation des ersten Szenarios umfasst Lastmuster mit mäßig schwankender Beanspruchung ohne abrupten Anstieg. Dies ist in der heutigen Geschäftswelt beispielsweise auf Email-Servern zu beobachten.
Das zweite Lastprofil erfüllt starke Belegungsvarianzen mit hohen Ausschlägen. Eine mögliche Implementierung sind Handlungsprozesse auf Ticket-Servern oder auf den Rechnern von online Auktionshäusern.
Im dritten Muster wird eine konstante Belastung angenommen. Sie ist typischerweise auf rechenstarken Suchmaschinen erkennbar. Batch-Job-Szenarien zeigen für Zeitintervalle mit anliegender Last ebenfalls solche Eigenschaften.
Die Umsetzung des vierten Modells verwirklicht moderate Laständerungen mit gelegentlich hohen Spitzen. Dies tritt meist auf Servern von News-Seiten auf.
Mit der Realisierung des fünften Profils wird der Fall abgedeckt, dass grundsätzlich Lastkontinuität vorherrscht. Sie fällt allerdings abhängig vom Jahres-

intervall unterschiedlich stark aus. Das Szenario befasst sich mit dem charakteristischen Verlauf im Forschungsumfeld. Die Systembelegung orientiert sich an der Projektdichte. Abbildung 6.7 skizziert die fünf Lastsimulationen.

Die zu bearbeitenden Tasks, also die simulierten Anfragen, werden so gesetzt, dass sie zusammengenommen das korrespondierende Lastmuster ergeben. Die einzelnen so erzeugten Lastgebirge sind in Abbildung 6.8 visualisiert. Die y-Achse repräsentiert die Gesamtkapazität. Sie ist durch die Summe an laufenden Instanzen bestimmt. Über die Dauer der Laufzeitrunden werden die Ressourcen unterschiedlich stark von den farbig dargestellten Tasks beansprucht. Die Überschreitung einer vertikalen Linie bedeutet, dass zur Lastbewältigung eine zusätzliche virtuelle Maschine zu starten ist.

Ersichtlich wird weiterhin, dass unterschiedlich viele Ressourcen in den einzelnen Simulationen nötig sind. Wie insgesamt zu erkennen, können die identifizierten Lastmuster recht gut angenähert werden.

Bei der Lastbearbeitung werden je drei Scheduling-Implementierungen verglichen. Die im Rahmen dieser Arbeit vorgestellte Lösung wird dabei auf Basis der Strategien des *Random*- und des *Round-Robin*-Ablaufs bewertet.

In den Simulationsdurchläufen wird zur Vergleichbarkeit die Systemkonfiguration für alle untersuchten Methoden zunächst an eine feste Menge von Ressourcen gebunden. Bei der Verwendung des bereits beschriebenen automatisierten Skalierungs- und Scheduling-Verfahrens passt sich die Umgebung dann jedoch an. Nach Erreichen des Optimums der Maschinenzahl werden in dieser Ausführung gemäß dem Algorithmus keine weiteren Instanzen hinzugeschaltet.

Die *Random*-Ausführungsreihenfolge verfolgt eine auf Zufall beruhende Abarbeitung der zu bedienenden Requests. Die aktuellen Auslastungswerte laufender Komponenten sind nicht berücksichtigt. Bei dieser vereinfachten

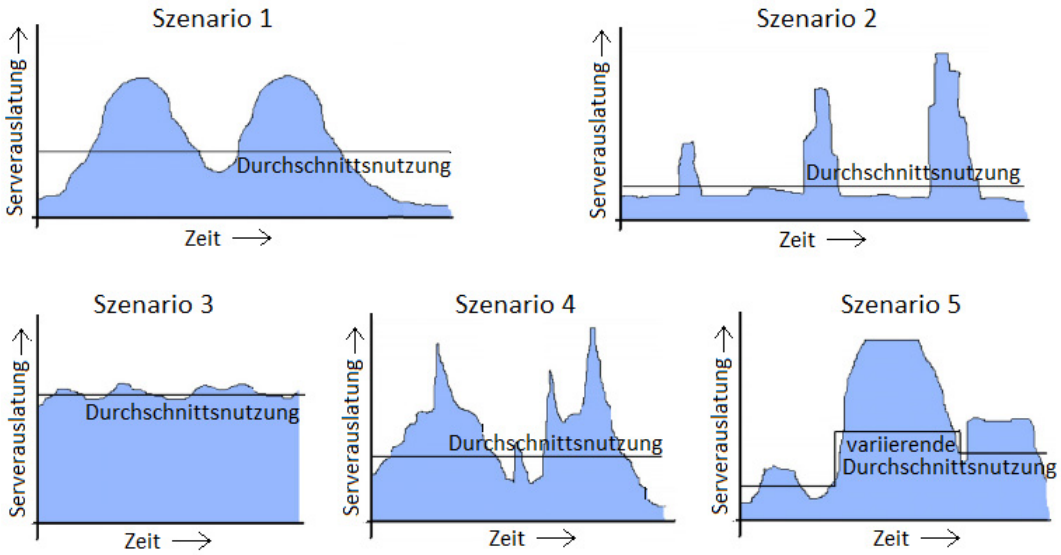
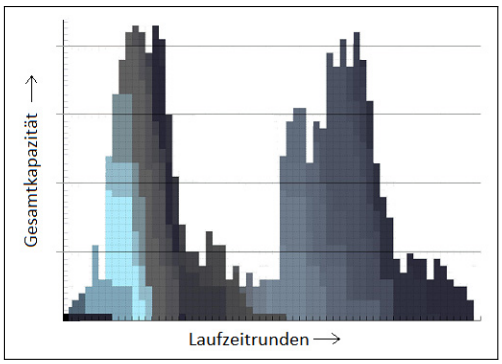
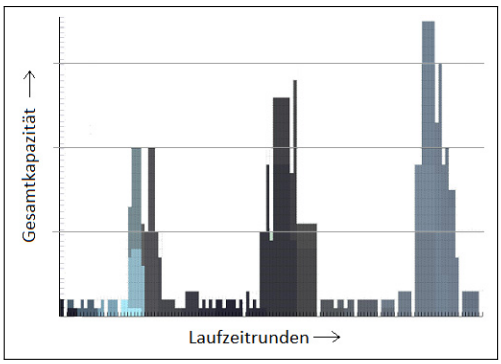


Abbildung 6.7: Lastszenarien der Simulationen angelehnt an Misra et al. [126]

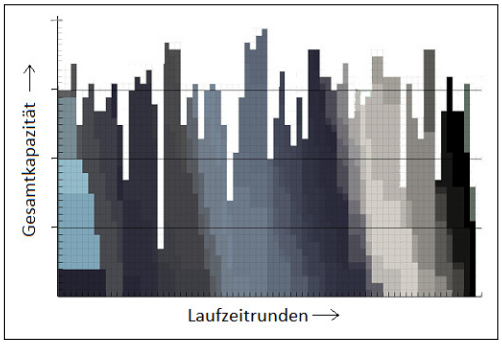
Scheduling-Praxis muss wenig Logik umgesetzt werden. Aus einem Pool von Tasks wird je eine Aufgabe nach der anderen zufällig herausgenommen und der Reihe nach bearbeitet. Theoretisch kann eine Maschine so stets rasch bedienbare Anfragen erhalten, während eine andere Ressource mit rechenintensiven Requests überhäuft wird. Folglich ist diese Abarbeitung der anstehenden Jobs sowohl für die wartenden Clients als auch für die bearbeitenden Server höchst unausgeglichen. Trotzdem zeigt sich in der Praxis und bei der Betrachtung ver-



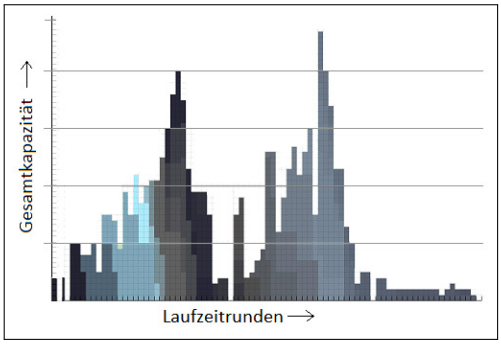
(a) Lastgebirge auf Email-Servern



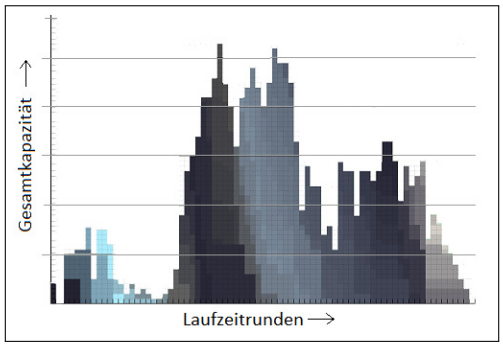
(b) Lastgebirge auf Ticket-Servern



(c) Lastgebirge auf Suchmaschinen



(d) Lastgebirge auf Servern für News-Seiten



(e) Lastgebirge im Forschungslaborumfeld

Abbildung 6.8: Lastgebirge der simulierten Lastszenarien

wandter Arbeiten, dass dieses Verfahren aufgrund des geringen Verwaltungs- und Implementierungsaufwands immer noch zum Einsatz kommt.
Die Vorgehensweise des *Round-Robin* ist in der klassischen IT-Welt seit langem bekannt. Bei diesem Rundlaufverfahren ist die Verteilung der Aufgaben möglichst so ausgerichtet, dass die Ressourcen gleichmäßig beansprucht werden. Dafür wird allen anfragenden Clients nacheinander je ein Zeitschlitz zur Ressourcennutzung eingeräumt. Falls die gesamte Anfrage in diesem Zeitraum nicht beantwortet werden kann, wird der noch zu behandelnde Aufgabenteil wieder in der Warteschlange eingereiht. Zur Weiterbearbeitung muss der Client die Zuteilung des nächsten Intervalls abwarten. Da diese schlichte Gesetzmäßigkeit gut in einem programmatischen Ablauf umzusetzen ist, wird dieser Ansatz von einer Vielzahl gewöhnlicher Lastbalancierer vertreten.
Mit dem entwickelten Selbstkalibrierungsmechanismus und seinem Regelverfahren bei abhängigen Ressourcenanforderungen ist das System über den Service-Load-Manager selbstständig skalierbar. Außerdem kann so eine verbesserte Abarbeitungsreihenfolge erstellt werden. Diese Lösung des automatischen Skalierens und des optimierten Schedulings wird in der folgenden Ergebnispräsentation verkürzt mit *Auto* bezeichnet.
In den Simulationen erfolgte zum einen jeweils die Bestimmung der *Rescheduled Tasks*, also der Anzahl an erneut einsortierten Anfragen. Andererseits zeichnet das System die Gesamtlaufzeit aller virtuellen Maschinen im Parameter *Runtime VMs* auf. Wie lange die Maschinen zur Bearbeitung der Gesamtlast aktiv sind, wird dabei in Runden gezählt.

6.6.2 Ergebnisse der Anwendungssimulationen

Für das Profil leicht schwankender Systembelegung mit nur geringen Lastspitzen wird als klassisches Business-Muster ein Email-Server simuliert. Die einzelnen Durchläufe evaluieren die drei Scheduling-Strategien je mit einer bis acht virtuellen Maschinen. Beim Start einer Ressource ergibt sich für alle Verfahren die gleiche Bearbeitungslaufzeit von 129 Runden. Mit steigender Instanzenanzahl treten die Unterschiede zwischen der hier entworfenen und der Standardlösung deutlich hervor. Tabelle 6.1 listet die Ergebnisse auf.
Für die *Round-Robin*-Strategie und die *Random*-Abarbeitungsreihenfolge ergeben sich bei zunehmender Instanzenanzahl fast identische, konstant steigende Gesamtlaufzeiten zwischen 129 und 504 Runden. Diese Ähnlichkeit begründet sich durch die rein zufällig gleichartige Auswahl der Task-Abarbeitung. Dagegen nimmt die *Runtime VMs* des *Auto*-Mechanismus wesentlich gemäßigter zu, auf Werte von 144, 173 und 191 Runden. Bereits bei fünf Maschinen mit 201 Runden wird die Maximallaufzeit erreicht.
Die Anzahl der *Rescheduled Tasks* beläuft sich zu Beginn dieser Szenarioumsetzung ebenfalls für alle drei Abarbeitungsmethoden auf den gleichen Ergebniswert. Er liegt bei 61 Wiedereinreihungen. Darauffolgend unterscheiden sich die in dieser Kategorie gewonnenen Resultate aller drei Strategien.

In der Zufallsbearbeitung des *Random*-Verfahrens sinken mit zunehmend bereitgestellter Kapazität die erneut eingereihten Anfragen sukzessive auf 50, 27, 22, 11, 9 und schließlich 7 Neubearbeitungen.
Im Rundlaufverfahren müssen bei zwei Ressourcen 40 Tasks und mit drei VMs 29 Anfragen erneut aufgenommen werden. In diesem Fall ist der *Random*-Ablauf sogar etwas besser. Allerdings schneidet der *Round-Robin*-Schedule bei vier und fünf Ressourcen wieder wesentlich besser ab, da die *Rescheduled Tasks* lediglich 18 beziehungsweise 9 Jobs umfassen. Mit sechs, sieben und acht Systemkomponenten nimmt diese Anzahl weiter ab auf 8, 5 und schließlich 2 Neuaufnahmen der Tasks. Demzufolge ermöglicht das *Round-Robin*-Scheduling eine raschere Reduktion der wieder eingeordneten Tasks.
Abbildung 6.9 stellt die ressourcenspezifischen Lastverläufe bei fünf aktiven Instanzen für das *Random*- und das *Round-Robin*-Verfahren gegenüber. Dies gibt einen beispielhaften Überblick, wie jede virtuelle Maschine über die Zeit bei der Bearbeitung der Anfragen belastet ist. Die horizontal übereinanderliegenden Lastgebirge unterscheiden die eingesetzten Ressourcen. Auf unterster Ebene ist die Bearbeitung der ersten Maschine abgebildet und im oberen Abschnitt die der fünften.
In den Schaubildern repräsentiert jedes farbige Pixel eine Anfrage in einer Runde der Gesamtlaufzeit. Die nichtfarbigen Zwischenräume verkörpern inaktive Phasen. Die Farben dienen der klareren Visualisierung der einzelnen Tasks. Grundsätzlich belegt eine komplexe Aufgabe, die hohe Rechenleistung erfordert, viel Kapazität auf der jeweils bedienenden Maschine. Dieser Task-Umfang ist über die vertikale Höhe je Rundenspalte abgebildet. Weiterhin ist die benötigte Dauer zur Verarbeitung der Anfrage über die zusammengehörigen Farbzeilen über mehrere Rundenspalten visualisiert.
Der Fall übereinanderliegender unterschiedlicher Farbblöcke steht für die parallele Ausführung mehrerer Tasks gleichzeitig. Sobald hierbei ein Request erfolg-

Scheduling-Strategie		**Anzahl VMs**							
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Random	Runtime VMs	129	148	189	252	315	378	441	504
	Rescheduled Tasks	61	50	27	27	22	11	9	7
Round-Robin	Runtime VMs	129	150	189	252	315	378	441	504
	Rescheduled Tasks	61	40	29	18	9	8	5	2
Auto	Runtime VMs	129	144	173	191	**201**	**201**	**201**	**201**
	Rescheduled Tasks	61	34	17	9	**2**	**2**	**2**	**2**

Grün: Optimales Kapazitätsvolumen erreicht

Tabelle 6.1: Simulationswerte für Business-Lastmuster

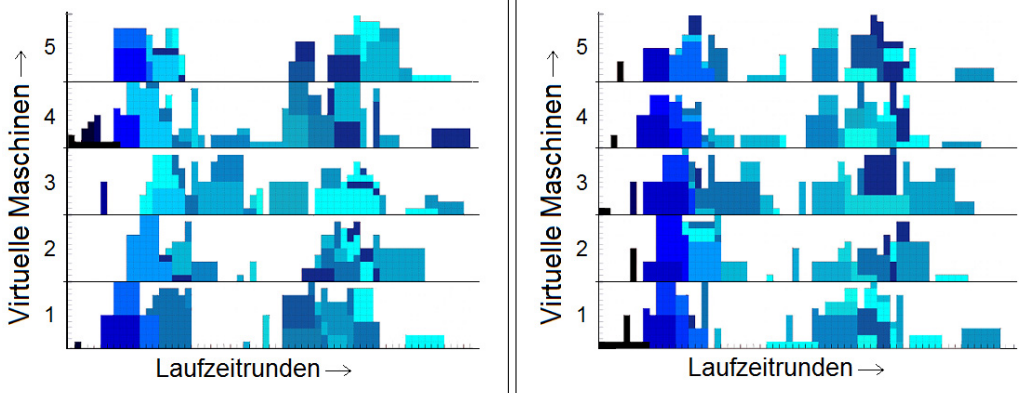
reich beendet ist, verschiebt sich der daraufliegende Farbblock maschinenintern nach unten. Dementsprechend sieht man in den Grafiken, wie viel Kapazität pro virtueller Maschine eine spezielle Anfrage belegt. Zudem zeigt sich der Umfang, den die Aggregation aller Jobs beansprucht.

Im Vergleich der Abbildungen 6.9a und 6.9b zeigt sich, dass der *Round-Robin*-Scheduler die Ressourcen etwas gleichmäßiger auslastet als das *Random*-Verfahren. Dies ist an der geringeren Anzahl nichtfarbiger Zwischenräume festzumachen. Die großen Blöcke zu Beginn der Gesamtlaufzeit bedeuten, dass bei beiden Verfahren mehrere Anfragen erneut eingereicht werden müssen.

Mit der entwickelten *Auto*-Methode sind diese *Rescheduled Tasks* gegenüber den anderen beiden Strategien erheblich schneller herabsetzbar. Bei stufenweiser Kapazitätszunahme auf fünf VMs findet mindestens eine Halbierung der Werte statt. Ab fünf Maschinen mit zwei erneuten Einsortierungen kann die wiederzubearbeitende Anfragenmenge aufgrund der Charakteristik der Lastmuster nicht weiter minimiert werden. Die Wertverläufe belegen, dass auch größere Instanzenanzahlen keine weitere Reduktion der *Rescheduled Tasks* herbeiführen. Es steigen lediglich die Ressourcenkosten.

Bemerkenswert ist, dass der vorgestellte *Auto*-Algorithmus diese Tatsache im Voraus automatisch diagnostiziert. Deshalb werden nach dem Start von fünf Maschinen auch keine weiteren allokiert. Folglich steigt die Gesamtlaufzeit des Systems nicht über einen Wert von 201 Runden. In diesem Szenario wird das optimale Kapazitätsvolumen bei fünf Ressourcen erfüllt. In der vorangegangenen Tabelle sind diese Werte grün gekennzeichnet.

Dieser Fakt tritt auch in Abbildung 6.10 deutlich hervor. Die Darstellung visualisiert die zeitabhängige Belastung der Maschinen bei der *Auto*-Anfragenbearbeitung. Da der Algorithmus regulierend eingreift, werden in inaktiven Phasen die Maschinen ohne anliegende Last vorerst freigegeben.



(a) Maschinenspezifische Lastgebirge im *Random*-Scheduling

(b) Maschinenspezifische Lastgebirge im *Round-Robin*-Scheduling

Abbildung 6.9: Lastdarstellung einzelner aktiver Maschinen auf Email-Servern bei Verwendung des *Random*-Schedulers (a) bzw. *Round-Robin*-Verfahrens (b)

Mit anderen Worten bedeuten in diesem Fall nichtfarbige Zwischenräume, dass die Ressource nicht läuft. Damit entstehen geringere Kosten als bei den *Random-* und *Round-Robin*-Verfahren. Bei diesen sind stets alle Instanzen aktiv, auch wenn aktuell keine Belegung zu verzeichnen ist. Ein Herunterfahren rechnet sich dort nicht, da die Zeiträume zwischen den Task-Verarbeitungen auf den einzelnen Maschinen zu kurz sind.

Abbildung 6.10 zeigt, dass die Jobs mit fünf Ressourcen in kürzester Zeit bearbeitet werden. Mit diesem Volumen enden die Laufzeitrunden wesentlich früher als mit geringerer Kapazität. Das ist durch die erheblich reduzierten *Rescheduled Tasks* im Vergleich zu den Fällen mit weniger Instanzen begründet.

Interessant ist die Beobachtung, dass im zweiten Zeitintervall viele Anfragen gleichzeitig eintreffen. Dann reicht ein geringes Kapazitätsvolumen nicht aus, um alle Tasks simultan zu bearbeiten. Das „Verhungern“ eines Jobs, der stets von höher priorisierten Anfragen hinten angestellt wird, muss verhindert werden. Deshalb erfolgt die Task-Bearbeitung nicht an einem Stück. Stattdessen werden die Anfragen immer wieder pausiert.

Ohne den in dieser Arbeit entwickelten Algorithmus müssen die Jobs deshalb zur weiteren Bearbeitung stets erneut einsortiert werden. Infolgedessen wächst auch die Bediendauer aller Anfragen erheblich. In den Verläufen ist dieser Umstand an längeren Farbelementen erkennbar. Da der Algorithmus dies rechtzeitig feststellt, wird automatisch skaliert.

Zudem wird aus den Darstellungen 6.9 und 6.10 ersichtlich, dass das *Auto*-Verfahren eine bessere Gruppierung der Anfragen vornimmt als die Standard-Scheduler. Die farbigen Blöcke der Tasks liegen beim entwickelten Algorithmus näher beieinander. Mithilfe dieser Aufteilung warten weniger Instanzen im Leerlauf auf Anfragen. Aufgrund der Abschaltung der Maschinen bei Nichtbelegung, wird die Vergeudung teurer Kapazität verhindert.

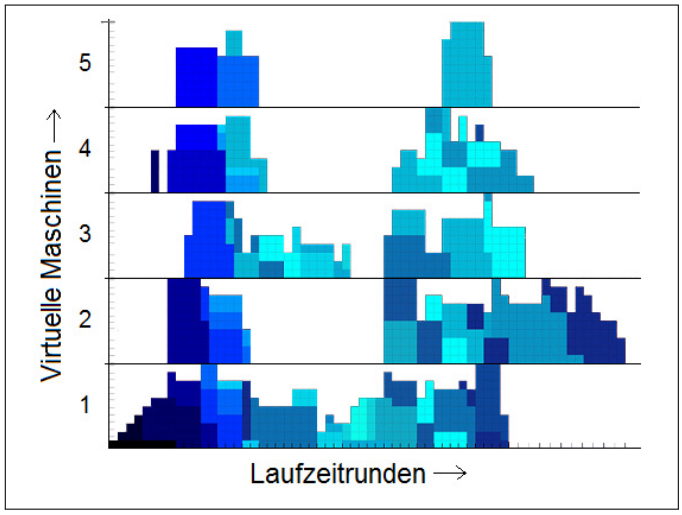


Abbildung 6.10: VM-Lastgebirge beim *Auto*-Scheduling auf Email-Servern

Das zweite Szenario implementiert starke Schwankungen mit hohen Ausschlägen in der gesamten Arbeitslast, wie sie auf Ticket- oder online Auktionshaus-Servern vorkommen. Die gemessenen Resultate aus Tabelle 6.2 offenbaren, dass ab fünf Maschinen *alle* Scheduling-Verfahren keinen weiteren Änderungen unterliegen – hier grau und blassgrün hinterlegt.

Ursächlich dafür ist das zugrundeliegende Lastmuster, das dies entsprechend vorsieht. Da sich ab diesem Ressourcenumfang ein konstanter Verlauf abzeichnet, werden nur die Ergebnisse des relevanten Ausschnitts von einer bis fünf VMs näher betrachtet. Die Verfahren sind analog zur vorherigen Simulation in Abbildung 6.11 visualisiert.

Wie im ersten Szenario nimmt auch in diesem Ablauf unabhängig von der angewendeten Abarbeitungsstrategie das Volumen an benötigten Runden mit wachsender Maschinenlaufzeit zu. Bei der *Random-* und der *Round-Robin-*Bearbeitung liegen die Simulationsresultate jeweils in einem Intervall von 146 bis 625 Runden. Die Werte steigen dabei pro hinzugenommener Instanz je um ca. 110 Runden. Sobald schließlich die fünf Ressourcen laufen, tritt – wie erläutert – keine weitere Veränderung der Ergebniswerte ein.

Demgegenüber ergeben sich mit dem entworfenen Algorithmus je Maschinenzuschaltung Ergebnisse von 146 Runden über 179 und 241 Runden bis zu nur 261 Runden. Die Qualitätsverbesserung der Abarbeitung durch den *Auto-*Scheduler stellt sich also im Vergleich zu den Richtmaßausführungen noch deutlicher heraus als im ersten Szenario.

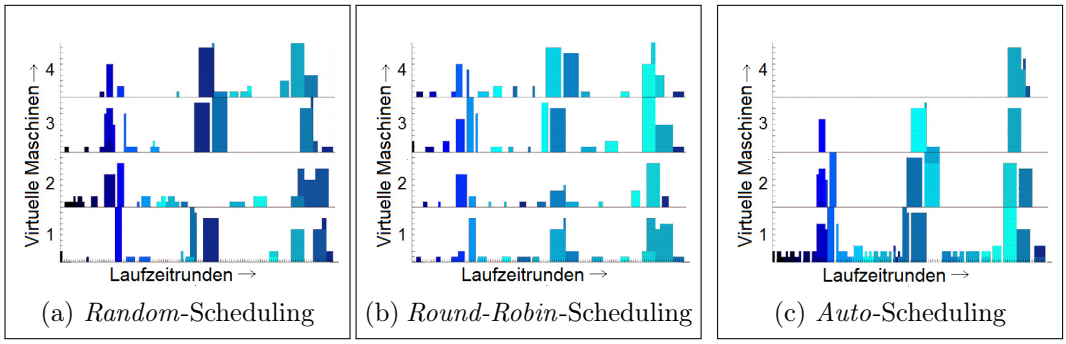
Unter Verwendung der beiden Standards werden jeweils bei einer Komponente 27 Anfragen und bei doppelter Kapazität 12 Aufgaben erneut bearbeitet. Beim *Random*-Verfahren sinkt die Anzahl von 10 Wiedereinreihungen bei drei Ressourcen über 6 Tasks auf 3 Jobs bei fünf virtuellen Instanzen und stagniert

Scheduling-Strategie		**Anzahl VMs**							
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Random	Runtime VMs	146	254	384	500	625	625	625	625
	Rescheduled Tasks	27	12	10	6	3	3	3	3
Round-Robin	Runtime VMs	146	256	375	500	625	625	625	625
	Rescheduled Tasks	27	12	3	4	2	2	2	2
Auto	Runtime VMs	146	179	241	**261**	**261**	261	261	261
	Rescheduled Tasks	27	9	5	**2**	**2**	2	2	2

Grün: Optimales Kapazitätsvolumen erreicht
Blassgrün / Grau: Keine Veränderungen für alle Scheduling-Verfahren

Tabelle 6.2: Simulationswerte auf Ticket- oder Auktionshaus-Servern

bei diesem Wert. Durch das *Round-Robin*-Scheduling wird die Rate bei drei VMs auf 3 *Rescheduled Tasks* verringert.
Allerdings zeigt sich dann wieder ein höherer Wert von 4 Neubearbeitungen trotz zusätzlicher Maschinen. Dieser Ausreißer ist jedoch nicht signifikant. Er fällt nicht weiter ins Gewicht, da sich ab fünf Ressourcen die Anzahl der *Rescheduled Tasks* auf 2 beläuft.
Die Auswertung der *Auto*-Strategie ergibt eine Drittelung der anfänglichen 27 Wiederbearbeitungen auf nur noch 9 Jobs bei zwei virtuellen Maschinen und minimiert sich auf 5 Tasks bei drei Komponenten. Bereits ab vier laufenden VMs mit den korrespondierenden 2 wiederholten Einreihungen greift der automatisierte Algorithmus regulierend ein. Er legt fest, dass nun keine weitere Kapazität hochgefahren werden muss, da vorausschauend erkannt wird, dass die *Rescheduled Tasks* nicht weiter verringerbar sind.
Analog zum ersten Szenario sind in Abbildung 6.11 die Lastverläufe bei vier aktiven Instanzen für die drei Scheduling-Verfahren gegenübergestellt. Darin ist anschaulich festzustellen, wie die *Round-Robin*-Methode die Anfragen verarbeitet. Der Scheduler springt jeweils zwischen den einzelnen virtuellen Maschinen hin und her. Auf diese Weise wird eine gleichmäßige Auslastung erreicht. Allerdings werden so stets alle Ressourcen eingesetzt. Damit sind effiziente Kosteneinsparungen nicht realisierbar.
Besonders klar tritt hervor, dass im *Auto*-Verfahren deutlich weniger Ressourcen über die Zeit aktiv sein müssen. Tatsächlich wird die vierte Maschine erst gegen Ende des Rundenverlaufs zugeschaltet. In der übrigen Zeit reichen geringere Kapazitäten aus. Zeitweise läuft sogar lediglich eine Instanz.



(a) *Random*-Scheduling (b) *Round-Robin*-Scheduling (c) *Auto*-Scheduling

Abbildung 6.11: Lastdarstellung aktiver Maschinen auf Ticket-Servern bei der Ausführung des *Random*-, *Round-Robin*- und *Auto*-Schedulings

Im dritten Simulationsdurchlauf wird eine konstante Arbeitslast auf rechenstarken Servern realisiert. Dies ist bei der Lastverarbeitung auf Zusammenschlüssen von Suchmaschinen oder bei Batch-Job-Systemen in Perioden anliegender Arbeitslast erkennbar. Basierend auf diesem zugrundeliegenden Lastmuster unterscheiden sich die drei Ausführreihenfolgen der Anfragenbearbeitung bei begrenztem Pool, wenn überhaupt, nur geringfügig.

Tabelle 6.3 stellt die Ergebnisse dar. Sie liegen mit einer Maschine jeweils bei 228 Laufzeitrunden. Im zweiten Fall werden Werte um 250 Runden gemessen. Innerhalb des Intervalls von drei bis fünf VMs schneidet die *Random*-Ausführungsfolge bezüglich der durchgeführten Runden recht gut ab. Mit 264, 312 und 375 Runden ist dieses Scheduling mindestens genauso effizient und sogar einmal um 20 Runden besser als die *Round-Robin*-Methode. Dabei liegt die Anzahl der wiedereingereichten Jobs bei der Zufallsbearbeitung bis vier VMs nur etwas über den Werten der *Round-Robin*-Methode.

Faktisch ergeben sich in der ersten Standardkategorie 106 bis 48 *Rescheduled Tasks* und im *Round-Robin* 106 bis 45 Jobs. Das kann auch die Tatsache begründen, warum das *Random*-Verfahren vielfach als Benchmark der Bearbeitungsqualität eingesetzt wird. Über eine große Verlaufsmenge können im Durchschnitt oft rein zufällig ähnliche Werte erzielt werden wie im wesentlich komplexer umzusetzenden *Round-Robin*. Dies gilt gerade in diesem Szenario, da sich die anliegende Last durchwegs an der Durchschnittsnutzung orientiert. Dies ist bereits in Abbildung 6.7 auf Seite 84 zu erkennen.

Der weitere Vorgang der zwei Standardmethoden zeigt mit entsprechender Ressourcenskalierung Werte von 511 und 518 Runden. Schlussendlich beläuft sich die Gesamtlaufzeit bei acht Maschinen jeweils auf 584 Runden. Allerdings wird bei Betrachtung der *Rescheduled Tasks* klar, dass der *Round-Robin*-Algorithmus ab fünf Instanzen deutlich besser abschneidet. Dabei ist der Wert von 39 Tasks in der *Random*-Ausführung auf 16 reduziert. Diese Menge nimmt in beiden Fällen auf 11 beziehungsweise 5 Jobs weiter ab.

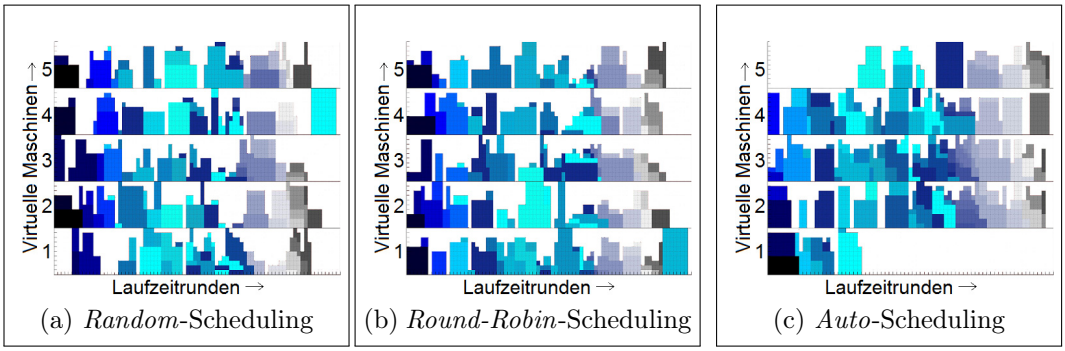
Die Analyse der Ergebnisse der entwickelten *Auto*-Ausführungsreihenfolge ergibt, dass sich ab drei genutzten Ressourcen sowohl die *Runtime VMs* als auch die *Rescheduled Tasks* wesentlich verbessern. Die Gesamtlaufzeit steigt ledig-

Scheduling-Strategie		**Anzahl VMs**							
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Random	Runtime VMs	228	244	264	312	375	438	518	584
	Rescheduled Tasks	106	88	62	48	39	19	16	11
Round-Robin	Runtime VMs	228	248	264	324	395	438	511	584
	Rescheduled Tasks	106	78	65	45	16	9	7	5
Auto	Runtime VMs	228	242	246	291	**294**	**294**	**294**	**294**
	Rescheduled Tasks	106	61	38	1	**0**	**0**	**0**	**0**

Grün: Optimales Kapazitätsvolumen erreicht

Tabelle 6.3: Simulationswerte für Suchmaschinenlast oder Batch-Job-Muster in Zeiten anliegender Last

lich auf 294 Runden. Das entspricht einem halb so großen Maximalwert im Vergleich zu den Resultaten der anderen beiden Scheduling-Strategien.
Auffallend ist zudem die rasante Minimierung der erneut zu bearbeitenden Anfragen. Der Wert fällt von anfänglich 106 Tasks rapide auf 61 und 38 Jobs beim Einsatz von einer bis drei Maschinen. Bei vier Instanzen ist lediglich ein Rescheduling nötig. Ab einer Ressourcenmenge von fünf ist keine Wiederbearbeitung mehr nötig. Analog zur bereits beschriebenen automatisierten Optimierung werden demzufolge ab diesem Kapazitätsvolumen keine weiteren Ressourcen hinzugeschaltet, da sie keinen Mehrwert bieten.
Abbildung 6.12 stellt den relevanten Ausschnitt bei fünf laufenden Maschinen für die drei Scheduling-Verfahren dar. Insgesamt sind die virtuellen Ressourcen bei allen drei Ausführungsreihenfolgen durchgehend recht stark ausgelastet. Dabei ist jedoch der entwickelte Algorithmus in Abbildung 6.12c etwas besser, da er eine effizientere Gruppierung der Tasks vornimmt. Das hat zur Folge, dass zu Beginn die fünfte virtuelle Maschine nicht benötigt wird. Zudem kann auch relativ schnell die zuerst gestartete Instanz wieder abgeschaltet werden. Die übrigen Ressourcen sind in der Lage, alle Anfragen zu bearbeiten.



(a) *Random*-Scheduling (b) *Round-Robin*-Scheduling (c) *Auto*-Scheduling

Abbildung 6.12: Lastdarstellung aktiver Maschinen in Such- oder Batch-Job-Systemen mit *Random*-, *Round-Robin*- und *Auto*-Scheduling

Die vierte Mustersimulation befasst sich mit fortlaufenden geringen Änderungen im Lastverlauf, wobei gelegentlich markant höhere Anfragemengen auftreten. Verwandte Beiträge zeigen, dass solche hervorstechenden Belastungsspitzen in regelmäßigen Arbeitslastabfolgen meist auf Servern zu verzeichnen sind, die der Verbreitung von Ereignissen dienen [96, 126].
Wie schon im zweiten Szenario erfolgt auch in der Umsetzung dieses Anwendungsfalls ab einer bestimmten Menge an Ressourcen keine weitere Wertveränderung der drei analysierten Scheduling-Verfahren. Die Anzahl liegt hier bei einem Umfang von sieben virtuellen Maschinen.
In den *Random*- und *Round-Robin*-Simulationen liegen die Gesamtrundenzeiten im identischen Wertebereich *Runtime VMs* = {127, 160, 240, 320, 400, 480, 560}. Dies ist abermals durch puren Zufall begründet. Dabei unterscheiden sich

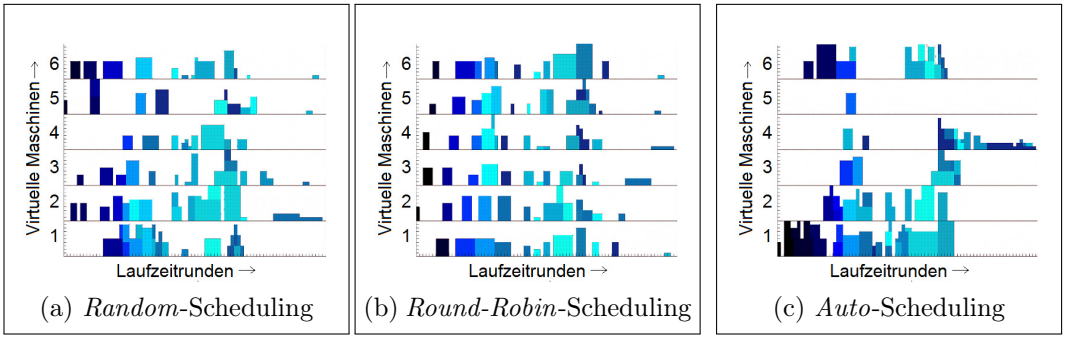
allerdings die Ergebnisse der *Rescheduled Tasks*. In der zufälligen Abarbeitung sind bei zwei Maschinen mit 33 Tasks sogar etwas weniger Wiedereinreihungen zu erkennen als bei der *Round-Robin*-Methode mit 35 Anfragen.
Ab drei Ressourcen erzielt das Rundlaufverfahren jedoch stets bessere Ergebnisse. Mit dieser Kapazitätenmenge werden zirka fünf Wiederaufnahmen weniger benötigt. Der beste erreichte Wert liegt mit 2 *Rescheduled Tasks* bei sieben virtuellen Instanzen. Für den *Random*-Ablauf kann lediglich auf 7 wieder aufzunehmende Jobs reduziert werden. Der Maximalwert ist in allen drei Algorithmen mit 48 erneuten Bearbeitungen bei nur einer Maschine gleich.
Wie in Ergebnistabelle 6.4 ersichtlich, ist der *Auto*-Algorithmus erneut fähig, die Menge an *Rescheduled Tasks* wesentlich schneller zu reduzieren. Die Werte belaufen sich für eine bis fünf Kapazitäten auf 48, 25, 14, 3 und 2 wieder einsortierte Anfragen. Ab sechs Maschinen müssen keine zusätzlichen Komponenten hochgefahren werden, da sich das bestmögliche Resultat bei 2 Wiedereinreihungen abzeichnet.
Diese Strategie kann folglich keine geringere Zahl an Wiedereinreihungen realisieren als der *Round-Robin*-Scheduler. Dennoch ist das *Auto*-Verfahren als effizienter einzuschätzen, da es dafür auf lediglich sechs Ressourcen angewiesen ist. Hingegen werden zur Reduzierung auf 2 *Rescheduled Tasks* beim *Round-Robin*-Verfahren mindestens sieben virtuelle Instanzen benötigt. In diesem Fall sind die Laufzeiten des gesamten Kapazitäten-Pools um 57 % höher als mit der *Auto*-Ausführung. Im optimierten Verfahren passt sich die höchste VM-Gesamtlaufzeit mit sechs geforderten Maschinen auf den deutlich geringeren Ergebniswert von 242 Runden an.
Die Lastgebirge der aktiven virtuellen Maschinen im Szenario der gelegentlich auftretenden Lastspitzen auf Servern von News-Seiten werden bei Verwendung der *Random*-, *Round-Robin*- und *Auto*-Scheduler in Abbildung 6.13 aufgezeigt.

Scheduling-Strategie		**Anzahl VMs**							
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Random	Runtime VMs	127	160	240	320	400	480	560	560
	Rescheduled Tasks	48	33	22	20	10	9	7	7
Round-Robin	Runtime VMs	127	160	240	320	400	480	560	560
	Rescheduled Tasks	48	35	17	13	7	5	2	2
Auto	Runtime VMs	127	146	182	204	219	**242**	**242**	242
	Rescheduled Tasks	48	25	14	7	3	**2**	**2**	2

Grün: Optimales Kapazitätsvolumen erreicht
Blassgrün / Grau: Keine Veränderungen für alle Scheduling-Verfahren

Tabelle 6.4: Simulationswerte für Lastspitzen bei News-Seiten-Servern

Konkret wird in diesen Darstellungen der Fall von sechs bereitgestellten Ressourcen repräsentiert. Dabei ist klar visualisiert, dass *Random* und *Round-Robin* trotz gleicher Gesamtlaufzeit unterschiedliche Ausführreihenfolgen umsetzen. Diese graphische Divergenz ist auch in der unterschiedlichen Anzahl an wiedereingereihten Tasks aufgrund von Überlastung begründet.



(a) *Random*-Scheduling (b) *Round-Robin*-Scheduling (c) *Auto*-Scheduling

Abbildung 6.13: Lastdarstellung aktiver Maschinen auf News-Servern bei der Ausführung des *Random*-, *Round-Robin*- und *Auto*-Schedulings

Mit dem fünften betrachteten Szenario wird ein Lastmuster im Umfeld der wissenschaftlichen Forschung umgesetzt. Abhängig von der anstehenden Projektdichte entstehen darin unterschiedlich ausgeprägte Arbeitslasten, die dann für eine gewisse Zeit linear verlaufen. Wenn weitere Forschungsaufgaben hinzukommen, liegt die Last fast augenblicklich und kontinuierlich gleichbleibend an, bis sie bei Projektabschluss wieder abfällt. Wie bereits in Abbildung 6.7 auf Seite 84 unten rechts dargestellt, variiert die durchschnittliche Nutzung im Unterschied zu den bisher vorgestellten Szenarien.

Die *Runtime VMs* des *Random*-Schedulers beläuft sich in dieser Simulation für eine bis vier virtuelle Maschinen auf Zeiten zwischen 203 und 336 Runden. Diese Werte liegen jeweils um etwa drei Zähler über den Laufzeiten des *Round-Robin*-Verlaufs mit einer Gesamtausführdauer von 203 bis 320 Runden im äquivalenten Ressourcenintervall. Bei fünf Instanzen ist die Messung von 385 Runden des *Random*-Verfahrens zufällig um zehn Runden besser als das Rundlauf-Scheduling. Daraufhin ergeben sich für beide Standards die gleichen Ergebnisse von 462, 539 und 616 Runden bei sechs, sieben und acht Maschinen.

Die Verwendung einer Maschine resultiert für alle Simulationen in einer Summe der Wiederaufnahmen von 75 Tasks. Bei Nutzung von acht Ressourcen ist jedoch im *Round-Robin* eine 70-prozentige Verbesserung im Vergleich zur *Random*-Bearbeitung zu verzeichnen.

Für das dazwischenliegende Kapazitätsvolumen erzeugt die *Round-Robin*-Methode durchgängig weniger *Rescheduled Tasks* als die Zufallsreihenfolge. Die Werte des *Random*-Schedulings entsprechen jeweils zirka den Vorgängerergebnissen des Rundlaufverfahrens. Bei der Zufallsbearbeitung wird folglich

jeweils eine Instanz mehr benötigt, um in etwa dieselben Ergebnisse an Wiedereinreihungen zu erzielen. Die spezifischen Resultate der Lastsimulation im Forschungsumfeld in Tabelle 6.5 belegen das mit deutlichen Werten.

Abermals zeigt das im Rahmen dieser Arbeit entwickelte Verfahren für automatische Skalierung und optimiertes Scheduling die besten Messresultate der Simulation. Die Gesamtlaufzeiten erstrecken sich von 203 Runden bei einer VM, über 259 Runden bei vier Ressourcen bis zu einer Zahl von 307 Rundenläufen bei sieben virtualisierten Komponenten. Daraufhin ist auch bei Zuschaltung zusätzlicher Ressourcen keine Verbesserung an *Rescheduled Tasks* zu erreichen. Folglich stoppt der Entscheidungsalgorithmus weitere Skalierungsprozesse mithilfe der bereits erläuterten Logik.

Besondere Aufmerksamkeit verdient der Schritt der Hochregulierung von sechs auf sieben Instanzen. Obwohl die *Runtime VMs* lediglich um vier Punkte auf 307 Runden steigt, findet gleichzeitig ein Halbieren der *Rescheduled Tasks* von 8 auf 4 Jobs statt. Abbildung 6.14 stellt einen anschaulichen Vergleich der drei Scheduling-Strategien dar. Aus den Lastgebirgen der simulierten Forschungsumgebungen ist ersichtlich, dass insgesamt sieben virtuelle Instanzen zur Bewältigung der Arbeitslasten benötigt werden.

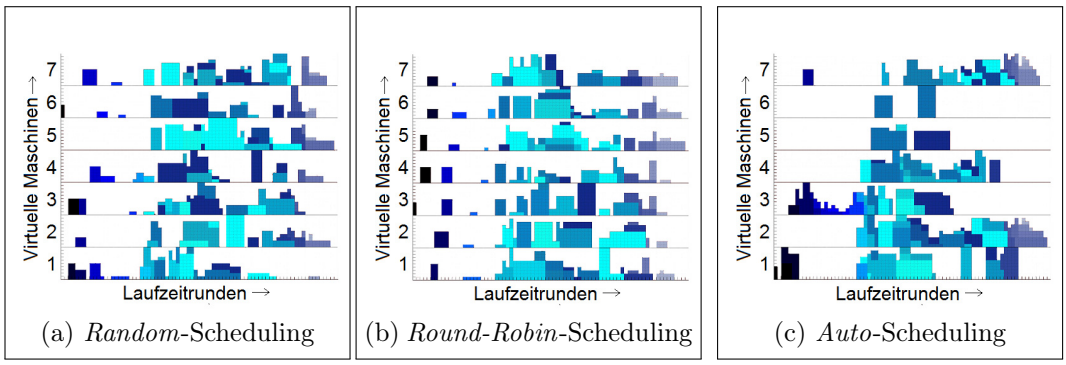
Wie bereits erläutert, variieren in diesem Muster die zugehörigen Arbeitslasten erheblich. Zu Beginn stehen bei der konkreten Simulationsimplementierung wenige Jobs an. Diese Menge nimmt mit der Zeit enorm zu. Das erklärt zum einen die gestückelten Farbblöcke zu Anfang der Gesamtlaufzeit.

Anderseits erschließen sich damit die größeren gleichfarbigen Blöcke, wie beispielsweise in Abbildung 6.14a auf der fünften Maschine zu erkennen: In kurz aufeinanderfolgenden Zeitpunkten laufen so viele Anfragen im System auf, dass sie der Standard-Algorithmus auf dieser Ressource nicht parallel bearbeiten kann. Entsprechend häufig werden die Requests erneut aufgeschoben. Sie müssen in einer weniger belasteten Zeit bearbeitet werden.

Scheduling-Strategie		**Anzahl VMs**							
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Random	Runtime VMs	203	236	270	336	385	462	539	616
	Rescheduled Tasks	75	63	48	39	31	24	20	20
Round-Robin	Runtime VMs	203	234	264	320	395	462	539	616
	Rescheduled Tasks	75	61	35	28	24	16	13	6
Auto	Runtime VMs	203	227	231	259	271	303	**307**	**307**
	Rescheduled Tasks	75	43	28	18	10	8	**4**	**4**

Grün: Optimales Kapazitätsvolumen erreicht

Tabelle 6.5: Simulationswerte schwankender Arbeitslast im Forschungsumfeld



(a) *Random*-Scheduling (b) *Round-Robin*-Scheduling (c) *Auto*-Scheduling

Abbildung 6.14: Lastdarstellung aktiver Maschinen im Forschungsumfeld bei der Ausführung des *Random*-, *Round-Robin*- und *Auto*-Schedulings

Abschließend werden in den folgenden Schaubildern alle bereits analysierten Simulationsresultate gegenübergestellt. Verschiedene Kurven visualisieren die gegenläufigen Ergebnisse der *Runtime VMs* und der *Rescheduled Tasks* für die Algorithmen des *Random*-, *Round-Robin*- und *Auto*-Schedulings. Die Legende für alle erzeugten Graphen ist in Abbildung 6.15 dargestellt.
Zusammenfassend ergibt sich, dass über alle Simulationen die Kurven der Ressourcenlaufzeiten stetig steigen. Dies ist durch die Akkumulation der Runden für das jeweilige Maschinenvolumen begründet. Infolgedessen entstehen mit wachsender Gesamtlaufzeit der virtuellen Komponenten zunehmend höhere Kosten bei ihrer Bereitstellung. Die beim Cloud-Anbieter zu zahlende Gebühr pro aktiver Instanz ist dafür die ausschlaggebende Ursache.
Auf der anderen Seite zeigen die dargestellten Werte der *Rescheduled Tasks*, dass mit größer werdenden *Runtime VMs* die Wiedereinreihungen stetig abnehmen. Diese Folge ist logisch nachvollziehbar, da mit größerer Kapazitätsmenge mehr Anfragen parallel abgearbeitet werden können. Dadurch werden die einzelnen zu bedienenden Tasks seltener erneut einsortiert.

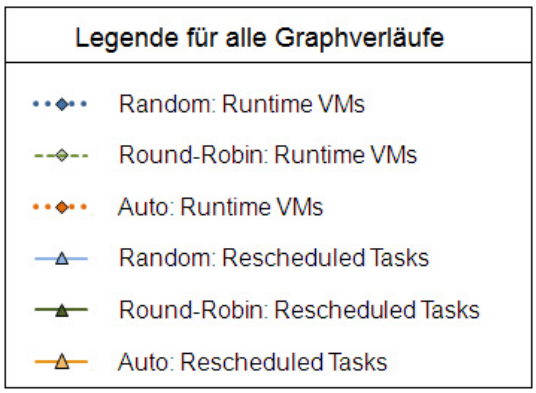


Abbildung 6.15: Legende aller Arbeitslastszenarien

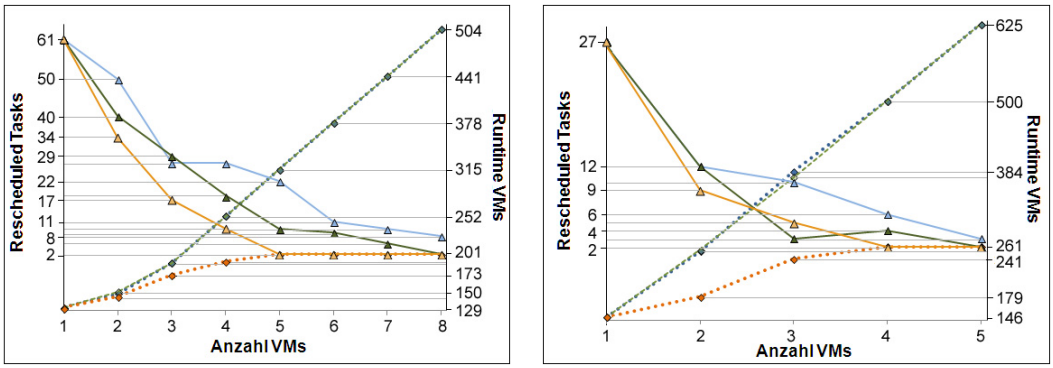
Im ersten Szenario wird ab fünf virtuellen Maschinen mit den korrespondierenden zwei Wiedereinreihungen keine weitere Reduktion der *Rescheduled Tasks* erzielt. Das zeigt der gelb dargestellte Wertverlauf der *Rescheduled Tasks* bei Verwendung des *Auto*-Schedulings in Abbildung 6.16a. Allerdings steigt ab diesem Punkt auch die rot dargestellte Gesamtlaufzeit der Maschinen nicht weiter an. Bereits ab zwei Ressourcen ist sie merklich besser als die *Runtime VM*-Kurven des *Random*- und des *Round-Robin*-Verfahrens.

Wie bereits erläutert, ergibt sich in der Umsetzung des zweiten Szenarios ab fünf Instanzen der konstante Fortlauf der beiden Parameter der Laufzeit und der Wiedereinreihungen in *allen drei* Strategien. Abbildung 6.16b weist bereits bei einem Umfang von vier Ressourcen keinen zusätzlichen Laufzeitanstieg auf. Dies legt die orange gestrichelt dargestellte Linie der *Auto Runtime VMs* offen. Sie bleibt ab diesem Wert konstant. Demgemäß werden die Kurvenentwicklungen lediglich bis zu dieser Maschinenzahl dargestellt.

Abbildung 6.17a stellt das Lastszenario der Suchmaschinen dar. Für die *Auto*-Abarbeitung genügen vier Ressourcen zur Realisierung der geringstmöglichen Menge an Neueinsortierungen. Besonders auffällig ist der erhebliche Sprung in der *Rescheduled Tasks*-Kurve bei der Erhöhung von drei auf vier Maschinen. Seine Ursache liegt in der enormen Reduzierung erneuter Einsortierungen von 38 Jobs auf lediglich eine wiederzubearbeitende Anfrage.

Beim Lastmuster auf den Servern von News-Seiten ergibt sich ab sieben virtuellen Instanzen für *alle* Verfahren keine weitere Änderung in den Ergebniswerten. Analog sind in Abbildung 6.17b die Simulationsresultate lediglich für diese entscheidende Komponentenzahl visualisiert.

Die fünfte Lastsimulation im Forschungsumfeld von wissenschaftlichen Laborszenarien ist in Abbildung 6.17c präsentiert. Am parallelen Verlauf der grünen und der blau dargestellten Kurven ist erkennbar, dass zur Erzielung ähnlicher Resultate der *Rescheduled Tasks* beim *Random*-Verfahren je eine virtuelle Maschine mehr nötig ist als mit dem *Round-Robin*-Scheduler.

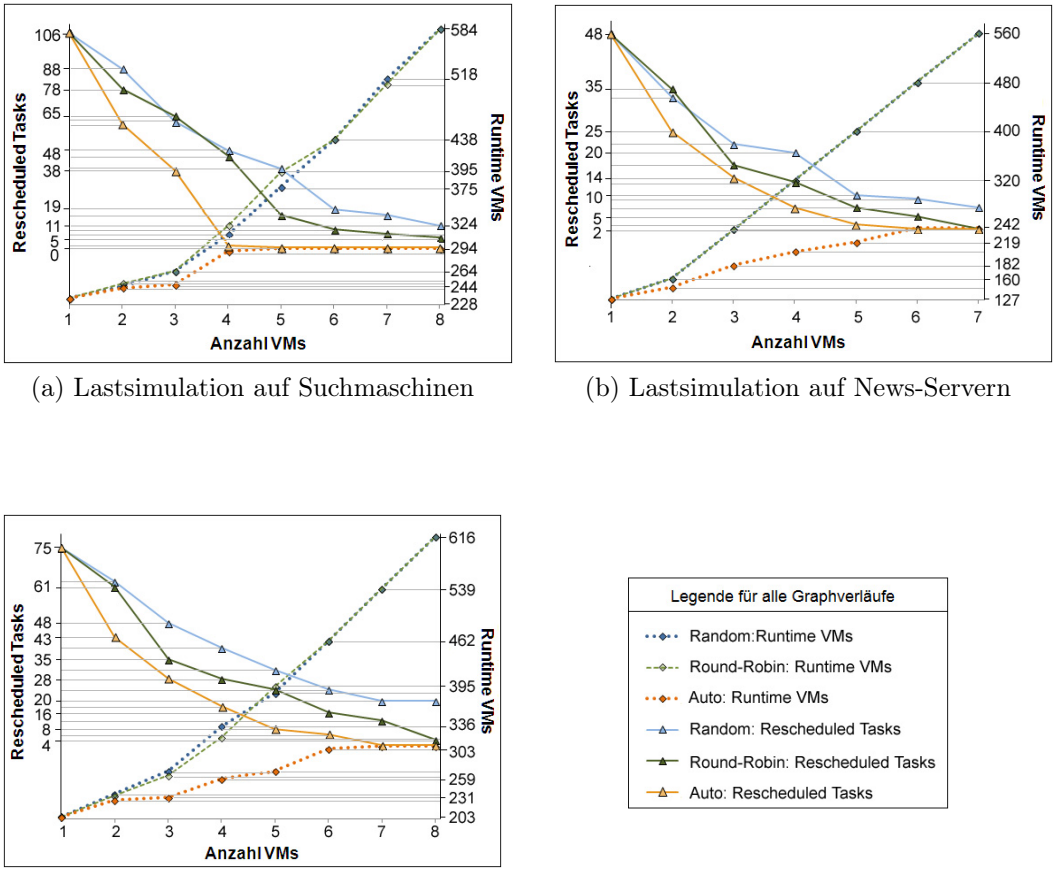


(a) Lastsimulation auf Email-Servern (b) Lastsimulation auf Ticket-Servern

Abbildung 6.16: Scheduling-Simulation für die Arbeitslastszenarien 1 und 2

Insgesamt schneidet die hier entworfene Bearbeitungsmethode der Jobs besser ab als die Strategien der Standardverfahren. Einerseits ist dies an den deutlich schneller fallenden und stets niedrigeren *Rescheduled Tasks*-Kurven des *Auto*-Algorithmus abzulesen. Zum anderen offenbart sich auch über alle Grafiken hinweg, dass die Kurvenverläufe der *Auto Runtime VMs* merklich gemäßigter ansteigen als die analoge Visualisierung der anderen zwei Abarbeitungen. Dabei rangieren die Ergebniswerte durchgehend unter den Resultaten des *Random*- und des *Round-Robin*-Schedulings.

In allen Simulationsergebnissen ist erkennbar, welche Anzahl an laufenden Maschinen optimal für die implementierten Anwendungsfälle ist. Dies ist in den Darstellungen immer genau der Punkt, ab dem sich die gelbe Linie der *Auto Rescheduled Tasks* und die rot gepunktete Linie der *Auto Runtime VMs* überlagern. Auf dieses Optimum skaliert sich die Systemumgebung mit dem *Auto*-Verfahren jeweils selbstständig. Das bietet einen enormen Vorteil gegenüber den herkömmlichen *Random*- und *Round-Robin*-Umsetzungen.



(a) Lastsimulation auf Suchmaschinen

(b) Lastsimulation auf News-Servern

(c) Lastsimulation im Forschungsumfeld

(d) Legende aller Lastsimulationen

Abbildung 6.17: Scheduling-Simulation für die Arbeitslastszenarien 3-5

6.7 Zusammenfassung

In den gängigen Cloud-basierten Umgebungen können die zugehörigen Dienste theoretisch eigenständig und losgelöst bereitgestellt werden. Häufiger sind jedoch Service-Zusammenschlüsse zu einem ineinandergreifenden Gesamtkonstrukt zu finden. Diesem liegt dann ein vielschichtiger Aufbau der Infrastruktur zugrunde. Die einzelnen Dienste einer Ebene werden dabei oft simultan von verschiedenen Clients angesprochen.

Infolgedessen stellt sich bei eingehender Betrachtung dieses komplexen Umfelds heraus, dass unübersichtliche Abhängigkeiten unter den beteiligen Systemteilen bestehen. Die notwendige Überwachung der Elemente ist enorm aufwändig und führt zu hohen Kosten auf oberster Management-Schicht. Damit entsteht die Frage, wie diese Services verwaltet werden sollten, um möglichst wenig administrativen Overhead zu verursachen.

In diesem Kapitel wurde als möglicher Lösungsweg eine Erweiterung des Ressourcenmanagement-Protokolls präsentiert. Durch diese Realisierung können abhängige Cloud-Instanzen proaktiv und automatisiert verwaltet werden. Die Auslastung jeder involvierten Komponente muss dafür nicht überwacht werden. Der Service-Load-Manager setzt einen Selbstkalibrierungsmechanismus ein, der die automatisch erkannten Ressourcenrelationen in einem Graphen abbildet. Im laufenden Betrieb ist den Clients dann automatisiert genau die nötige Anzahl an Ressourcen zur Verfügung zu stellen. Das Ressourcenregelverfahren des Managers minimiert den Skalierungaufwand und ermöglicht eine optimierte Ausführreihenfolge der lastverursachenden Anfragen.

Die Ergebnisse der analytischen Überprüfung verdeutlichen die Möglichkeit der automatischen Größenregulierung durch den Algorithmus. Die Cloud-Umgebung skaliert sich selbstständig auf das Auslastungsoptimum hin.

Weiterhin wurden Simulationen für fünf Szenarien durchgeführt, die verschiedene Lastmuster heutiger Cloud-Umgebungen implementieren. Die Evaluation erfolgte anhand einer Gegenüberstellung des vorgestellten Verfahrens mit den geläufigen Standardstrategien des *Random-* und *Round-Robin*-Schedulings.

Dabei belegen die Simulationsresultate, dass der in dieser Arbeit entworfene Algorithmus bezüglich der Anfragenverarbeitung stets am besten abschneidet. Außerdem skaliert sich das System im Falle der *Auto*-Methodenrealisierung eigenständig auf ein Optimum an laufenden Instanzen. Tatsächlich ergibt sich bei gleicher Anzahl virtueller Maschinen mit der entwickelten Strategie eine erhebliche Reduktion der erneut zu bearbeiteten Tasks. Zudem zeigt sich im Durchschnitt eine halb so große Gesamtlaufzeit dieser Komponenten im Vergleich zu den Standardverfahren. Insgesamt kann das Ressourcenmanagement mit dieser Erweiterung stark verbessert werden.

7 Ressourcenfestlegung mit Nutzerdaten sozialer Netze

Mit der bisher geschaffen Lösung werden Anfragen durch effizient und selbstregulierend Vorab-Reservierungen bearbeitet. Dieses Kapitel beschäftigt sich mit Situationen, in denen aufgrund fehlender Bedarfsanmeldung die Festlegung solch optimierter Schedules nicht möglich ist. Grundlegend gewährleistet die Verwendung hochskalierbarer Server leider nicht automatisch ein performantes Applikationsverhalten. Ursächlich sind die Anwendungszugriffe auf verschiedene Daten. Die nötige Netzübertragung dieser Informationen führt zu Verzögerungen und wird als Netzwerk-Latenz bezeichnet [180].

Die einfachste und gängigste Variante reservierungsfreier Anfragenverteilung gelingt mithilfe von Lastbalancierern per Domain Name Service, kurz DNS. Der DNS-Server entscheidet, zu welcher Instanz der anfragende Nutzer verbunden wird – im simpelsten Fall per Round Robin. Der Client wird so an einen freien, eventuell sehr weit entfernten Server weitergeleitet. Dies verursacht unnötig hohe Latenzzeiten durch längere Übermittlungswege.

Zudem entstehen im Software-as-a-Service-Bereich unvorhersehbare Verzögerungen, da die Startzeit weiterer Instanzen auf tiefer Dienstebene dem Nutzer unbekannt ist. Bei der Bearbeitung steigender Last kommt es teilweise zum Reaktionsverzug von bis zu dreizehn Minuten [105]. Dies ist abhängig von der Dienstkomplexität und Charakteristik der genutzten Cloud-Plattform [165].

SaaS-Anwendungen gewinnen zunehmend an Bedeutung, wie am steigenden deutschen Marktvolumen auf rund eine Milliarde Euro abzulesen ist [50]. Angesichts dessen muss auch ohne Reservierung definierbar sein, *wann wie viel* Kapazität nötig ist. Ebenso muss eine Lösung gefunden werden, *wo* die Ressourcen verfügbar sein sollen. Um die Angebote auf den Plattformen effizient zu konfigurieren, muss eine Prognose des lokalspezifischen Bedarfs erfolgen.

Nachfolgend wird auf bestehende Forschung zu diesem Thema eingegangen. Dabei werden Probleme bei der Lastbalancierung unter Berücksichtigung des hauptsächlichen Lastaufkommens aufgeführt. Die angesprochene Latenz lässt sich reduzieren, indem Anwendungen und Daten näher zu den Nutzern gebracht werden. Deshalb wird ein Ansatz vorgestellt, der die Last unter Berücksichtigung der Ressourcenlokalität automatisiert verteilt. Er erkennt wachsende Arbeitslast mittels sozialen Netzwerkdaten zu Nutzerinteressen. Dann wird die Lokalität mit dem Hauptressourcenbedarf durch die geclusterten Aufenthaltsorte der Clients bestimmt. Dieses Kapitel schließt mit der Erläuterung der Simulationen und wichtigen Evaluationserkenntnissen ab.

7.1 Verwandte Arbeiten zur vorausschauenden Ressourcenbereitstellung

Zur möglichst exakten und vorausschauenden Approximation der benötigten Kapazitätsmenge ist die Prognose zukünftiger Last auf den zugehörigen Ressourcen entscheidend. Allerdings ist diese ebenso schwierig wie essentiell, um die Quality-of-Service für die Nutzer zu garantieren [101]. Auch Chen et al. [34] schlagen vor, kommende Last vorherzusagen und diese Informationen im Skalierungsprozess zu nutzen. Dafür sammeln sie im Rechenzentrumsbetrieb Daten über die Systemauslastung. Der Lastverlauf wird mit einer Auswahl an Basisfunktionen und deren Linearkombinationen mittels Regressionsanalyse angenähert. Damit ist die Wahrscheinlichkeit eines Anstiegs abschätzbar.

Wie vorab erwähnt, gibt es Arbeiten, die anstelle von statistischen Analysen eine Mustererkennung durchführen. Dabei ist zu berücksichtigen, dass man aufgrund eines Musters der Vergangenheit, nicht immer auf den gleichen aktuell zu erwartenden Lastanstieg schließen kann.

Gmach et al. [160] bieten eine automatische Verteilungslösung. In ihrem System werden Anfragen bei Überlastung fair aufgeteilt, indem sie auf die am wenigsten beanspruchten Server übertragen werden. Für periodisch auftretende Lastspitzen werden zeitnahe Vorhersagen genutzt. Bei unvorhergesehenen Änderungen stellt ein Administrator Anweisungen beispielsweise zur Instanzenauslastung bereit. Allerdings werden dabei erhebliche Verzögerungen für den Nutzer in Kauf genommen. Sie entstehen durch das Weiterleiten der Aufträge über weite Entfernungen. Außerdem ist das Anwachsen der Systemlast nur sehr zeitnah zu erkennen. Da jedoch das Bereitstellen von Ressourcen bis zu einer Viertelstunde dauern kann, ist eine kurzfristige Zuschaltung nach Beginn des Anstiegs häufig unzureichend [105].

Wird bei Lastverteilungsmechanismen das Best-Effort-Prinzip als maßgebliche Basis angenommen, ist mit pauschaler minimalistischer Qualitätszusicherung der Cloud-Struktur und Internet-Erreichbarkeit zu rechnen. Der Forschungsfokus der Lastprognose kann sich auf die Nutzersicht verlagern [109]. Die Bestimmung, wie die Nutzer das Netzwerk – also einen entscheidenden Teil der Cloud-Umgebung – verwenden, ist entscheidend. Genau dies verursacht nämlich Arbeitslast auf der zugrundeliegenden Infrastruktur. Diese Informationen sind prinzipiell für Skalierungsprozesse nutzbar.

In dem Kontext präsentieren Phillips et al. [139] interessante Ergebnisse, wie Anwender das Internet zur Informationsbeschaffung durchsuchen. Ihre Erkenntnisse sind besonders aufschlussreich bezüglich der Analyse, wie sich durch die Kommunikationsverschiebung auf virtuelle Kanäle über verschiedene Geräte neue Formen zur Organisation und Verwaltung von Informationen bilden. Dabei nehmen zunehmend soziale Netzwerke wie *Facebook* und virtuelle Kommunikationswege beispielsweise über *Twitter* einen beachtlichen Stellenwert ein [51, 172]. Hierin ist der Informationsfluss enorm schnell. Zudem sind Zusammenhänge unter einzelnen Plattformen erkennbar [96], [185].

Neben dem kommunikativen Austausch gewinnen soziale Netze immer mehr für zeitkritische Geschehnisse wie Notfälle oder Krisen an Bedeutung [178]. Diese treten zwar unvorhersagbar auf, doch fanden Yang und Leskovec [184] mit ihren Studien über Informationsflüsse im Internet heraus, dass die Netzlast nach relevanten Ereignissen zunimmt. Die in Abbildung 7.1 definierten Muster skizzieren die soziale Informationsausbreitung.

Die Muster M_1 und M_2 verlaufen recht symmetrisch, während in M_3 ein äußerst rasanter Anstieg der Popularität zu beobachten ist. Ansonsten weisen die ersten drei Muster nur geringfügige Unterschiede auf. In den Mustern M_4 bis M_6 sind mehrere Lastspitzen erkennbar. M_4 zeigt nach dem größten Ausschlag der Last noch einen weiteren geringeren. In M_5 ist eine niedrige Spitze vor dem Maximalanstieg zu sehen. Im Muster M_6 sind nach der höchsten Lastspitze noch mehrere, niedrigere Schwankungen festzustellen. Bei allen Mustern nimmt die Popularität zu Anfang erheblich zu. Die jeweilige Kurvensteigung erstreckt sich über zirka 20 Stunden. In dieser Periode kommt sie einem exponentiellen Verlauf am nächsten. Anhand dessen wird ein Modell entworfenen, das die zukünftige Popularität vorhersagt.

Die Popularitätskurve in Abbildung 7.2 zeigt, welche Knoten zu einem Zeitpunkt von einem Ereignis wissen und wie sich weitere Verläufe schätzen lassen. Irrelevant ist, welcher Client explizit im nächsten Schritt informiert wird. Vielmehr dient das Modell der Beurteilung, wie viele Knoten global über einen Punkt benachrichtigt sind [183]. Das ermöglicht eine theoretische Lastvorhersage mittels zeitnaher Datenextraktion.

Die Nutzer verwenden verschiedenste Anwendungen zum Informationsgewinn und zur Kommunikation. Auf Dienstseite müssen Performanzwerte analysiert werden, da sie indirekt durch den Nutzer bestimmt sind. Performanz beschreibt die Leistung des Systems im Sinne der verrichteten Arbeitsmenge [166].

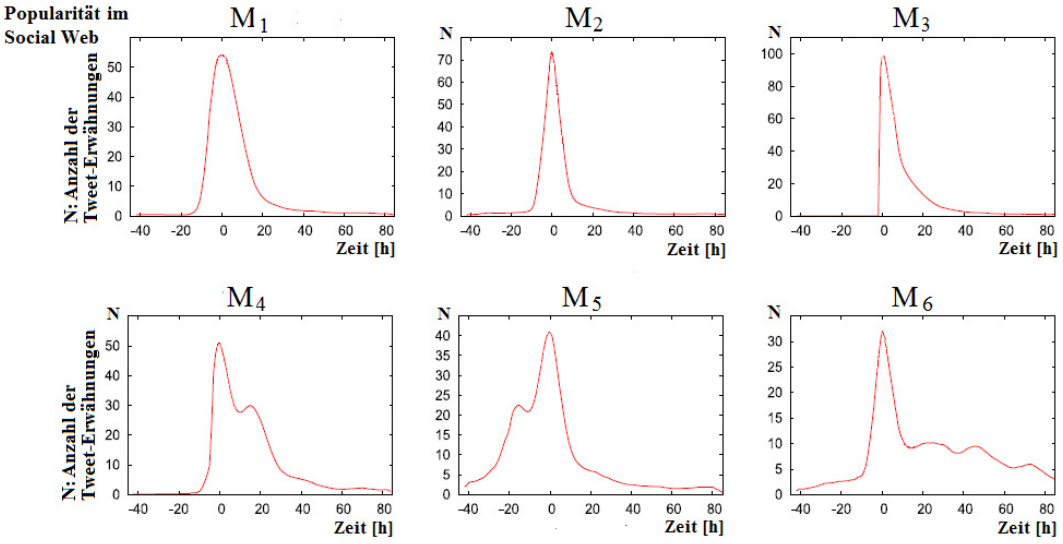


Abbildung 7.1: Muster für Popularitätsverläufe in sozialen Netzwerken [184]

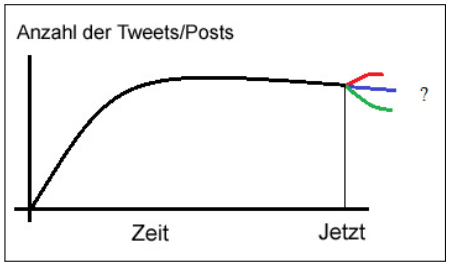


Abbildung 7.2: Popularitätskurve nach Leskovec [96]

Nach Stang [167] beeinflussen unterschiedliche Systemelemente die Performanz der Umgebung. Unter anderem zählen hierzu neben CPU-Rechenstärke, RAM, Plattenspeicher, Plattenzugriffszeit und Stärke der Server-Client-Netzverbindungen auch der allgemeine Datendurchsatz, die durchschnittliche Wegstrecke zwischen Server und Client sowie die Komponentenanzahl.

Ein Cloud-System ist nur wirtschaftlich praktikabel, wenn es viele Nutzer bedienen kann. Dementsprechend ist der Drang nach Performanzverbesserung in kommerziellen IT-Produkten enorm groß. Immer neue Lösungen werden integriert. Applikationen wie *Skype* und die Java-Spezifikation *JXTA* verdanken ihre Leistungsfähigkeit beispielsweise dem Konzept der Super-Peers [102, 124].

Generell fungiert ein Super-Peer ähnlich zu einem zentralen Server für eine Client-Untermenge. Die Super-Peers sind untereinander und mit den anderen Peer-Knoten verbunden. Sie leiten Nachrichten über dieses Overlay-Netz und geben Anfragen im Namen ihrer Clients weiter oder beantworten sie direkt selbst [182]. Aufgrund ihrer überdurchschnittlich hohen Bandbreite und geringeren Latenzanfälligkeit eignen sie sich gut für infrastrukturelle Aufgaben des Netzwerks. In einem *Skype*-Netz wird beispielsweise ein eingebundener Knoten zu einem Super-Peer, wenn er eine äußerst stabile Netzanbindung über eine definierte Zeit aufweist. Durch Super-Peers kann *Skype* so eine höhere Dienstqualität als zentrale Ansätze anbieten [163].

Heutige Cloud-Anwendungen werden häufig von mobilen Endgeräten genutzt. Da diese allerdings anfällig für Schwankungen im Funkverkehr sind, lässt sich das vielversprechende Super-Peer-Konzept kaum umsetzen. Soll für eine Smartphone-Applikation eine Super-Peer-ähnliche Infrastruktur aufgesetzt werden, muss der Anbieter selbst die korrekte Konstruktion sicherstellen. Das ist angesichts des enormen Aufwands oft nicht gewollt [148]. Deshalb beinhaltet der folgende Ansatz einen Platzierungsmechanismus, der nicht durch die Bandbreite einzelner Teilnehmer bestimmt ist. Vielmehr wird die erwartete Nutzerdichte im Netzwerk abgeschätzt und zur Organisation genutzt.

Weitere wissenschaftliche Untersuchungen zur adaptiven Dienstbereitstellung in Cloud-Systemen befassen sich zunehmend mit entstehenden Kosten und der Realisierung von Selbstinstandhaltung der Services [54]. Allerdings werden aufgrund der Konzentration auf zugrundeliegende Infrastrukturen und Anbieter meist die eigentlichen Nutzer sowie das Software-Level vernachlässigt.

Aufgrund großer Distanzen zwischen Server-Lokalität und Nutzer kann die Anwendungsbedienung negativ beeinflusst werden. Folglich ist eine flächendeckende Verbreitung der Applikation sinnvoll. Dafür bieten alle großen Cloud-Plattform-Anbieter unterschiedliche Rechenzentren und damit verschiedene Hosting-Lokalitäten für die Software-Anwendung [180]. Durch die Verteilung der Cloud-Dienste über mehrere Hosting-Orte sind theoretisch neben der Ausfallsicherheit auch kurze Antwortzeiten realisierbar.

Die gängige regelbasierte Instanzenverwaltung über Lastparameter und Schwellwerte umfasst selten Lokalitätsinformationen. Zur effizienten Ressourcenverwaltung sind allerdings auch Informationen über den Ort der Lastkonzentration und den Server-Standort des SaaS-Angebots zu integrieren.

Dieser Ansatz ist bereits aus anderen Wissenschaftsbereichen bekannt, wo Mobilität von Computing-Entitäten betrachtet wird. So werden in Peer-to-Peer-Netzen oder bei der Software-Agenten-Migration routende oder koordinierende Einheiten bezüglich der aktuellen Last und Nutzung platziert [79, 182].

Da sich dieser Beitrag allerdings auf die Lastbalancierung von Cloud-Angeboten bezieht, wird ein Mechanismus vorgestellt, der zur Prognose Daten aus sozialen Web-Anwendungen verwendet. Diese Applikationen bieten standardmäßig REST-APIs an, mit denen Nutzerinformationen extrahierbar sind. Damit sind die verschiedenen Rechenzentren eines Anbieters bestmöglich nutzbar und kostengünstige Auslastungen erzielbar.

7.2 Ressourcenmanagement mit sozialen Netzwerk-Daten

Wenn keine aktive Reservierung vorgenommen wird, muss die aufkommende Last beim Cloud-Dienst adäquat prognostiziert werden. Die im Rahmen dieser Arbeit entwickelte automatisierte Methode realisiert das dementsprechende Ressourcenmanagement in Lastsituationen. Einen groben Systemüberblick bieten Schwanengel et al. in [154].

Wie vorangegangene Forschungsarbeiten offenbaren, sind Lastvorhersagen nach wie vor risikobehaftet. Dies liegt daran, dass kaum zu präzisieren ist, *wann, wo* und *wie viele* Nutzer den bereitgestellten Dienst anfragen werden. Wenn keine selbstständige Bedarfsanmeldung durch die Komponenten erfolgt, ist es demnach schwer, rechtzeitig, effizient und automatisch einer potenziellen Systemüberlastung entgegenzusteuern.

Eine mögliche Lösung besteht in der Integration von Nutzerinformationen. Mit diesen ist identifizierbar, ob zusätzliche Kapazitäten zu einem bestimmten Zeitpunkt benötigt werden. Nach Yang et al. [184] ist davon auszugehen, dass die nutzerseitige Interessenbekundung in sozialen Netzen und die tatsächliche Anfrage einer semantisch korrespondierenden Anwendung korrelieren. So lässt

sich ein Zusammenhang zwischen der Popularität einer Anwendung und der lokalspezifischen Auslastung der hostenden Server herstellen.

Diese Tatsache ist vor allem für SaaS-Betreiber attraktiv. Wenn ihre auf der PaaS- oder IaaS-Schicht aufsetzende Software bei anderen Cloud-Anbietern gehostet wird, haben die SaaS-Betreiber kaum Kontrolle über die zugrundeliegende Infrastruktur. Folglich müssen sie im Interesse ihrer eigenen Rentabilität auf anderen Wegen Einfluss auf die involvierten Systemkomponenten nehmen. Um einen größtmöglichen Teil ihrer Nutzer bestmöglich zu bedienen, kann ein SaaS-Anbieter abhängig vom gewählten PaaS- oder IaaS-Betreiber auf verschiedene Hosting-Lokalitäten zugreifen. Beispielsweise gibt es in den USA zahlreiche Hosting-Orte an der West- oder Ost-Küste. Daneben gibt es auch verschiedene Rechenzentren in Europa, wie in Irland oder Amsterdam, sowie in Asien, z.B. in Hongkong oder Tokyo, und im australischen Sydney.

Mit diesem Wissen gilt es ein Mittel zu finden, das eine ortsbezogene Wahrscheinlichkeitsprognose für die künftige Lastentwicklung einer betrachteten SaaS-Anwendung realisiert. Ziel ist es, eine komplexe Aussage zu formulieren wie: „In den nächsten fünf Stunden wird sich die Anzahl der Nutzer, die auf den im Amsterdamer Rechenzentrum gehosteten Musikanalyse-Service zugreifen, mit einer Wahrscheinlichkeit von 80 % verdoppeln“. So sollen einerseits minimale Kosten bei geringem Ressourceneinsatz entstehen. Andererseits soll hochverfügbare Diensterbringung durch frühzeitige Instanzenzuschaltung bei vermuteten Lastanstiegen erzielt werden.

Wenn der Anbieter die Hosting-Lokalitäten über ein Land verteilt, gibt es triviale Indikatoren, die einer lokalspezifischen Lastprognose dienen. Auf zugrundeliegende Fakten wie die Schaltung nach der jeweiligen Tageszeit und der korrespondierenden Zeitzone ist Verlass. Außerdem sind statische Einflüsse für eine Bestimmung von Nutzerballungen anhand von Statistiken über Verkaufszahlen von Smartphones oder durch Marktanteile der Mobilfunkbetreiber identifizierbar. Ähnliches gilt für die Ermittlung von Ballungszentren nach Bevölkerungszahlen.

Wie bereits erläutert, sind bis heute Überlastsituationen nur durch manuelle Nutzereingriffe mit selbst definierter Anweisungslogik zu regulieren. Damit beim Eintritt unvorhergesehener Ereignisse die Systemperformanz nicht gefährdet wird, muss der SaaS-Anbieter auch weitere Parameter überwachen. Um dabei die Systemskalierung voll automatisiert zu lösen, bestimmt der entwickelte Algorithmus die zu erwartende Nutzermenge unabhängig von angekündigten oder noch nicht bekannten Geschehnissen.

Grob gesagt wird in der Prognose die anstehende aggregierte Last auf einem Server, der eine spezifische Applikation bereitstellt, geschätzt. Dieser Vorhersagewert basiert auf der Menge von Tweets zu einem bestimmten Thema, das mit dieser Anwendung korreliert. Generell ist es auch möglich, Informationen jeder anderen REST-basierten Plattform zu nutzen. In dieser Umsetzung agiert *Twitter* als erster Indikator für Nutzerinteressen, da die Reaktion der Internetnutzer hierüber sehr zeitnah erfolgt. Zudem lässt sich von einer großen Menge

an Tweets zu einem bestimmten Thema gut auf ein verstärktes anstehendes Nutzungsverhalten der assoziierten Applikation schließen [94].
Bei der Analyse dieser Informationen wird überprüft, inwieweit die Tweets mit der Server-Last korrelieren. Nach der Korrelationsuntersuchung wird ein Wert angegeben, mit welcher Wahrscheinlichkeit ein Lastanstieg zu erwarten ist. Das mögliche Volumen der Arbeitslast auf den Servern sowie der Zeitpunkt ihres Auftretens sind basierend auf den historischen Verlaufsdaten vorhersagbar.
Die Lastentwicklung kann in den verschiedenen Rechenzentren, die dieselbe Applikation anbieten, erheblich variieren. Deshalb werden im zweiten Schritt Informationen über den Ort der Lastkonzentration in den Prozess integriert. So wird die Entscheidung erleichtert, wo zusätzlich benötigte Ressourcen zu starten sind. Diese Daten werden wiederum aus Nutzerauskünften über ihre Aufenthaltsorte gewonnen. Dabei werden die nötigen Informationen aus sozialen Web-Anwendungen wie Geocaching- und Geotagging-Diensten mithilfe der zugehörigen Programmierschnittstellen, also der REST-APIs, extrahiert.
Daraufhin kann mit dem aggregierten Anfragevolumen ein Ballungsort von Nutzern identifiziert werden und die bevorzugte Hosting-Lokalität gewählt werden. Mit diesen in Abbildung 7.3 skizzierten Vorgängen kann in Überlastsituationen eine frühzeitige Bereitstellung und ortsnahe Zuteilung vor der eigentlichen Nutzung und ohne aktive Reservierung erfolgen. In den folgenden Abschnitten werden der Algorithmus und die Darstellung näher erläutert.

7.2.1 Identifikation von steigender Arbeitslast

Der entworfene Algorithmus ermittelt eventuelles Wachstum im künftigen Lastaufkommen eines Dienstes. Um einen plausiblen Wert des diesbezüglichen Ausmaßes zu bestimmen, analysiert er über Twitter das weltweite Nutzerinteresse an der betrachteten SaaS-Anwendung.
Dann erfolgt die Verlaufsabschätzung der damit verbundenen Last auf der zugrundeliegenden Infrastruktur. Eine Warteschlange speichert alle auf den Dienstkomponenten anstehenden Anfragen. Die Last bezieht sich auf die Warteschlangenlänge der Ressourcen, die das SaaS-Angebot bedienen [3].

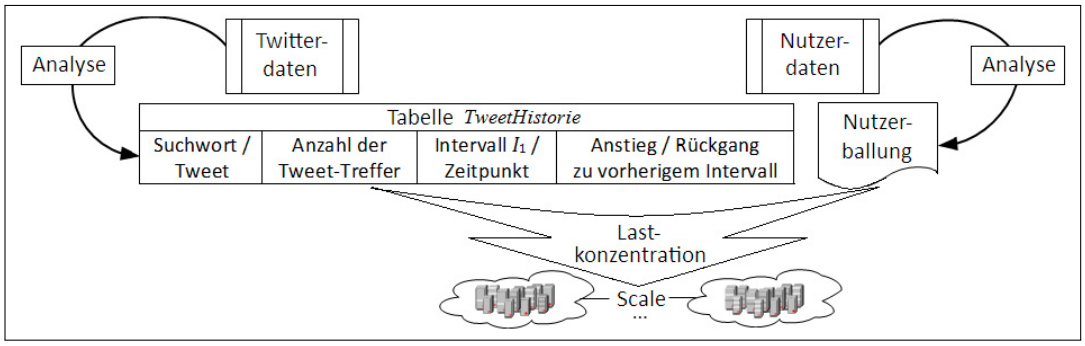


Abbildung 7.3: Ablauf der Schätzung des Ressourcenbedarfs

Im ersten Schritt wird mithilfe eines geeigneten Twitter-Suchbegriffs die fortschreitende Interessenszunahme an dem SaaS-Angebot identifiziert. Beispielsweise spiegeln Suchbegriffe der Art 'Microsoft Software Update' ein Nutzerinteresse an angekündigten Software-Rollouts von Microsoft wieder. Konkret kann via Twitter-API festgestellt werden, wie viele Tweet-Erwähnungen zu dieser Thematik erscheinen.

Daraufhin wird untersucht, wie umfangreich das Interesse ist. Dabei fließen die Informationen ein, in welchem Zeitraum und an welchem Ort die Tweets veröffentlicht wurden. Die dazu benötigten Verlaufsdaten können direkt aus der Twitter-API abgefragt werden. Die Programmierschnittstelle lässt nämlich die Suche nach beliebigen Begriffen zu und gibt auf Anfrage die zugehörigen Tweets mit ihrem Erscheindatum sowie ihrem Ort zurück. Nach diesbezüglicher Analyse kann der SaaS-Anbieter rechtzeitig abschätzen, wann ein hohes Kapazitätsvolumen für den Download bereitstehen muss.

Die Popularitätsmessung eines Themas und die Bestimmung der Nachrichtenherkunft erfolgen bei der Twitter-API in einem Nachrichten-Stream im JSON-Format. JSON steht für JavaScript Object Notation und ist ein leichtgewichtiges Textdatenformat zum Austausch zwischen Anwendungen [47].

Der Stream-Eingang ist allerdings bezüglich des zulässigen Nachrichtenempfangs pro Stunde durch den Anbieter limitiert. Infolgedessen wird im Rahmen dieser Arbeit ein eigenes implementierter Stream-Server verwendet. Somit ist es möglich, unbegrenzt Nachrichten zu verschicken und durch die Beeinflussung der Nachrichtenmenge auch rekonstruierbare Evaluationsergebnisse zu erhalten. Das Nachrichtenformat enthält dabei einen Zeitstempel und die Koordinate, von wo die Nachricht abgesetzt wurde.

Über die Twitter-API kann ein Anstieg der Anzahl an Tweet-Erwähnungen zu einem das SaaS-Angebot betreffenden Thema identifiziert werden. Sobald eine solche Zunahme verzeichnet wird, soll mittels gespeicherter Historie eine Vorhersage zu dem künftig aufkommenden Lastverhalten ableitbar sein. Wie bereits in Abbildung 7.3 gezeigt, müssen für den dargestellten Algorithmus verschiedene Parameter verwaltet werden.

Zunächst werden in einer Tabelle $TweetHistorie$ die relevanten Suchbegriffe für die API-Abfrage bei Twitter gespeichert. Zudem wird die Häufigkeit der Erwähnungen in Tweets innerhalb eines bestimmten Zeitintervalls I_1 identifiziert. Der Algorithmus entnimmt dann regelmäßig die Suchbegriffe aus der Tabelle und ermittelt im Vergleich zum vorherigen Intervall den Anstieg beziehungsweise Rückgang ihrer Anzahl:

```
array A (T, I_1):
    for each i in T do:
         if (tweet = searched) and (time(i) elem of I_1):
           -> freq += freq(i);
           -> time(i) = time(tweet);
    return T;
```

Durch die Arbeiten von Yang und Leskovec [184] ist bekannt, dass die identifizierten Lastmuster gerade in den ersten Stunden ein steiles Popularitätswachstum im Lastverhalten aufweisen. Da es sich in diesem Fall um Informationsverbreitung in sozialen Netzwerken handelt, kann sogar von einem anfänglich exponentiellen Wachstum ausgegangen werden [183]. Deshalb wird für die Simulation des implementierten Tweet-Servers die folgende Funktion (7.1) mit exponentiellem Wachstum der Tweets definiert:

$$f_{Tweet}(t) = e^{\alpha t} + \beta \tag{7.1}$$

Die eulersche Zahl e repräsentiert die Funktionsbasis. Der Wachstumsfaktor der Funktion wird mit α bezeichnet. β entspricht dem Anfangsniveau, also dem Verschiebungsfaktor auf der y-Achse.
Der Startzeitpunkt $t_1 = 0$ ist mit dem Anfangswert y_1 belegt. Zu einem späteren Zeitpunkt t_2 kann sich ein beliebiger Wert y_2 ergeben. So ist die Gleichung folgendermaßen zu lösen:

$$\begin{aligned} \text{I)}\ & y_1 = e^{\alpha \cdot 0} + \beta = 1 + \beta \Rightarrow \beta = y_1 - 1 \\ \text{II)}\ & y_2 = e^{\alpha t_2} + \beta \\ \text{I in II)}\ & y_2 = e^{\alpha t_2} + y_1 - 1 \Rightarrow \alpha = \frac{ln(y_2 - y_1 + 1)}{t_2} \end{aligned}$$

Somit sind die Werte für α und β ermittelt und können eingesetzt werden:

$$f_{Tweet}(t) = e^{\frac{ln(y_2 - y_1 + 1)}{t_2} \cdot t} + y_1 - 1$$

Folglich kann die Wachstumsfunktion (7.1) mit gegebenen Zeitpunkten sowie den Start- und Endwerten konfiguriert werden. Das Popularitätswachstum lässt sich in verschiedenen Simulationsläufen variieren.
Im zweiten Schritt wird die Anzahl der Tweet-Erwähnungen mit der Lastkurve auf den SaaS-Ressourcen in Beziehung gesetzt. Dafür werden der Lastverlauf im jeweils gleichen Zeitintervall der Tweet-Messung erfasst und die Lastparameter abgespeichert. So sind Tweet- und Lastkurve vergleichbar.
Durch die Bestimmung des Anstiegs oder Rückgangs von Tweet-Erwähnungen sind die Korrelationsunterschiede zu einem zweiten konfigurierbaren Zeitintervall I_2 identifizierbar. Genauer gesagt werden die Steigungen der Tweet- und Lastkurven berechnet und der zeitliche Versatz im Kurvenanstieg zwischen Tweet-Erwähnungen und Arbeitslast ermittelt. Dafür wird zum definierten Zeitintervall I_1 die Häufigkeit der Suchbegriffe aus der Tabelle $TweetHistorie$ entnommen und die Steigung für das Intervall I_2 errechnet. Anschließend wird mit dem Standardverfahren des Chi-Quadrat-Koeffizienten untersucht, ob der Tweet-Verlauf mit der Kurve der Lastentwicklung korreliert.
Mit dem Chi-Quadrat-Homogenitätstest wird per Hypothesenprüfung festgestellt, ob verschiedene Stichproben – in diesem Fall die Verteilungen der Tweet-

und der Lastkurve – derselben Grundgesamtheit angehören [145]. Der entwickelte Algorithmus betrachtet zwei Hypothesen:

H_0: Die Steigungen der Systemlast und der Tweets sind identisch verteilt.

H_1: H_0 trifft nicht zu.

Der Hypothesentest erfordert die Bildung der Chi-Quadrat-Prüfgröße, die sich nach Pearson folgendermaßen berechnet [136]:

$$\chi^2 = \sum_{j=1}^{k} \sum_{i=1}^{m} \frac{(n_{ij} - E_{ij})^2}{E_{ij}}$$

Zu diesem Zweck sind zunächst die Randverteilungen, genauer die aktuelle Zellhäufigkeit n_{ij} und die erwartete Häufigkeit jeder Zelle E_{ij}, zu ermitteln. Dafür sind von allen Zeilen und Spalten die jeweiligen Summen s_i und s_j zu berechnen. Mit diesen Summen ist die zu erwartende Zellhäufigkeit E_{ij} bestimmbar. Beim Einfügen jeder Tabellenzeile werden dafür die Zeilensummen s_i mit berechnet. Für die Spaltensummen s_j der Tweets und der Last wird eine Iteration über alle Tabellenzeilen durchgeführt. Nach der Tabelleniteration wird ein Summenwert s aller Zeilen- und Spaltensummen festgestellt, womit eine erwartete Zellhäufigkeit E_{ij} angeben werden kann:

$$s = \sum_{i=1}^{m} s_i + \sum_{j=1}^{k} s_j$$

$$E_{ij} = \frac{s_i \cdot s_j}{s}$$

So wird mittels der erwarteten Zellhäufigkeit die Chi-Quadrat-Prüfgröße berechnet. Diese Prüfgröße beschreibt in diesem Fall die Wahrscheinlichkeit einer Korrelation zwischen der Tweet- und der Lastkurve. Der konkrete Wahrscheinlichkeitswert ist in einer Tabelle mit den Quantilen der Chi-Quadrat-Verteilung abzulesen [14]. Zur Hypothesenüberprüfung muss die Anzahl der Freiheitsgrade FG bestimmt werden. Das erfolgt über die Anzahl der Spalten c und die Anzahl der Zeilen r auf folgende Weise [145]:

$$FG = (c - 1) \cdot (r - 1)$$

Nun wird die Hypothese H_0 verworfen, wenn die Prüfgröße χ^2 in der Quantil-Tabelle mit den Freiheitsgraden eine geringere Wahrscheinlichkeit aufweist als gefordert [14]. Im anderen Fall wird H_0 akzeptiert. Eine Korrelation zwischen der Popularität der Cloud-Anwendung und einem Lastanstieg auf ihr ist anzunehmen. Für die providerspezifische Festlegung, ab welcher Korrelationswahrscheinlichkeit zusätzliche Ressourcen eines Cloud-Dienstes hochgefahren werden sollen, ist zuerst die Anzahl der Messpunkte festzulegen.

Um für einen Chi-Quadrat-Homogenitätstest ein aussagekräftiges Ergebnis zu erhalten, gelten nach Rinne und Voß ein paar grundliegende Bedingungen der Daten [145, 175]. Die Messreihe muss ausreichend groß sein, damit lokalbedingte Schwankungen im Kurvenverlauf weniger Gewicht haben.

Zur Vergleichbarkeit des errechneten Ergebniswerts mit den Prüfgrößen der Quantil-Tabelle von beispielsweise Andreß [14], werden mindestens 30 Messwerte benötigt [145]. Außerdem müssen die Messpunkte insgesamt positiv sein. Konkret müssen alle erwarteten Zellhäufigkeiten $E_{ij} \geq 1$ sein. Für 80 % der erwartete Zellhäufigkeiten muss $E_{ij} > 5$ gelten und für die Zellhäufigkeit $n_{ij} \geq 10$. Dies ist erforderlich, da sonst Schwankungen kleiner Zahlen deutlich höher ins Gewicht fallen.

Sobald mit diesen Berechnungen festgestellt wird, dass die Hypothese H_0 zutrifft, gilt eine Tweet-Last-Korrelation als erkannt. Basierend auf der Tweet-Kurve wird so die Wahrscheinlichkeit für den tatsächlichen Lastanstieg mithilfe des Chi-Quadrat-Koeffizienten ermittelt. Dabei kann für einen signifikanten Zusammenhang – was dem positiven Chi-Quadrat-Test entspricht – von einer 80-prozentigen Wahrscheinlichkeit ausgegangen. Der Wert wird bis zu einer Wahrscheinlichkeit von 99 % hochskaliert.

Mit der resultierenden Wahrscheinlichkeit kann die dem Zeitintervall zugeordnete Lastkurve angegeben werden. Somit ist das Anfragevolumen zu prognostizieren und die Verzögerung zur tatsächlich auftretenden Last abzuschätzen. Nach einer Korrelationserkennung werden mit dem in Kapitel 5 beschriebenen Protokoll zusätzlich erforderliche Ressourcen vorab reserviert. Im Falle von drohenden Überlastsituationen werden so auch ohne Vorab-Reservierung Ressourcen rechtzeitig vor dem erforderten Zeitpunkt hochgefahren.

Der Algorithmus stößt also bei hohen Wahrscheinlichkeitswerten einer Korrelation selbstständig einen angemessenen Skalierungsprozess an. Um überflüssige Ressourcenallokationen zu verhindern, sollen lediglich die signifikanten Lastanstiege im Chi-Quadrat-Homogenitätstest analysiert werden. Allerdings ergibt der Test ebenfalls eine erhebliche Korrelationswahrscheinlichkeit, falls keine oder nur äußerst geringe Steigungen in den Tweet- und Lastkurven bestehen, solange diese konstant und auf gleiche Weise verlaufen. Für die korrekte automatische Anpassung des Kapazitätsvolumens ist es deshalb unumgänglich, diese flachen und konstanten Kurvenverläufe der Tweet-Erwähnungen und der Arbeitslast vorab mithilfe einer Lastmusteranalyse herauszufiltern.

Für die Erkennung der Lastmuster werden die aufgezeichneten Kurven mit bereits identifizierten Mustern verglichen und ihre Ähnlichkeit überprüft. Abbildung 7.4 veranschaulicht diesen Algorithmus. Wie dargestellt, fließen in die Musteranalyse die Historien des Tweet- und Lastverlaufs ein. Dadurch wird vorab erkannt, ob die Kurven steigen, fallen oder konstant verlaufen. Der Chi-Quadrat-Test berücksichtigt dann nur die Kurven, die in dem überprüften Zeitraum eine signifikante Steigung der Arbeitslast aufweisen. Auf diese Weise wird bei unwesentlichen kleineren Anstiegen im Tweet-Verlauf eine unnötige Zuschaltung von Instanzen vermieden.

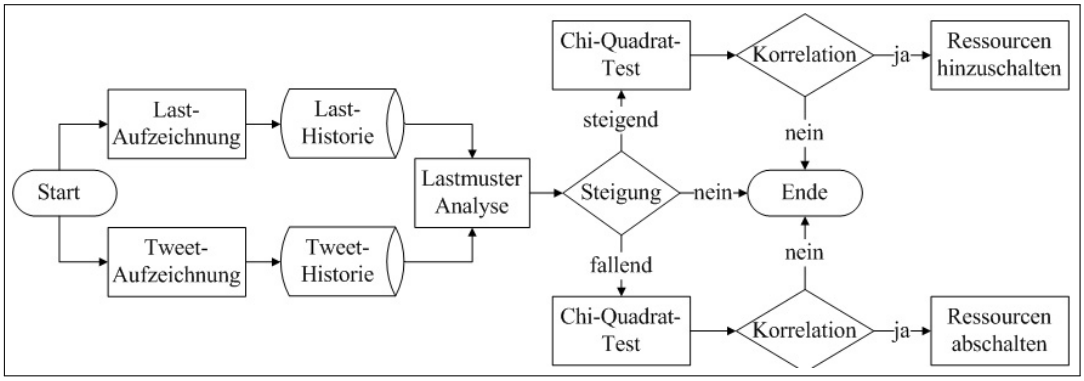


Abbildung 7.4: Algorithmus zur Allokation der Ressourcen

Gemäß dem Algorithmus ist bei einem Rückgang des Nutzerinteresses generell auf eine Lastabnahme zu schließen. Dies entspricht einer „fallenden Steigung“. Ein Rückgang der Tweet-Erwähnungen zieht allerdings nicht immer einen Lastabfall nach sich, da trotz fallender Tweet-Kurve die Anwendung eventuell weiter genutzt wird. Im Allgemeinen ist jedoch laut den von Yang et al. [184] ermittelten Lastmustern durchaus davon auszugehen, dass das Nutzungsinteresse ab einem bestimmten Punkt abnimmt. Bei entsprechender Korrelationserkennung in den fallenden Kurven sind dann Instanzen wieder abzugeben. Dadurch werden unnötige Kosten durch Ressourcen im Leerlauf reduziert.

Nach der Fallunterscheidung per Lastmusteranalyse werden also einerseits bei positivem Abgleich eines Lastanstiegs Ressourcen hinzugefügt. Andererseits werden bei einem Lastrückgang Ressourcen heruntergefahren. Die entwickelten Formeln zur detaillierten Bestimmung der optimalen Menge zu allokierender Instanzen und der entstehenden Kosten, werden in Abschnitt 7.4 erläutert.

7.2.2 Ortsspezifische Bedarfsbestimmung

Neben der *Anzahl* erforderlicher Ressourcen ist zu ermitteln, an welchem *Ort* die Instanzen hauptsächlich benötigt werden. Um eine Prognose drüber abgeben zu können, wo der Hauptbedarf vorliegt, wird zunächst eine Liste von konkreten realen Lokalitäten initialisiert. Beispielsweise sind Lokalitäten wie Bibliotheken oder Universitäten für Software-as-a-Service-Angebote relevant, die ihren Fokus im Bildungs- oder Forschungssektor sehen.

Anschließend werden neben den Twitter-basierten Ortsinformationen auch Daten von Geocaching- oder Geotagging-Diensten genutzt, um die tatsächlichen Aufenthaltsorte angemeldeter mobiler Nutzern zu extrahieren.

Die mit der Abfrage-API ermittelten Daten dienen der Erhebung des Benutzeraufkommens und bestimmen den aggregierten Ressourcenbedarf mit dem jeweiligen Interessenschwerpunkt. Bei der Erkennung einer hohen Anzahl von Personen an einem großen Veranstaltungsort wie einer Konzerthalle

ist zum Beispiel die Wahrscheinlichkeit für die Nutzung eines SaaS-basierten Musikaufnahme- oder Foto-Sharing-Dienstes erhöht.

Mithilfe der Ortsinformationen ist eine optimierte Zuteilung des anfragenden Nutzers an die bestgeeignete Hosting-Lokalität des Cloud-Anbieters erreichbar. Wie erwähnt, bieten die PaaS- und IaaS-Betreiber ihren Kunden zum Hosten der SaaS-Anwendungen weltweit verschiedene Rechenzentren an. Microsoft Windows Azure unterteilt die Welt grob in die Regionen West und Nord Europa, Süd-Ost und Ost Asien sowie Nord, Ost, West und Süd USA [119]. Da andere Anbieter eine ähnliche Aufteilung vornehmen, bilden im Folgenden genau diese Regionen die Basis für den Algorithmus.

Diesen Großregionen werden die einzelnen Tweet-Erwähnungen, die im vorherigen Schritt des entwickelten Algorithmus aufgezeichnet wurden, zugeordnet. Dafür wird die reale Welt so mit Polygonen überspannt, dass ein Viereck entsteht, welches an den Polarkreisen endet. Diese reduzierte Größe ist ausreichend, da an den Polarkreisen kein Internet-Empfang möglich ist. Von dort wird kaum eine Nutzung des SaaS-Angebots erfolgt. Somit wird die zu durchsuchende Polygonmenge angemessen begrenzt.

Wie erwähnt, setzt der implementierte Test-Server JSON-Nachrichten in einem Stream ab. Der in Kapitel 5 erläuterte Service-Load-Manager hört ihn für eine frei definierbare Zeitdauer ab. Damit wird geprüft, aus welchem der festgelegten Polygone die einzelnen Tweets kommen. Zu diesem Zweck fragt der Manager die konkreten Koordinaten der Ortsinformationen per Twitter-API ab. Mithilfe der ermittelten geographischen Längen- und Breitengrade sind die ankommenden Tweet-Nachrichten einer spezifischen Region zuzuordnen. Dies wird in einem Schema mit dem Datum, den Längen- und Breitengraden der Koordinaten sowie den Referenzorten abgespeichert.

Für die betrachtete Periodendauer muss berechnet werden, wie viele Tweets pro Region in diesem Zeitintervall eingehen. Dafür werden die einzelnen Tweet-Lokalitäten mit ihren Koordinaten gruppiert und dann je Region die Tweet-Erwähnungen gezählt. Eine Tabelle speichert die Ergebnisse dieser Aggregation. Sie umfasst konkret neben dem Ort, dem Intervall und der Anzahl der im Intervall aufgezeichneten Tweets auch den Lastumfang der betroffenen SaaS-Anwendung. Zudem werden die für den Chi-Quadrat-Test benötigten Zeilensummen der Tweet- und Lastwerte abgelegt.

Durch die Gruppierung wird einerseits die Wahrscheinlichkeitsbestimmung eines spezifischen Lastvolumens abhängig vom Dienst erleichtert. Andererseits kann bei der Allokation zusätzlicher Instanzen ein dem Nutzer nahe gelegener Ort gewählt werden. Das verhindert Verzögerungen, die durch das Weiterleiten von Anfragen über weite Entfernungen entstehen.

Dieses entwickelte Verfahren ist offensichtlich außerordentlich komplex. Um den Algorithmus leichter verständlich zu machen, werden im Folgenden konkrete Beispiele angeführt. Sie zeigen, welche Parameter vom protokollspezifischen Service-Load-Manager verwaltet werden. Die entworfene Prozedur ist knapp in [154] beschrieben.

- Zunächst werden die Hosting-Lokalitäten des Cloud-Anbieters, der das SaaS-Angebot bereitstellt, in einer Menge H gespeichert. Dies kann für die Region West USA mit dem Beispiel Kalifornien bedeuten:
  $H_{\text{Kalifornien}} = \{\text{San Francisco, Portland, Oakland, etc}\}$.
- Für jeden Eintrag der Hosting-Lokalitäten aus H wird eine Tabelle R_H geführt, die als Elemente die Referenzorte der Hosting-Lokalitäten beinhaltet, zum Beispiel für Kalifornien: $R_{\text{San Francisco}} = \{\text{Powell St., Embarcadero, Mosc. Center, etc}\}$.
- Zudem wird je eine Tabelle A_H pro Hosting-Lokalität H generiert. Sie speichert in den Zeilen die Nutzeranzahl für einen Referenzort pro Zeitintervall. Die Tabelle ist als Ringpuffer mit fester Spaltenanzahl realisiert, deren Zeilen die Referenzorte enthalten. Sie werden zu bestimmten Uhrzeiten fortlaufend gefüllt. Tabelle 7.1 zeigt eine solche Implementierung mit halb- beziehungsweise stündlicher Nachverfolgung.

Kalifornien	...	07:00	07:30	08:00	08:30	09:30	10:00	...
Powell St.	...	43	235	121	43	4	445	...
Embarcadero	...	45	435	143	54	6	344	...
Mosc. Centre	...	56	562	186	43	7	742	...
Union Square	...	33	333	134	35	3	532	...

Tabelle 7.1: Beispielwerte für Tabelle $A_{\text{Kalifornien}}$

- In der Lokalitätenmenge L werden reale, für den SaaS-Anbieter relevante Veranstaltungsorte mit der zugehörigen Nutzermenge gespeichert. Damit können zu den Hosting-Orten Referenzpunkte geschaffen werden, z.B. für $L_{\text{Kalifornien}} = \{\text{Santa Cruz, Los Angeles, San Diego, etc}\}$ die Beispielmenge $R_{\text{San Francisco}}$. Da so auch die Entfernung zu echten Veranstaltungsorten bestimmbar ist, können die Anfragen auf ein nahe gelegenes Rechenzentrum des Betreibers abgebildet werden.
- Eine Tabelle B speichert die gestellten SaaS-Angebote des zugrundeliegenden Cloud-Betreibers. Mittels Join-Anweisung der Tabellen $A_H \bowtie B$ ist es möglich, eine Aussage zu treffen, ob eine bestimmte SaaS-Anwendung für den geführten Referenzort von Bedeutung ist. Durch statistische Extrapolation kann dann auch ihre Relevanz für den tatsächlich betrachteten Ort festgelegt werden.
  Beispielsweise ist es für SaaS-Anwendungen des Bildungssektors sinnvoll, als Referenzort Bibliotheken zu wählen, um auf die tatsächliche Nutzung in einer betrachteten Universität zu schließen. Hingegen steigt mit Erkennung einer Nutzerballung bei einem großen Konzert eher die Wahrscheinlichkeit für die Nutzung einer Foto-Sharing-Plattform.

Abbildung 7.5 visualisiert den Systemablauf. Darin sind verschiedene Rechenzentren [A] eines Anbieters platziert. Sie werden mit der Liste der Referenzorte [B], die eine signifikante Relevanz für das SaaS-Angebot haben, parametrisiert (1). Diese Rechenzentren hosten die betrachtete Anwendung [C].

Gemäß dem beschriebenen Algorithmus beginnt der Prozess mit der Parameterbelegung der Menge H der Hosting-Lokalitäten eines Anbieters. Ebenso findet eine Initialisierung mit den verschiedenen Referenzorten R_H der Hosting-Lokalitäten statt. Die Liste der Referenzorte beinhaltet in diesem Beispiel Universitäten, Bibliotheken, Fußballstadien und Konzerthallen. Sie werden in den Zeilen der Tabelle A_H gespeichert.

Schritt (2) führt den Abgleich der Applikation und des Hosting-Orts durch. Auf Basis der Referenzorte wird die nächstgelegene Hosting-Lokalität ausgewählt. So können beispielsweise Referenzorte in Europa entweder auf ein Rechenzentrum in Holland oder auf eines in Irland abgebildet werden. Dies ist abhängig davon, welche der Hosting-Lokalitäten den europäischen Ort besser bedient.

In Schritt (3) werden über die geeigneten APIs von Twitter und Geocaching- oder Geotagging-Diensten [D], wie beispielsweise Foursquare [61], die Aufenthaltsorte [E] einzelner mobiler SaaS-Anwendungsnutzer in periodischen Zeitintervallen abgefragt. Die Länge des Intervalls kann frei gewählt werden, sollte aber in etwa eine Rate von 30 Minuten haben. Bei längeren Abständen gehen Werte verloren und bei kürzeren lohnt sich der Verwaltungsaufwand nicht. In diesem Intervall werden die Nutzerzahlen pro Hosting-Lokalität aggregiert und zugehörige Trends ermittelt. Auf diese Weise kann die Erhebung von Nutzerballungen an einem speziellen Ort erfolgen. Das ermöglicht die Bestimmung der zu bevorzugenden Hosting-Lokalität in Schritt (4).

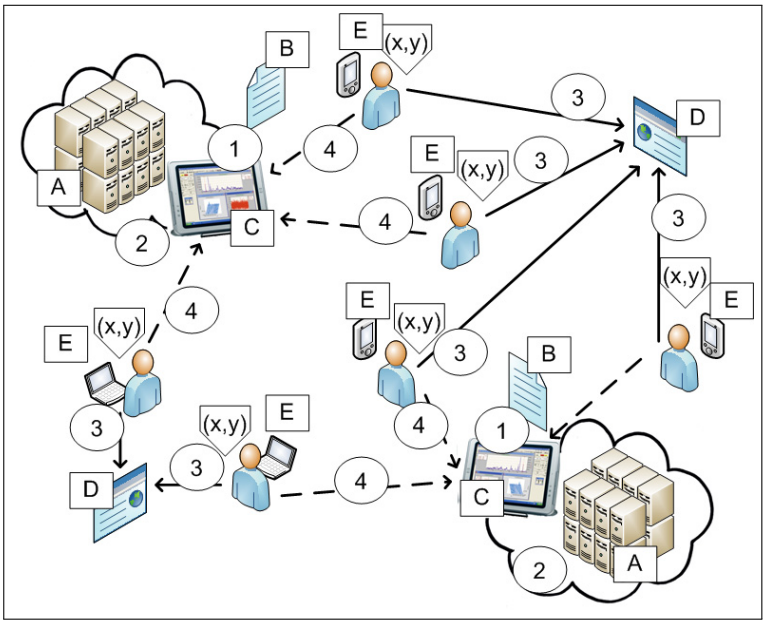


Abbildung 7.5: Ablauf ortsspezifischer Bedarfsbestimmung [154]

Bei der konkreten Festlegung einer Hosting-Lokalität drängte sich zunächst die Überlegung auf, einfach ein Paket des entwickelten Ressourcenmanagement-Protokolls an die IP-Adresse oder den Hostnamen des jeweiligen Services zu senden. Allerdings werden IP-Adressen standardmäßig per DHCP, also mithilfe des Dynamic Host Configuration Protocols, dynamisch vergeben. Infolgedessen müssten Hostnames beim Service-Load-Manager explizit in einer Tabelle verwaltet werden. Das bedeutet einen erheblichen Aufwand, da neue Einträge manuell hinzuzufügen und alte entsprechend wieder zu löschen sind.

Um diese Zusatzbelastung zu umgehen, fiel die Entscheidung darauf, einen Broadcast an alle verbunden Services zu schicken. Ein Dienst muss dann unterscheiden können, ob ein empfangenes Paket für ihn oder einen anderen bestimmt ist. Dafür wird das Protokoll um ein zusätzliches Feld erweitert. Es enthält die Region, wo die Ressourcen bereitzustellen sind.

Sobald ein Service ein Broadcast-Paket empfängt, vergleicht er diesen Feldeintrag mit seinem eigenen Ort. Bei Übereinstimmung werden gemäß der Anweisung die benötigten Ressourcen gestartet. Ein solches Paket wird genau dann versendet, wenn für eine Region bezüglich des in Abschnitt 7.2.1 beschriebenen Algorithmus eine Korrelation zwischen Tweet und Last ermittelt ist. So wird sichergestellt, dass die erforderlichen zusätzlichen Ressourcen in der nahen Umgebung des Clients zur Verfügung stehen.

7.3 Evaluation des Algorithmus zur ortsbasierten Ressourcenallokation

Für die Bewertung des Algorithmus ortsbezogener Ressourcenallokation basierend auf sozialen Netzwerkdaten werden verschiedene Aspekte evaluiert. Zunächst wird die Eignung des Chi-Quadrat-Tests zur Korrelationsbestimmung zwischen den Arbeitslast- und Tweet-Kurven analysiert. Zur Identifikation der Stärken und Schwächen des Tests werden Messfehler quantifiziert. Diese können beim Empfang der sozialen Web-Nachrichten auftreten.

Daraufhin werden in Übereinstimmung mit Leskovec et al. [184] beispielhafte exponentielle Tweet-Kurven erstellt. Parallel erfolgt die Simulation mehrerer Lastkurven mit variierendem, ebenfalls exponentiellem Wachstum. Die erzeugten Kurven der Tweets und Lastverläufe werden jeweils per Chi-Quadrat-Test auf Korrelation überprüft. Wie Abschnitt 7.3.1 darlegt, kann damit eine ungefähre Abschätzung zur Sensitivität des Verfahrens erfolgen.

Der zweite Teil der Evaluation widmet sich der Frage, ob durch den entworfenen Algorithmus eine kürzere Antwortzeit der genutzten Dienste realisierbar wird. Dabei gilt es vor allem zu bestimmen, inwiefern die Verarbeitungszeiten der Server und die Entfernungen zwischen Nutzer und Dienst auf die Antwortzeiten einwirken. Dafür wird in Abschnitt 7.3.2 mithilfe einiger Simulationen

die Dauer zwischen Absetzen einer Anfrage von einem Server der festgelegten Regionen und dem Empfang der Dienstantwort ermittelt.
Abschließend werden in Abschnitt 7.3.3 die gewonnenen Simulationsresultate analysiert und die Ergebnisse bewertet. Dies dient der aussagekräftigen Untersuchung, ob eine ortsspezifische Ressourcenallokation tatsächlich die Dienstantwortzeit senken kann.

7.3.1 Fehler-Quantifizierung der Chi-Quadrat-Methode

Zur Evaluation des Chi-Quadrat-Verfahrens werden zunächst die Daten ausgewertet, die fortlaufend beim im Rahmen dieser Arbeit entwickelten Tweet-Server eingehen. Die Messung erstreckt sich über fünf Stunden. Die Implementierung des Tweet-Servers lässt die gezielte Steuerung des Tweet-Wachstums in den eingangs definierten Regionen zu. Der Nachrichtenversand erfolgt sequentiell jeweils für eine Region. So sind pro Sequenz gezielt nur die Nachrichten aus einer einzelnen Region zu verzeichnen.
Die Sequenzlänge wächst mit der Zeit stetig an. In Abbildung 7.6 ist die jeweilige Sendedauer, sprich eine Sequenz je betrachteter Region, durch farbige Blöcke visualisiert. Die vertikalen Linien trennen die einzelnen Slots und entsprechen den Messintervallen.
Der Tweet-Client sammelt je spezifischem Intervall I die Tweets und gruppiert sie nach Regionen. Dadurch kann die Situation entstehen, dass eine Sequenz die Intervalllänge überschreitet und ein Block dem nächsten Intervall zugeordnet wird – wie der mittlere grün dargestellte Block in Abbildung 7.6.
Mit fortschreitender Simulationsdauer unterliegt die Messprozedur damit einer immer größeren Streuung. Dieser Sachverhalt ist auch in Abbildung 7.7 erkennbar. Sie zeigt den Tweet-Umfang der Messung und verdeutlicht graphisch die Streuung im Zeitverlauf. Je weiter rechts die Messpunkte auf der x-Achse liegen, desto mehr oszillieren und schwanken die Ergebniswerte.
Da der Fehler jedoch quantifizierbar ist, kann man ihn sich zunutze machen. Durch die Umwandlung dieser Eigenschaften in eine messbare Größe werden die folgenden Konsequenzen für die Parametrisierung gewonnen.
Zur Quantifizierung ist die Streugröße über eine Funktion F definiert. Dabei wird die bereits erläuterte Wachstumsfunktion (7.1) für die Tweet-Eingänge

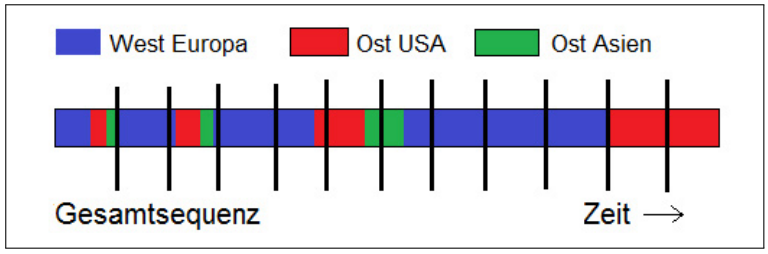


Abbildung 7.6: Sequenzierung durch den Tweet-Server

auf dem Server verwendet: $f_{Tweet}(t) = e^{\alpha t} + \beta$. In der folgenden Formel stellt i_a den Anfang des beobachteten Intervalls I und i_e das Ende dar. Der maximale Fehler F_{max} lässt sich festlegen als:

$$\begin{aligned} F_{max} &= \pm \int_{i_a}^{i_e} f_{Tweet}(t)\ dt = \pm \int_{i_a}^{i_e} (e^{\alpha t} + \beta)\ dt \\ &= \pm(\frac{1}{\alpha}(e^{\alpha i_e} - e^{\alpha i_a}) + \beta(i_e - i_a)) \end{aligned} \tag{7.2}$$

Durch Formel (7.2) wird deutlich, wie sich der maximale Fehler F_{max} reduzieren lässt. Einen großen Einfluss hat die frei wählbare Intervalldauer bei der Chi-Quadrat-Bestimmung. Um Abweichungen zu minimieren, sind die Intervallabschnitte des Tests folglich möglichst gering zuhalten. Zudem ist der Streufehler durch eine größere Anzahl an Messpunkten verringerbar. Mit einer geeigneten Belegung dieser beiden Parameter können Ausreißer den Verlauf weniger stark manipulieren.

Eine genaue Bestimmung von F_{max} erfordert gesammelte Daten aus bereits bestehenden Historien. Dieser Evaluation sind die in Abschnitt 7.1 ausgeführten Forschungsergebnisse zugrunde gelegt. Die hieraus gewonnenen Erkenntnisse zum Popularitätsverlauf zeigen, dass sich der exponentielle Kurvenanstieg über das erhebliche Zeitintervall von über 20 Stunden erstreckt. Somit wird deutlich, dass die Messungen zur Erkennung eindeutiger Muster mehrere Stunden der Vergangenheit mit berücksichtigen müssen.

Die Tweet-Kurve mit den Werten der fünfstündigen Messung in Abbildung 7.7 zeigt, dass vor allem zu Kurvenbeginn der Streuungsverlauf kaum zu erkennen ist. Allerdings nimmt die Streuung durch das exponentielle Verhalten der Popularitätskurve mit der Zeit massiv zu. In der Abbildung ist das anhand der enormen Ausschläge weiter rechts auf der x-Achse sichtbar.

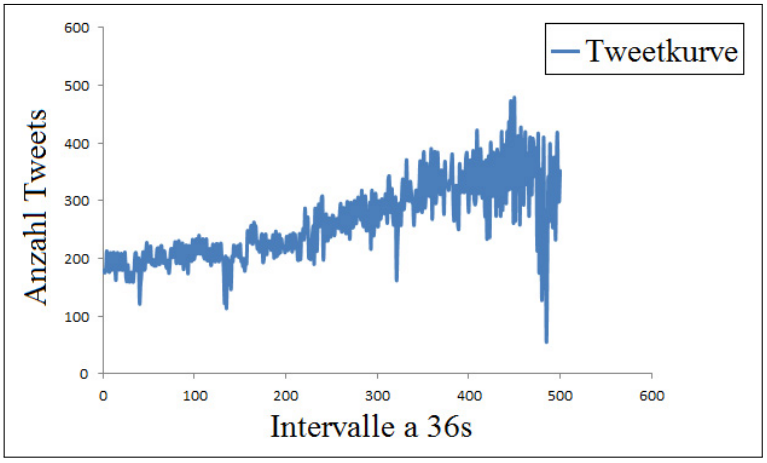


Abbildung 7.7: Streuung der Tweets nach fünf Stunden

Durch die Beeinflussung der Parameter von Intervallgröße und Häufigkeit der Messpunkte lässt sich jedoch die Streuung minimieren, was entscheidend für die Güte des Chi-Quadrat-Homogenitätstests ist.

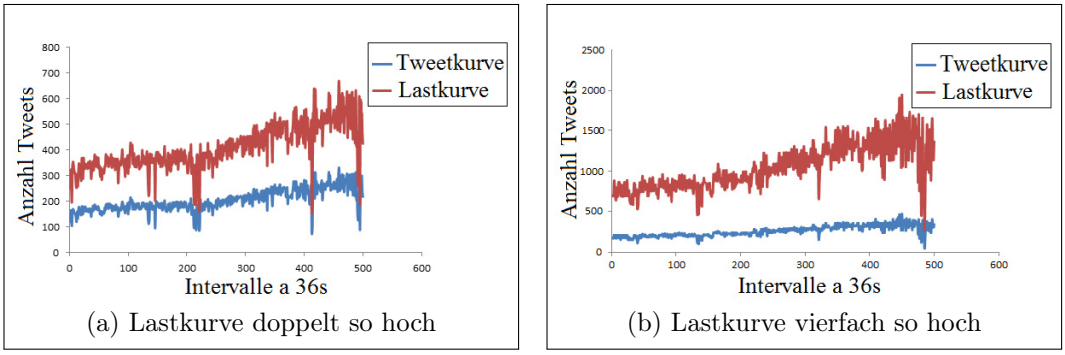
Um das Verständnis für die Sensitivität der Chi-Quadrat-Methode im vorgestellten Algorithmus zu fördern, werden nun einige Beispiele erläutert. Sie behandeln die Korrelationsbestimmung zwischen den erstellten Exponentialkurven der Tweets und der Arbeitslasten mit variierendem Wachstumsparameter. Abbildung 7.8 stellt die Ergebnisse der Kurvensimulationen gegenüber.

Es werden zwei Simulationsläufe durchgeführt. Darin wird ermittelt, ob die Lastkurve ein Vielfaches zur Kurve der Tweet-Erwähnungen ist – sprich wie sich die Wahrscheinlichkeit der Korrelation verhält. Dafür folgt die Lastkurve der Form $\gamma \cdot e^{\alpha t} + \beta$. Das erwähnte Vielfache ist durch den Faktor γ ausgedrückt. Er wird in den Tests einmal auf $\gamma = 2$ und einmal auf $\gamma = 4$ gesetzt. Folglich werden eine doppelt so hohe Last auf den Tweet-Servern im Vergleich zur Tweet-Anzahl und eine vierfache Lasterscheinung simuliert.

In Abbildung 7.8a ist ein χ^2-Wert von 40,47 zu erkennen. Laut Andreß entspricht dies bei 500 Freiheitsgraden einer Übereinstimmung mit über 99,99 % [14]. Folglich korrelieren Tweet- und Lastkurve bei doppelter Arbeitslast und gleichem Wachstumsfaktor mit einer enorm hohen Wahrscheinlichkeit.

Abbildung 7.8b zeigt die resultierende Tweet- und Lastkurve bei einem Vielfachfaktor von $\gamma = 4$, wobei sich ein χ^2-Wert von 126,22 ablesen lässt. Dies entspricht ebenfalls einer Korrelationswahrscheinlichkeit von 99,99 % [14]. Aus den Ergebnissen dieser beiden Simulationen kann man schließen, dass bei einer identischen Wachstumsrate α sogar bei unterschiedlichen Vielfachen γ der Arbeitslast im Vergleich zur Tweet-Anzahl Korrelationen der beiden Kurven identifiziert werden können.

Vorteilhaft ist, dass der Anbieter eines SaaS-Dienstes selbst bestimmen kann, ab welcher Korrelationswahrscheinlichkeit Ressourcen hinzugeschaltet werden. Dabei gilt es, einen Mittelweg zu finden zwischen hohen Kosten bei großzügiger Allokation und andererseits geringer Nutzerzufriedenheit bei zurückhaltenden Strategien. Ist die Wahrscheinlichkeitsgrenze also zu niedrig gewählt, besteht die Gefahr, dass Ressourcen zusätzlich bereitstehen, die ungenutzt bleiben und



(a) Lastkurve doppelt so hoch

(b) Lastkurve vierfach so hoch

Abbildung 7.8: Lastkurve doppelt bzw. vierfach so hoch wie Tweet-Anzahl

unnötige Kosten erzeugen. Wenn andererseits lediglich bei sehr hohem Wahrscheinlichkeitswert eine Instanzenzuschaltung durchgeführt wird, kann vorhandener Bedarf nicht mit ausreichender Kapazität bedient werden. Dies schlägt sich im Performanzverlust bei der Dienstnutzung nieder.

Die Ergebnisse dieses Abschnitts zeigen, dass Korrelationen zwischen der Popularitätskurve von Tweets in sozialen Netzen und dem Lastverlauf auf den bedienenden Servern des SaaS-Angebots erkennbar sind. Mithilfe des Chi-Quadrat-Tests ist die tatsächliche Wahrscheinlichkeit dafür zu bestimmen. Dies gilt, solange von einem gleichen Wachstum der beiden Kurven ausgegangen werden kann, was sich mit den Arbeiten von Yang und Leskovec belegen lässt [183, 184]. Ist die Anzahl der Messpunkte ausreichend umfangreich, hat die Chi-Quadrat-Analyse eine hohe Aussagekraft. Schwankungen zwischen den benachbarten Regionen der unterteilten Zeitintervalle sind beherrschbar.

7.3.2 Simulation zur Bestimmung der Antwortzeiten

Nach der regionsspezifischen Korrelationserkennung zwischen Anwendungspopularität und Systemlast, werden die ortsbezüglichen Antwortzeiten bestimmt. Die folgenden Simulationen analysieren die Round Trip Delays (RTDs) zu verschiedenen Rechenzentren. Die RTDs entsprechen der Zeit vom Absetzen eines Requests an einen Server bis zum Empfang der Antwort beim anfragenden Client. Eine knappe Simulationsbeschreibung findet sich in [159].

Die Proxy- und DNS-Server laufen in verschiedenen Rechenzentren unterschiedlicher Regionen. Der konkrete Server-Ort wird mit dem Dienst *Geobytes* ausgemacht [65]. Er ermittelt den Sendeort einer IP-Adresse. Die Messung erfolgt über das Internet Control Message Protocol (ICMP). Mittels ICMP werden in Rechnernetzen Informations- und Fehlermeldungen unter Verwendung des Internet-Protokolls in Version 4 (IPv4) ausgetauscht [170].

Die ICMP-Messung zielt darauf ab, das Verhältnis der einzelnen RTDs in einem konkreten Wert zu bestimmen. In der Simulation werden je 250 ICMP-Pakete an die vier festgelegten Regionen West Europa, Nord Europa, Ost USA und Ost Asien mittels *ping*-Aufruf aus München verschickt.

Region	mittlere RTD [ms]
Nord Europa	44
West Europa	47
Ost USA	141
Ost Asien	362

Tabelle 7.2: Round Trip Delays zu den Standorten der Rechenzentren

Die Regionen Süd-Ost Asien, Nord USA, West USA und Süd USA werden dabei nicht näher beleuchtet. Wie Tabelle 7.2 zeigt, liefert die Untersuchung der vier betrachteten Regionen aussagekräftige Ergebnisse. In dieser Tabelle sind die gemittelten RTD-Messwerte für die verschiedenen Regionen verzeichnet.

Um Vergleichswerte zu generieren, werden nun die einzelnen Zeiten der Round Trip Delays ins Verhältnis zur hiesigen Region West Europa gesetzt. Dafür wird der folgende Szenarienablauf analysiert. Zunächst ist ein Dienst in den Regionen Nord Europa, Ost USA oder Ost Asien verfügbar. Dann wird ein Lastanstieg in der Region West Europa suggeriert. Nach der Ressourcenallokation in dieser Region ist der Dienst auch direkt dort verfügbar. Die so ermittelten Verhältniswerte der Round Trip Delays zur Region West Europa sind in Tabelle 7.3 aufgelistet. Sie stellen in den weiteren Simulationen die fundierte Grundlage für die Ermittlung der Antwortzeiten.

Prinzipiell wird die Antwortzeit T einer Cloud-Applikation maßgeblich von zwei Zeitparametern beeinflusst. Zum einen fällt die Dauer der Datenübertragung T_D zwischen anfragenden Clients und antwortenden Servern ins Gewicht. Andererseits muss die Verarbeitungszeit T_S der beanspruchten Ressource berücksichtigt werden. Die Gesamtantwortzeit ist als Summe der Datenübertragungsdauer und der Verarbeitungszeit zu verstehen: $T = T_D + T_S$.

Die Simulationen ermöglichen einen Vergleich der Antwortzeiten einzelner Regionen. Die Größenordnungen der beiden beeinflussenden Zeiten werden so variiert, dass die drei Kategorien *Klein*, $\mathit{Mittel} = 10 \cdot \mathit{Klein}$ und $\mathit{Groß} = 10 \cdot \mathit{Mittel} = 100 \cdot \mathit{Klein}$ festzulegen sind. Die Übertragungs- und Verarbeitungszeiten T_D und T_S verlängern sich demnach pro höherer Kategorie um ein Zehnfaches im Vergleich zur darunterliegenden Klasse.

In der Simulation werden weiterhin die Übertragungsverhältnisse der vier betrachteten Regionen aus Tabelle 7.3 eingesetzt. Als Ausgangspunkt zur Simulation der absoluten Transferdauer wird für eine Übertragung der Kategorie *Klein* zu der Region Ost Asien eine Zeit von $T_{D\,\text{Klein}} = 10$ s $= 10.000$ ms festgelegt. Alle weiteren Übertragungszeiten der anderen Regionen und Kategorien werden von dieser hochgerechnet. Zur Vergleichbarkeit sind die Verarbeitungszeiten T_S der Server für alle Regionen gleich konstant festgesetzt. Die einzelnen Verarbeitungen reichen von $T_{S\,\text{Klein}} = 100$ ms über $T_{S\,\text{Mittel}} = 10 \cdot T_{S\,\text{Klein}} = 1.000$ ms bis zu $T_{S\,\text{Groß}} = 10 \cdot T_{S\,\text{Mittel}} = 10.000$ ms.

Region	Verhältnis
Nord Europa	0,94
Ost USA	3,2
Ost Asien	8,23

Tabelle 7.3: Round Trip Delay Verhältnisse zur Region West Europa

In der konkreten Simulationsumsetzung werden n Threads über den Service-Load-Manager ausgeführt. Eine sleep-Anweisung realisiert die Transferdauer T_D zwischen Service und Client. Danach legt der wiedererwachte Thread eine Nachricht in der Warteschlange ab, die den Übertragungsabschluss signalisiert. Dies stößt den Service-Thread an, die Verarbeitungszeit T_S wiederum durch eine sleep-Anweisung zu simulieren. Abschließend übermittelt der Service an den Client, dass die Transaktion abgeschlossen ist. Diese Gesamtdauer wird als Antwortzeit gespeichert.
Die Simulation führt je 20 Threads für die Services und Clients sowie 50 Transaktionen pro Client durch. Sämtliche Anfragen kommen aus der Region West Europa. Dabei werden alle Kombinationen von Kategorien der Datenübertragungs- und Verarbeitungszeit implementiert.

7.3.3 Ergebnisse und Bewertung der Simulationen

Nachfolgend werden die aufgezeichneten Antwortzeiten zu den einzelnen Regionen erörtert und ihre Mittelwerte in Tabellen gelistet. Tabelle 7.4a beinhaltet die Messungen zur hiesigen Region West Europa. Es wird davon ausgegangen, dass dort das SaaS-Angebot vom Nutzer angefordert wird. Deshalb dienen sie als Bewertungsgrundlage. Die mittleren Antwortzeiten liegen zwischen knapp eineinhalb Sekunden bei der Kategorie *Klein* für die Verarbeitungs- und Übertragungszeit und fast zweieinhalb Minuten in der Kategorie *Groß*.
In den weiteren Simulationen werden Szenarien betrachtet, in denen die Ressourcen nicht in der Region West Europa laufen, sondern in entfernten Rechenzentren. Nach der Korrelationserkennung zwischen der Last auf dem Cloud-Dienst und seiner Popularität im sozialen Netz wird also in diesem Fall nicht ortsbasiert Zusatzkapazität angefragt. Damit kann ein Vergleich angestellt werden, wie sich die Antwortzeiten nach der Ressourcenallokation in den unterschiedlichen Regionen Nord Europa, Ost USA und Ost Asien verhalten.

Übertragungszeit T_D [ms]	Verarbeitungszeit T_S [ms]		
	Klein	**Mittel**	**Groß**
Klein	1382	2282	11285
Mittel	12917	13820	22827
Groß	128259	129186	138210

(a) Antwortzeiten Region West Europa

	Verarbeitungszeit T_S [ms]		
	Klein	**Mittel**	**Groß**
Klein	1319	2218	11219
Mittel	12285	13188	22176
Groß	121865	122757	131795

(b) Antwortzeiten Region Nord Europa

Tabelle 7.4: Antwortzeiten der Regionen West und Nord Europa

Die gemittelten Werte der Region Nord Europa belegen in Tabelle 7.4b, dass die Antwortzeiten mit knapp einer Sekunde bis gut zwei Minuten dort sogar minimal geringer sind als die Antwortzeiten innerhalb der Region West Europa. Trotz der größeren Entfernung zur Region Nord Europa kommt es zu kürzeren Round Trip Delays im Vergleich zu näher gelegenen Servern in West Europa. Dies lässt sich durch einen besseren Ausbau der nordeuropäischen Infrastruktur erklären. Aufgrund dieser Beobachtung wird auf die weitere Zeitbewertung für die Region Nord Europa verzichtet.

Das zeigt, dass geographische Nähe nicht automatisch zu besseren Antwortzeiten führt. Folglich muss bei der Ressourcenzuschaltung beachtet werden, dass für diese beiden Regionen in Europa die Instanzenbereitstellung im Rechenzentrum der gleichen Region nicht zwingend ratsam ist.

Dagegen versprechen die hohen Antwortzeiten für die Regionen Ost USA und Ost Asien eine wesentliche Verbesserung der Service-Qualität bei einer Verschiebung der Ressourcenallokation in der Region West Europa.

Tabelle 7.5a umfasst die Antwortzeiten aus der Region Ost USA. Bereits für die kleinste Rubrik der Verarbeitungs- und Übertragungszeit erhält man zu dieser Region eine Antwortzeit von fast vier Sekunden. In der Kategorie *Groß* für Transfer und Bearbeitung wird ein Wert von über sechseinhalb Minuten ermittelt. Die erfassten Antwortzeiten aus der Region Ost Asien in Tabelle 7.5b zeigen sogar noch höhere Werte. Sie reichen in derselben Rubrikaufteilung von zirka zehn Sekunden bis fast siebzehn Minuten.

Nun soll der Zeitgewinn bei einer ortsbezogenen Bereitstellung zusätzlicher Instanzen in Überlastsituation bestimmbar sein. Zu diesem Zweck werden diese ermittelten Antwortzeiten aus Tabelle 7.5 mit den Aufzeichnungen in der Region West Europa aus Tabelle 7.4a ins Verhältnis gesetzt.

Die gemessenen Gesamtzeiten zur Region Ost USA beziehungsweise zu Ost Asien werden durch die einzelnen Ergebniswerte der Simulation innerhalb der Region West Europa dividiert. Das erleichtert den Vergleich der Antwortzeiten abhängig von den jeweiligen Kategorien.

Übertragungszeit T_D [ms]	Verarbeitungszeit T_S [ms] **Klein**	**Mittel**	**Groß**
Klein	3947	4844	13847
Mittel	38575	39465	48465
Groß	384639	385474	394548

(a) Antwortzeiten der Region Ost USA

	Verarbeitungszeit T_S [ms] **Klein**	**Mittel**	**Groß**
Klein	10103	10999	19998
Mittel	100098	100933	110104
Groß	1000285	1000979	1010852

(b) Antwortzeiten der Region Ost Asien

Tabelle 7.5: Antwortzeiten der Regionen Ost USA und Ost Asien

Die so definierten Verbesserungsfaktoren der vier Regionen sind in Tabelle 7.6 gegenübergestellt. Tabelle 7.6a enthält die kategorieabhängigen Zeiteinsparungen, die sich bei der Allokationsverschiebung der Ressourcen von der Region Ost USA nach West Europa ergeben. In der Übertragungskategorie *Groß* und der Verarbeitungskategorie *Klein* erscheint der größte Zeitgewinn in einem Faktor von 3,0. Dieser Wert fällt diagonal ab bis zu einem Faktor von 1,23 für die Übertragungskategorie *Klein* und der Verarbeitungskategorie *Groß*.
Werden die Ressourcen in West Europa anstatt in der Region Ost Asien bereitgestellt, ergeben sich die Verbesserungsfaktoren aus Tabelle 7.6b. Die Tendenz fällt vom größten Verbesserungsfaktor mit 7,8 bei der Übertragungskategorie *Groß* und der Verarbeitungskategorie *Klein* diagonal ab bis zum Minimalwert der Verbesserung von 1,77 bei der Übertragungskategorie *Klein* und der Verarbeitungskategorie *Groß*.
Dabei offenbaren sich deutlich größere Schwankungen innerhalb der speziellen Kategoriegruppen von Ost Asien im Vergleich zu den Verbesserungsfaktoren bei der Allokationsverschiebung nach Ost USA. So ergibt sich für die Kombination von Verarbeitungs- und Übertragungskategorie im jeweils mittleren Bereich ein Wert von 2,86. Dieser steigt wieder auf 4,82 an, wenn die Übertragungszeit auf *Klein* sinkt. Dieser Umstand ist mit den unterschiedlichen Tageszeiten der durchgeführten Messung erklärbar.
Die Verbesserungsfaktoren der Antwortzeiten nach Verlegung der Ressourcenallokation aus der Region Ost USA und der Region Ost Asien nach West Europa in Tabelle 7.6 verdeutlichen wichtige Ergebnisse. Je kleiner die Verarbeitungszeit der Ressource, desto kürzer wird die Antwortzeit nach der ortsbasierten Zuschaltung in der nahen Region. Diese logische Überlegung wird hier mit konkreten Zahlen belegt.
Die Abbildungen 7.9a und 7.9b stellen die Einflusstendenzen der Verarbeitungs- und Übertragungszeit dar. Sie zeigen die Zeitgewinne bei ortsbezogener Allokation in West Europa im Vergleich zur entfernten Bereitstellung in den Regionen Ost USA und Ost Asien. Damit wird folgender Fakt

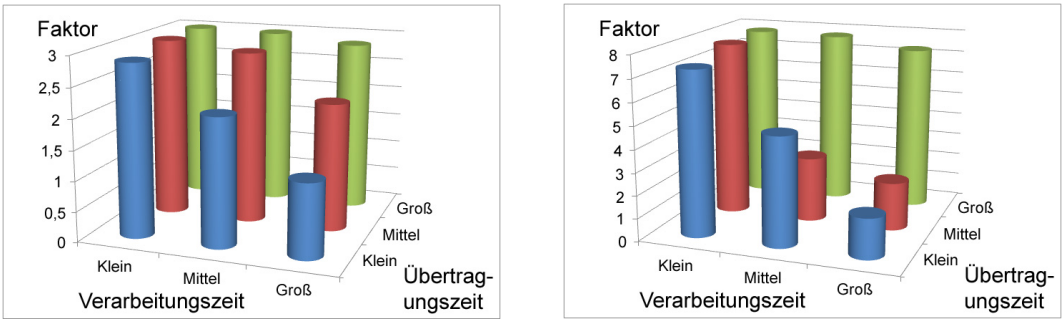
ΔT	**Klein**	**Mittel**	**Groß**
Klein	2,86	2,12	1,23
Mittel	2,99	2,86	2,12
Groß	3,00	2,98	2,85

(a) Faktor Ost USA – West Europa

ΔT	**Klein**	**Mittel**	**Groß**
Klein	7,31	4,82	1,77
Mittel	7,75	2,86	2,12
Groß	7,80	7,75	7,31

(b) Faktor Ost Asien – West Europa

Tabelle 7.6: Verbesserungsfaktoren der Antwortzeit bei Allokationsverschiebung von Ost USA bzw. von Ost Asien nach West Europa



(a) Zeitgewinn Ost USA – West Europa

(b) Zeitgewinn Ost Asien – West Europa

Abbildung 7.9: Verarbeitungs- und Übertragungszeit bei Verschiebung der Kapazität von Ost USA bzw. von Ost Asien nach West Europa

visualisiert: Mit steigender übertragener Datenmenge verbessern sich auch die Gewinne bei den Antwortzeiten. Das beste Ergebnis erzielt die Kombination der Kategorie *Groß* in der Übertragungszeit und der Kategorie *Klein* in der Verarbeitungszeit. Damit wird der enorme Einfluss der Verarbeitungszeiten auf die ermittelten Verbesserungsfaktoren klar.

Insgesamt zeigen die Simulationen unterschiedliche Round Trip Delays zu den verteilten Server-Regionen. Diese Variationen lassen sich durch verschiedene Ursachen erklären. Gründe können beispielsweise die variierende Anzahl der Übertragungsknoten zwischen Client und Server oder generell der unterschiedliche Ausbau der Netzinfrastruktur sein.

Abschließend ist festzustellen, dass sich bei der Verschiebung der Ressourcen in die nahe Umgebung des Benutzers in jedem Fall für ihn eine erhebliche Verbesserung der Antwortzeit des angefragten Cloud-Dienstes ergibt. Die Verzögerungszeit sinkt erheblich, wenn die Ressourcenallokation in derselben Region erfolgt, wo der Ursprung der Last liegt. Diese kürzeren Antwortzeiten steigern die Nutzerzufriedenheit sowie die Popularität des Dienstes. Infolgedessen sind höhere Gewinne erzielbar. Im umgekehrten Fall führen schlechte Performanzwerte zu geringerer Beliebtheit. Sie sind mit Umsatzeinbußen verknüpft.

7.4 Mathematische Analyse und Entwicklung der Kostenfunktionen

Nachdem mithilfe der Simulationen eine konkrete Datengrundlage geschaffen wurde, folgt nun die mathematische Analyse der Ergebnisse. Diese weitere Überprüfung dient der Entwicklung bedeutender Funktionen. Sie lassen einerseits die Berechnung der genauen Anzahl benötigter Ressourcen für einen festgelegten Zeitpunkt zu. Andererseits machen sie die entstehenden Kosten aus Nutzer- und Anbieter-Sicht ermittelbar. Die anfallenden Kosten für den

Betreiber und eine hohe Zufriedenheit der Nutzer bei großzügigem Kapazitätsvolumen stehen sich dabei gegenüber. Zur Lösung des Konflikts wird eine Formel abgeleitet. Sie bestimmt rechnerisch das optimale Mittel der Ressourcenmenge für einen Cloud-Dienst.
Gemäß dem entwickelten Algorithmus erfolgt die Allokation von Ressourcen abhängig von der Korrelationswahrscheinlichkeit zwischen Popularität eines SaaS-Angebots in sozialen Netzen und gemessener Last auf den bearbeitenden Servern. Der Anbieter kann dabei festlegen, ab welchem durch den Chi-Quadrat-Test erfassten Wahrscheinlichkeitswert die Anwendung hoch- und auch herunterskaliert werden soll.
Diese Bestimmung ist äußerst kritisch. Wenn der Schwellwert zur Bestätigung der Korrelationshypothese beim Herunterfahren virtueller Maschinen zu hoch gesetzt ist, läuft der Anbieter Gefahr, unnötig ungenutzte Ressourcen im Idle-Modus vorzuhalten. Dadurch entstehen eigentlich vermeidbare Kosten. Andernfalls besteht bei einer zu niedrigen Wahrscheinlichkeitsgrenze das Risiko, dass eventuell zu viele Instanzen heruntergefahren und nicht mehr alle Nutzer bestmöglich bedient werden.
Bei der Zuschaltung gilt der umgekehrte Fall. Wird ihre Grenze hoch angesetzt, findet nur bei enormer Laststeigerung eine Allokation statt. Da so seltener Instanzen hochgefahren werden, steigen die Kosten nicht unnötig. Vor allem in erheblichen Lastsituationen verschlechtern sich damit aber auch die Antwortzeiten einer Region für die Nutzer, da seltener Korrelationen erkannt werden. Grundlegend ist dann ein geringeres Ressourcenvolumen als tatsächlich benötigt in der nutzerspezifischen Region verfügbar. Durch das folglich notwendige Anfragen einer entfernteren Region, die ebenfalls das gewünschte SaaS-Angebot hostet, ergeben sich höhere Datenübertragungszeiten.
Im Rahmen dieses Kapitels wird eine Lösung entwickelt, die für diese Problemstellung ein geeignetes Mittel findet. Dafür wird mathematisch abgeschätzt, wie sich die Werte der Antwortzeiten im Zeitverlauf t konkret verhalten, wenn eine eigentlich erforderliche Ressourcenzuschaltung aufgrund falsch gesetzter Schwellwerte nicht erfolgt. Für die Berechnung werden die folgenden Variablen und Funktionen benötigt.

- $T_S(t)$ bezeichnet die Verarbeitungszeit der Server zum Zeitpunkt t.
- $L(t)$ legt die Jobanzahl in der Warteschlange zum Zeitpunkt t fest.
- T_J beschreibt die Berechnungsdauer für einen Job.
- $f_{Tweet}(t) = e^{\alpha t} + \beta$ stellt die bereits erläuterte exponentielle Funktion (7.1) des Tweet-Wachstums dar.
- R bezeichnet die Anzahl der zur Verfügung stehenden Ressourcen.
- t_{Shift} ist der zeitliche Versatz zwischen der Tweet- und der Lastkurve.
- t_{χ^2} bestimmt die Dauer bis zum nächsten Chi-Quadrat-Test.

Die Server-Verarbeitungszeit T_S ergibt sich aus den zu verarbeitenden Rechenjobs in der Warteschlange $L(t)$ – sprich der Last – verteilt auf die verfügbaren Ressourcen R multipliziert mit der zugehörigen Verarbeitungszeit je Job T_J:

$$T_S(t) = \frac{L(t)}{R} \cdot T_J \tag{7.3}$$

Die Evaluationsresultate in Abschnitt 7.3.1 belegen, dass exponentielle Wachstumsvorgänge mit gleichem Wachstumsfaktor α von unterschiedlicher Ausprägung γ im zeitlichen Versatz von t_{Shift} gut korrelieren. Im weiteren Verlauf soll davon ausgegangen werden, dass ein Zusammenhang zwischen der Arbeitslast und der Tweet-Popularität besteht. Zum Zeitpunkt t gilt die Lastfunktion (7.4) mit $\beta' = \gamma \cdot \beta$. Dabei ist β' ein zum zugrundeliegenden β der Tweet-Kurve unterschiedliches Anfangsniveau. $\gamma > 0$ beschreibt das γ-Vielfache des Wachstums der Tweet-Erwähnungen.

$$L(t) = \gamma \cdot e^{\alpha(t - t_{Shift})} + \beta' \tag{7.4}$$

Grundsätzlich sind bei der Lastbewältigung bestimmte Zeitkonstanten zu berücksichtigen. Neben der Dauer zum Hochfahren der Ressource wird auch eine gewisse Zeit benötigt, bis sie tatsächlich einsatzfähig ist. Diese Bootzeit ist abhängig von der Art des Betriebssystems und der Imagegröße der Instanz. Ein Image ist eine Datei, die den standardmäßigen Grundstand der virtuellen Komponente bei der Installation automatisch aufspielt.
Die Forschungsarbeiten von Mao et al. [105] zeigen, dass die Gesamtzeit zur vollständigen Aktivierung der Maschinen auch heute noch bis zu dreizehn Minuten dauert. Aufbauend auf dieser Erkenntnis wird für die weiterführende Evaluation eine zusätzliche Variable t_{VM} eingeführt. Sie beschreibt diese zeitliche Verzögerung von der Anfrage zusätzlicher Kapazität zum Zeitpunkt t bis zur tatsächlichen Zugriffsmöglichkeit auf die Ressource.
Abbildung 7.10 veranschaulicht die einzelnen Zeit- und Kurvenparameter. Wie hierin erkennbar, korrelieren die zum Zeitpunkt t erfasste Lastkurve $L(t)$ und die Tweet-Wachstumsfunktion $f_{Tweet}(t)$ mit einem zeitlichen Versatz um die

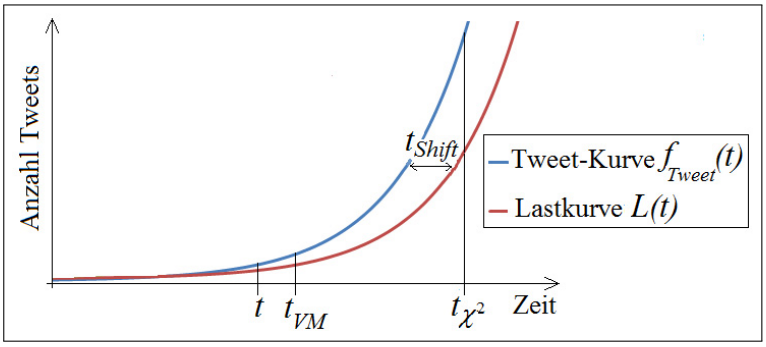


Abbildung 7.10: Visuelle Darstellung der Kurvenparameter

Variable t_{Shift}. Der Zeitpunkt t_{χ^2} signalisiert, dass die aktuelle Periode, in der von einer Korrelation ausgegangen wird, beendet ist. An diesem Punkt muss also eine erneute Überprüfung des Zusammenhangs zwischen Arbeitslast und Tweet-Erwähnungen erfolgen.
Wenn nun allerdings beim Betreiber ein zu hoher Schwellwert eingestellt ist, kann die gegebene Korrelation nicht per Chi-Quadrat-Test erkannt werden. Wie die weitere Analyse zeigt, hat dies weitreichende Folgen für die Antwortzeiten der Dienstanfragen.
Bei zu großen Grenzwerten muss die nächste Prüfsequenz nach dem Zeitintervall t_{χ^2} abgewartet werden. An diesem Punkt besteht nämlich wiederum die Chance, dass die eigentlich vorliegende Korrelation letztlich doch erkannt wird. Dies gilt allerdings nur, wenn sich die beiden Kurven in dieser Periode bis zur erneuten Überprüfung weiter angenähert haben. Dadurch steigt die Wahrscheinlichkeit der Korrelation im Chi-Quadrat-Test und der hoch angesetzte Schwellwert kann eventuell erreicht werden.
Im Folgenden wird bestimmt, wie sich die Verarbeitungszeit bei nicht erkannter Korrelation im Zeitintervall bis zum nächsten Chi-Quadrat-Test t_{χ^2} entwickelt. Die dafür ermittelte Formel berechnet die Verarbeitungszeit T_S in der Überprüfungsperiode und nach der Aktivierung weiterer Maschinen, also zum Zeitpunkt $t_{\chi^2} + t_{VM}$. Zunächst ergibt sich aus Formel (7.4) für die Last zum Zeitpunkt $t + t_{\chi^2} + t_{VM}$:

$$L(t + t_{\chi^2} + t_{VM}) = \gamma \cdot e^{\alpha(t+t_{\chi^2}+t_{VM}-t_{Shift})} + \beta' \tag{7.5}$$

Analog zu Formel (7.3) gilt für die Verarbeitungszeit T_S zu diesem Zeitpunkt:

$$\begin{aligned} T_S(t + t_{\chi^2} + t_{VM}) &= \frac{L(t + t_{\chi^2} + t_{VM})}{R} \cdot T_J \\ &\overset{7.5}{=} \frac{(\gamma \cdot e^{\alpha(t+t_{\chi^2}+t_{VM}-t_{Shift})} + \beta')}{R} \cdot T_J \end{aligned} \tag{7.6}$$

Wenn man daraufhin die Verarbeitungszeit $T_S(t + t_{\chi^2} + t_{VM})$ im Zeitpunkt $t + t_{\chi^2} + t_{VM}$ ins Verhältnis zur Verarbeitungszeit $T_S(t)$ zu Beginn der Bearbeitung setzt, kürzt sich die Variable der aktuellen Ressourcenanzahl heraus. Die so entstehende Schätzungsformel (7.7) kann die Verschlechterung der Verarbeitungszeit bestimmen.

$$\frac{T_S(t + t_{\chi^2} + t_{VM})}{T_S(t)} = \frac{(\gamma \cdot e^{\alpha(t+t_{\chi^2}+t_{VM}-t_{Shift})} + \beta') \cdot T_J}{(\gamma \cdot e^{\alpha(t-t_{Shift})} + \beta') \cdot T_J} \tag{7.7}$$

Im Laufe der Zeit ist β' vernachlässigbar, da mit den Gesetzen des exponentiellen Wachstums $\gamma \cdot e^{\alpha t} \gg \beta'$ gilt. Zudem können T_J sowie γ gekürzt werden. Nach entsprechender Auflösung mit der Rechenregel der Exponentialfunktionen $e^{a+b} = e^a \cdot e^b$ entfallen auch e^t und $e^{t_{Shift}}$. Folglich ergibt sich für das Verhältnis aus Formel (7.7):

$$\frac{T_S(t + t_{\chi^2} + t_{VM})}{T_S(t)} \approx \frac{e^{\alpha(t+t_{\chi^2}+t_{VM}-t_{Shift})}}{e^{\alpha(t-t_{Shift})}} = e^{\alpha(t_{\chi^2}+t_{VM})}$$

Daraus kann für die Verarbeitungszeit $T_S(t + t_{\chi^2} + t_{VM})$ gefolgert werden:

$$T_S(t + t_{\chi^2} + t_{VM}) \approx e^{\alpha(t_{\chi^2}+t_{VM})} \cdot T_S(t) \tag{7.8}$$

Man nehme nun an, die Arbeitslast wächst kontinuierlich exponentiell weiter an. Allerdings werden vorsorglich keine zusätzlichen Ressourcen hochgefahren und eingerichtet. Mit dem Ausdruck (7.8) lässt sich für den Zeitraum bis zum nächsten Chi-Quadrat-Test und der erforderlichen Bootzeit nach schlussendlicher Korrelationserkennung die entstehende Verarbeitungszeit feststellen. Demnach steigt in diesem Fall auch die Zeit der Diensterarbeitung exponentiell an. Die Zeitspannen t_{χ^2} zwischen den einzelnen Chi-Quadrat-Korrelationstests und die Bootdauern t_{VM} zusätzlicher Maschinen tragen entscheidend zum Anwachsen der Verarbeitungszeit T_S bei. Im Interesse der SaaS-Kunden sind sie folglich möglichst kurz zu halten.
Falls bei einer Kurvenkorrelation der Grenzwert hingegen so eingestellt ist, dass der Chi-Quadrat-Test positiv ausfällt, kann eine korrekte Ressourcenallokation stattfinden. Bislang wurde mit dem Fokus auf die Verarbeitungszeit T_S lediglich die Anbieterseite mit der entsprechenden Rechenkraft ihrer Server betrachtet. Die Antwortzeit T einer Cloud-Applikation ist jedoch auch von der Dauer der Datenübertragung T_D zwischen Client und Server abhängig. Es gilt:

$$T(t) = T_D(t) + T_S(t) \tag{7.9}$$

Diese Übertragungszeit $T_D(t)$ wird durch die Entfernung beeinflusst, die zwischen den bereitgestellten Ressourcen des SaaS-Angebots und den anfragenden Clients besteht. Da sich diese in unterschiedlichen Regionen befinden können, ist es möglich, dass ein erheblicher Verzögerungsfaktor entsteht. Bis im Falle von steigender Tweet-Last und der Korrelationserkennung die zusätzlich benötigten Ressourcen zur Verfügung stehen, steigt die Verarbeitungszeit der Server exponentiell weiter, wie in Formel (7.8) angegeben. Basierend auf Formel 7.9 folgt für die Antwortzeit:

$$T(t + t_{\chi^2} + t_{VM}) \overset{7.8}{=} T_D(t) + e^{\alpha(t_{\chi^2}+t_{VM})} \cdot T_S(t) \tag{7.10}$$

Im Folgenden beschreibt t_{Alloc} den Zeitpunkt, ab dem die hinzugefügten Ressourcen tatsächlich zur Verfügung stehen. Des Weiteren ist das im vorangegangenen Abschnitt ermittelte Verhältnis der Round Trip Delays über den Parameter q ausgedrückt. Zum Zeitpunkt $t_{Alloc} \geq t + t_{\chi^2} + t_{VM}$ steht dem anfragenden Nutzer die angeforderte Kapazität in genau der Region zur Verfügung, wo der Lastanstieg verzeichnet wird. Dabei gilt für die Übertragungszeit $T_D(t_{Alloc}) = \frac{1}{q} \cdot T_D(t)$. Die Transferdauer verbessert sich also durch die Zuschaltung der SaaS-Angebot-unterstützenden Kapazitäten in der nahen Region des Clients um den Faktor q. Dieser Fakt ist mit den bereits erläuterten Simulationsergebnissen belegbar.
Die Server-Verarbeitungszeit ist analog zu Formel (7.3) $T_S(t_{Alloc}) = \frac{L(t_{Alloc})}{R} \cdot T_J$. Da zu diesem Zeitpunkt zusätzliche Ressourcen R bereitstehen, kann die Überlast schließlich ausgeglichen werden. Nach der Aktivierung weiterer Maschinen ergibt sich die Antwortzeit: $T(t_{Alloc}) = \frac{1}{q} \cdot T_D(t) + \frac{L(t_{Alloc})}{R} \cdot T_J$. Für den Vergleich zwischen $T(t_{Alloc})$ und $T(t + t_{\chi^2} + t_{VM})$ gilt folgendes Verhältnis:

$$\frac{T(t_{Alloc})}{T(t + t_{\chi^2} + t_{VM})} \overset{7.10}{=} \frac{\frac{1}{q} \cdot T_D(t) + \frac{L(t_{Alloc})}{R} \cdot T_J}{T_D(t) + e^{\alpha(t_{\chi^2} + t_{VM})} \cdot T_S(t)}$$

$$T(t_{Alloc}) = T(t + t_{\chi^2} + t_{VM}) \cdot \frac{\frac{1}{q} \cdot T_D(t) + \frac{L(t_{Alloc})}{R} \cdot T_J}{T_D(t) + e^{\alpha(t_{\chi^2} + t_{VM})} \cdot T_S(t)} \tag{7.11}$$

Je geringer der Round Trip Delay, desto mehr verkürzt sich die Antwortzeit zum Zeitpunkt zusätzlich verfügbarer Ressourcen. Aus Formel (7.11) lässt sich außerdem folgern, dass bei steigendem RTD-Verbesserungsfaktor q die Datenübertragungszeit deutlich sinkt. Insgesamt ergibt sich so nach dem Instanzenstart in der nahen Client-Region eine merklich kürzere Gesamtantwortzeit.
Damit ist die vielversprechende Auswirkung der ortsbasierten Allokation von Ressourcen in unmittelbarer Client-Umgebung auf die Dienstantwortzeit bewiesen. Unklar ist bis jetzt allerdings, wie viele Ressourcen bei steigender Last konkret hochgefahren werden müssen.
Wie bereits angedeutet, stehen sich hierbei zwei Argumentationsseiten gegenüber. Zum einen will der SaaS-Anbieter seinen Dienst mit möglichst wenig Rechenkapazität betreiben, da er für jede Instanz der zugrundeliegenden Cloud-Infrastruktur zahlen muss. Großzügig zusätzliche und dann möglicherweise ungenützte Server-Ressourcen anzufordern, ist demzufolge mit unnötigen Kosten verbunden. Andererseits ist es im Interesse des Nutzers, wenn er auf einen maximal performanten Cloud-Dienst Zugriff hat. In diesem Falle steigen auch die Nutzerzufriedenheit und infolgedessen die Beliebtheit des SaaS-Angebots. Das kann sich wiederum in einem positiven Effekt auf die gewinnbringenden Einnahmen des Dienstbetreibers widerspiegeln.

Folglich ist es notwendig, ein optimales Mittel zwischen diesen beiden offenbar gegensätzlichen Interessen zu finden. Aus diesem Grund wird zunächst eine Funktion bestimmt, welche die nötige Ressourcenanzahl festlegt. Die anschließend ermittelten Kostenfunktionen für Nutzer und Cloud-Anbieter machen damit genau dieses Optimum berechenbar.

Auch im Folgenden bezeichnet f_{Tweet} die Exponentialfunktion der Tweet-Kurve, also das regionsspezifische Popularitätswachstum eines Cloud-Dienstes in sozialen Netzen. Weiterhin wird festgelegt, dass der Chi-Quadrat-Test eine Korrelation mit der Lastkurve L der anfragenverarbeitenden Server erkennt. Sie ist zur Tweet-Kurve f_{Tweet} zeitlich um t_{Shift} versetzt und entspricht ihrem γ-Vielfachen. Daraus folgt, dass der Ursprung der Arbeitslast ebenfalls in der Region der korrelierenden Tweet-Kurve liegt.

Wie vorab bewiesen, steigt die Last während der Bootzeit t_{VM} und der Dauer t_{χ^2} bis zur nächsten Korrelationsüberprüfung weiter. Folglich ist es für den Anbieter optimal, wenn er die Last bereits bis zum nächsten Test inklusive der Maschinenstartzeit aus dem Tweet-Verlauf vorhersagen kann. So sind bereits zum nächsten Prüfzeitpunkt ausreichend Ressourcen zur Verfügung zu stellen. Zu diesem Zeitpunkt ist der Tweet-Verlauf jedoch noch nicht bekannt. Auch er muss noch im Laufe der Prüfperiode bestimmt werden. Näherungsweise wird deshalb mit den bekannten Werten für den Zeitpunkt $t + t_{VM}$ gerechnet.

$R(t)$ beschreibt die zum Zeitpunkt t nötige Ressourcenmenge. Der Dienstanbieter kann mit dem Faktor g beeinflussen, in welchem Umfang er Ressourcen im Verhältnis zur auftretenden Last hochfahren will.

$$R(t) = g \cdot L(t + t_{VM}) = g \cdot \gamma \cdot f_{Tweet}(t + t_{VM} - t_{Shift}) \tag{7.12}$$

Im folgenden Verlauf bezeichnet k_A den Kostenpunkt für eine Ressourceninstanz pro Zeiteinheit. Mit der Formel der Ressourcenanzahl (7.12) lassen sich schließlich die Kosten K_A für den Anbieter berechnen als:

$$\begin{aligned} K_A &= R(t) \cdot t_{\chi^2} \cdot k_A \\ &\overset{7.12}{=} g \cdot \gamma \cdot f_{Tweet}(t + t_{VM} - t_{Shift}) \cdot (t_{\chi^2} - t_{VM}) \cdot k_A \end{aligned} \tag{7.13}$$

Wie erwähnt, ist γ das um t_{Shift} zeitlich versetzte Vielfache der Lastkurve bezogen auf die Tweet-Kurve. So lässt sich γ wie folgt darstellen:

$$\gamma = \frac{L(t)}{f_{Tweet}(t - t_{Shift})} \tag{7.14}$$

Wird nun dieser Ausdruck (7.14) des Parameters γ in der Kostengleichung einsetzt, ergibt sich für die Anbieterkosten:

$$K_A \overset{7.14}{=} g \cdot \frac{L(t)}{f_{Tweet}(t - t_{Shift})} \cdot f_{Tweet}(t + t_{VM} - t_{Shift}) \cdot (t_{\chi^2} - t_{VM}) \cdot k_A \tag{7.13}$$

Bei Betrachtung der Nutzerseite, stellt sich heraus, dass auch hier Kosten entstehen. k_N ist dabei der Kostenfaktor für den Benutzer. Dient ein Cloud-Service dem Nutzer für kommerzielle Zwecke stellt dieser Faktor beispielsweise die Kosten pro Arbeitszeit in einem Unternehmen dar. Die Last entspricht der Anzahl an Nutzern im System zum Zeitpunkt $t + t_{VM}$. Die Summe aller Benutzeranfragen ergibt die Kosten K_N. Sie werden folgendermaßen bestimmt:

$$K_N = T_S(t) \cdot k_N \cdot L(t + t_{VM}) \tag{7.15}$$

Auf welche Weise die Last verteilt wird und wie viel Ressourcenkapazität dafür zur Verfügung steht, ist über die Verarbeitungszeit der Server T_S bestimmt. Nachfolgend wird davon ausgegangen, dass die Lastaussteuerung gleichermaßen auf allen bereitstehenden Ressourcen erfolgt. Dann ist die Verarbeitungszeit T_S ausdrückbar als Last $L(t + t_{VM})$ dividiert durch die Ressourcenmenge $R(t)$ multipliziert mit der Rechenzeit der einzelnen Rechenjobs T_J. Außerdem ist analog zu Formel 7.5 die Lastfunktion in abgewandelter Form ersetzbar durch $L(t + t_{VM}) = \gamma \cdot f_{Tweet}(t + t_{VM} - t_{Shift})$. Zur genaueren Ermittlung der Nutzerkosten ergibt sich die Gleichung:

$$\begin{aligned}
K_N &= \frac{L(t + t_{VM})}{R(t)} \cdot T_J \cdot k_N \cdot L(t + t_{VM}) \\
&= \frac{L(t + t_VM)}{R(t)} \cdot T_J \cdot k_N \cdot \gamma \cdot f_{Tweet}(t + t_{VM} - t_{Shift}) \\
&= \frac{\gamma \cdot f_{Tweet}(t + t_{VM} - t_{Shift})}{R(t)} \cdot T_J \cdot k_N \\
&\quad \cdot \gamma \cdot f_{Tweet}(t + t_{VM} - t_{Shift}) \\
&\overset{7.12}{=} \frac{\gamma \cdot f_{Tweet}(t + t_{VM} - t_{Shift})}{g \cdot \gamma \cdot f_{Tweet}(t + t_{VM} - t_{Shift})} \cdot T_J \cdot k_N \\
&\quad \cdot \gamma \cdot f_{Tweet}(t + t_{VM} - t_{Shift}) \\
&= \frac{1}{g} \cdot T_J \cdot k_N \cdot \gamma \cdot f_{Tweet}(t + t_{VM} - t_{Shift}) \\
&\overset{7.14}{=} \frac{1}{g} \cdot T_J \cdot k_N \cdot \frac{L(t)}{f_{Tweet}(t - t_{Shift})} \cdot f_{Tweet}(t + t_{VM} - t_{Shift})
\end{aligned} \tag{7.15}$$

Damit sind die zwei sich gegenüberstehen Kostenfunktionen für den Nutzer (7.15) und den Anbieter (7.13) erstellt. Sie hängen beide vom Faktor g ab und schneiden sich exakt in diesem Punkt. Dies ist in Abbildung 7.11 visualisiert. Der Schnittpunkt ist als Kompromiss zu verstehen zwischen einer großzügigen

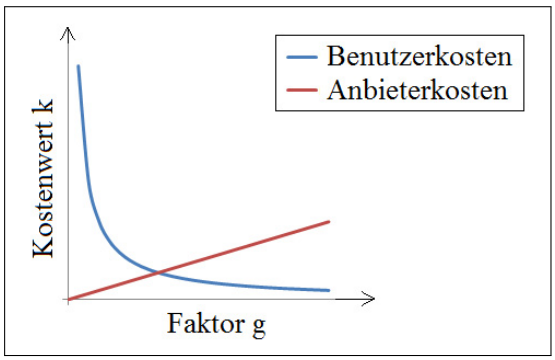


Abbildung 7.11: Verhalten der Kostenfunktionen

Ressourcenallokation sowie damit einhergehenden hohen Nutzerzufriedenheit und einer zurückhaltenden Vorgehensweise mit geringeren Kosten auf Anbieterseite. Dieser optimierte Mittelweg ist über die Gleichsetzung der beiden Kostenfunktionen $K_N = K_A$ auf diese Weise darstellbar:

$$\begin{aligned} &\frac{1}{g} \cdot T_J \cdot k_N \cdot \frac{L(t)}{f_{Tweet}(t - t_{Shift})} \cdot f_{Tweet}(t + t_{VM} - t_{Shift}) \\ &= \; g \cdot \frac{L(t)}{f_{Tweet}(t - t_{Shift})} \cdot f_{Tweet}(t + t_{VM} - t_{Shift}) \cdot (t_{\chi^2} - t_{VM}) \cdot k_A \end{aligned} \tag{7.16}$$

Bei der Auflösung der Gleichung ergibt sich so für das optimale Mittel g:

$$g = \sqrt{\frac{k_N}{k_A} \cdot \frac{T_J}{t_{\chi^2} - t_{VM}}} \tag{7.17}$$

Folglich lässt sich ein ausgewogenes Verhältnis der bereitzustellenden Ressourcenmenge dynamisch berechnen. Dies stellt eine Kompromisslösung zwischen der Nutzer- und der Anbieter-Sichtweise dar. Die Kostenfaktoren von Anbieter und Nutzer werden gleichermaßen berücksichtigt. Über die Job-Rechendauer T_J findet die Anwendungskomplexität Beachtung. Weiterhin wird die Periodendauer t_{χ^2} bis zum nächsten Chi-Quadrat-Test, die Bootdauer t_{VM} und der zeitliche Versatz zwischen Last- und Tweet-Kurve einbezogen.
Um nun einen konkreten Wert für die optimale Ressourcenanzahl zu ermitteln, können die aktuell verzeichneten Lastwerte $L(t)$ sowie die Tweet-Kurven $f_{Tweet}(t - t_{Shift})$ und $f_{Tweet}(t + t_{VM} - t_{Shift})$ aus der Datenbank ausgelesen werden. Für die zu allokierende Ressourcenmenge ergibt sich zum Zeitpunkt t:

$$R(t) = \sqrt{\frac{k_N}{k_A} \cdot \frac{T_J}{t_{\chi^2}}} \cdot \frac{L(t)}{f_{Tweet}(t - t_{Shift})} \cdot f_{Tweet}(t + t_{VM} - t_{Shift}) \tag{7.18}$$

Gemäß dem Algorithmus erfolgt eine Skalierung lediglich im Falle eines positiven Abgleichs in der vorherigen Lastmuster-Analyse. Hierbei werden Steigungsänderungen der Kurven erkannt. Danach ist es möglich, die Fälle steigender Verläufe von denen fallender Kurven zu unterscheiden. Auf dieser Basis ist definierbar, ob aus dem bestehenden Pool an Ressourcen R Instanzen hoch- oder herunterzufahren sind. Die Anzahl zukünftig benötigter Ressourcen $R(t)$ ist über die aufgestellte Formel (7.18) festgelegt.
Im Falle eines Anstiegs ist mit $\Delta R = R(t) - R$ bestimmbar, wie viele Ressourcen zuzuschalten sind. Bei abfallender Last können umgekehrt $\Delta R = R - R(t)$ Ressourcen abgeschaltet werden. So wird dynamisch sowohl auf ansteigende als auch auf abfallende Last reagiert. Mithilfe der flexibel bestimmbaren Variable g ist zudem eine ausgewogene Allokation von Ressourcen realisierbar.
Zum Abschluss des theoretischen Teils zeigt das folgende Beispielszenario, wie erfolgversprechend die entwickelten Formeln sind. Die dafür benötigten Parameterwerte werden so belegt: der Kostenfaktor der Benutzer beläuft sich auf $k_N = 100\,e/h$ und der Kostenfaktor des Anbieters liegt bei $k_A = 5\,e/h$. Die Verarbeitungszeit eines Jobs beträgt $T_J = 0.1s$ und die Periodendauer zwischen den Chi-Quadrat-Tests ist $t_{\chi^2} = 3600s$. Für die Last wird ein Wert von $L(t) = 326$ ermittelt. Die Anzahl der Tweets wird mit $f_{Tweet}(t - t_{Shift}) = 104$ sowie $f_{Tweet}(t + t_{VM} - t_{Shift}) = 143$ in der Datenbank ausgelesen. $R(t)$ und die Verarbeitungszeit T_S berechnen sich wie folgt:

$$
\begin{aligned}
R(t) &= \sqrt{\frac{k_N}{k_A} \cdot \frac{T_J}{t_{\chi^2}}} \cdot \frac{L(t)}{f_{Tweet}(t - t_{Shift})} \cdot f_{Tweet}(t + t_{VM} - t_{Shift}) \\
&= \sqrt{\frac{100\,e/h}{5\,e/h} \cdot \frac{0.1s}{3600s}} \cdot \frac{326}{104} \cdot 143 \approx 0.0236 \cdot \frac{326}{104} \cdot 143 \approx 12 \\
T_S &= \frac{L(t + t_{VM})}{R(t)} \cdot T_J = \frac{\gamma \cdot f_{Tweet}(t + t_{VM} - t_{Shift})}{R(t)} \\
&= \frac{\frac{326}{104} \cdot 143}{12} \cdot 0{,}1\text{s} \approx 4{,}1\text{s}
\end{aligned}
$$

Um die zur Tweet-Rate korrelierende Last zu bewältigen, müssen zum Zeitpunkt t in der betrachteten Region zwölf Instanzen verfügbar sein. Werden die zwölf Ressourcen ortsbezogen rechtzeitig gestartet, belaufen sich die Verarbeitungszeiten der beanspruchten Server T_S auf etwas mehr als vier Sekunden.
Falls aufgrund eines hohen Schwellwerts der Wahrscheinlichkeit irrtümlich keine zusätzlichen Ressourcen gebootet werden, obwohl die korrelierende Last steigt, verschlechtert sich diese Verarbeitungszeit. Der Provider spart zwar mit einer hoch eingestellten Grenze scheinbar Kosten ein, da weniger Instanzen betrieben werden. Allerdings wächst im schlimmsten Fall die Server-

Verarbeitungszeit exponentiell – auch weil das Booten der Instanzen so zeitintensiv ist. Das kann sich negativ auf das Image des Anbieters auswirken und zu geringerer Popularität seiner Anwendungen führen.
Die entwickelten Funktionen ermöglichen insgesamt eine Ermittlung der Anzahl benötigter Ressourcen für einen bestimmten Zeitpunkt. Zudem ist die aus diesem Kapazitätsvolumen resultierende Verarbeitungszeit feststellbar.

7.5 Zusammenfassung

Die Skalierungsprozesse benötigter Instanzen sind in Cloud-Systemen mit erheblichem Aufwand verbunden. Bis heute bestehen keine voll automatisierten Regelsysteme. So erfordert die Ressourcenregulierung den manuellen Eingriff mittels durch den Nutzer definierter Anweisungslogik. Zudem ist das Zuschalten von zusätzlicher Kapazität mit einer zeitlichen Verzögerung von bis zu einer Viertelstunde verbunden. Meist wird deshalb zu viel Kapazität vorgehalten, was effizientes Ressourcenmanagement erschwert.
In diesem Kapitel wurde dafür ein Algorithmus entwickelt. Er widmet sich der Lösung, wo wann wie viele Ressourcen im Falle eines Lastanstiegs gestartet und bei Abfallen der Arbeitslast heruntergefahren werden sollen. Die entworfene Methode ermittelt zunächst, ob ein Wachstum im Lastaufkommen zu erwarten ist. Er bestimmt global das Nutzerinteresse an der betrachteten Anwendung mithilfe von sozialen Web-Applikationen wie Twitter. Daraus wird der weitere Verlauf der verbundenen Ressourcenlast abgeleitet. Die Evaluation zeigt, dass eine Korrelationserkennung zwischen der Popularitätskurve von Tweets und dem Lastverlauf durch den Chi-Quadrat-Test möglich ist.
Der Anbieter kann den Schwellwert der Korrelationswahrscheinlichkeit frei wählen. Zu beachten ist einerseits, dass bei großzügiger Ressourcenbereitstellung möglicherweise teure Kapazität vergeudet wird. Andererseits zeigt die mathematische Analyse bei zurückhaltenden Strategien schlechte Antwortzeiten – im schlimmsten Fall mit exponentiellem Wachstum. Der Anbieter riskiert so eine Abnahme an Nutzern.
Zur Lokalisation des genauen Orts mit steigendem Ressourcenbedarf ermittelt der Algorithmus, aus welcher Region die meisten Tweets kommen. Werden zusätzliche Instanzen in der korrespondierenden Umgebung gestartet, entfallen unnötige Latenzzeiten aufgrund geringerer Routing-Entfernungen. Die Simulationsergebnisse zur Bestimmung der Antwortzeiten in Clouds zeigen, dass sich die ortsbezogene Ressourcenallokation positiv auf die Antwortzeit auswirkt.
Wie viele Ressourcen in einer Region bereitzustellen sind, wurde im Verlauf des Kapitels mithilfe der Korrelation zur gemessenen Popularität in sozialen Netzwerken festgestellt. Die hergeleitete Formel berechnet dynamisch die Menge zu startender Ressourcen. Dies erfolgt unter anderem anhand der aktuellen Last- und Popularitätskurve sowie der Bootzeit und der Kostenfaktoren bei Nutzer und Anbieter. Der damit definierte Trade-off balanciert die Kosten so-

wohl für den Anbieter als auch für den Nutzer optimal. Der Anbieter kann die Ressourcen flexibel so aussteuern, dass bei geringen Kosten eine möglichst schnelle Systemreaktion erfolgt.
Auf der anderen Seite ist per Musteranalyse auch ein Abfall der Last feststellbar. Mit der entworfenen Formel wird die korrekte Anzahl an nicht mehr benötigten Instanzen vorausschauend heruntergefahren, ohne dass unnötige Kosten durch ungenutzte Kapazitäten im Leerlauf entstehen.
Insgesamt ermöglicht der im Rahmen dieser Arbeit entwickelte Algorithmus zum einen eine ortsbezogene, bessere Bearbeitung von nicht angemeldetem Ressourcenbedarf. Zum anderen wird eine Reduzierung der Antwortzeit während der Diensterbringung realisiert.

8 Zusammenfassung und Ausblick

Cloud Computing ist ein hochaktuelles Thema. Einerseits wird es von führenden IT-Unternehmen bereits großflächig eingesetzt, auf der andern Seite bietet es der Wissenschaft ein interessantes Forschungsfeld. Da verschiedene Aspekte die Technologie beeinflussen, existiert bis heute keine einheitliche Auslegung für Cloud Computing. In den ersten beiden Kapiteln dieser Arbeit wurden die bedeutendsten Begriffe und charakteristischen Merkmale herausgearbeitet.

Die Elastizität der Cloud ermöglicht eine flexible Aktivierung und Abgabe virtueller Maschinen. Damit sind theoretisch Unter- und Überprovisionierungen vermeidbar. Allerdings zeigen die im dritten Kapitel vorgestellten Forschungsarbeiten und kommerziellen Lösungen, dass die Überwachung und Zuteilung der Cloud-Ressourcen enorme Herausforderungen darstellen.

Cloud-Systeme sind erheblichen Lastschwankungen ausgesetzt. Da die Allokation von Ressourcen mit sehr langen Wartezeiten verknüpft ist, wird meist überprovisioniert. Dieser beliebig große Kapazitäten-Pool bringt enorme Kosten mit sich. Die Bestimmung mit welcher minimalen Ressourcenmenge ein Dienst maximal effizient erbracht werden kann, ist folglich essentiell. Das derzeit gängige regelbasierte Ressourcenmanagement stellt dafür nur unzureichende Automatismen. Sie erfordern die Definition und Kontrolle durch den Nutzer.

Wie im vierten Kapitel erläutert, sind bei der Behandlung anstehender Arbeitslast viele externe Faktoren zu berücksichtigen. Es müssen Gebühren, Lastvolumen, Bereitstellungsdauer und Service-Level-Agreements einkalkuliert werden. Die in dieser Arbeit identifizierten Lastmuster ermöglichen bei Laständerungen eine rechtzeitige Zuschaltung von Ressourcen.

Eine beliebige, auf mangelhafter Schätzung basierende Instanzenallokation führt neben hohen Kosten auch zu Energieverschwendung. Folglich sind bei der Aufschaltung des hochgerechneten Kapazitätsvolumens ebenfalls ökologische Aspekte zu beachten. Diese Arbeit begegnet den identifizierten Problemen, indem sie eine Lösung für die effiziente Lastverteilung und das zugehörige Ressourcenmanagement vorstellt. Sie realisiert die wohlproportionierte Zu- und Abschaltung von Ressourcen.

Der erste Schritt dieser Lösung beruht auf der aktiven, vorausschauenden Bedarfsangabe der Clients. Das im fünften Kapitel entwickelte Protokoll reserviert die nötigen Ressourcen und meldet dies an den Nutzer zurück. Das dafür entworfene Modell definiert den Kommunikationsaustausch der Komponenten und ermöglicht so ein sehr effizientes Ressourcenmanagement.

Durch die protokollspezifische Verzögerung gewisser Anfragen um die vom Client zulässige Zeit ergeben sich weniger ungenutzte Maschinen. Zudem sind zur Deckung des aktuellen Bedarfs seltener Systemskalierungen nötig. Somit werden alle Nutzer mit konstanteren Ressourcenmengen bedient. Gleichzeitig wird die Auslastung der bereitgestellten Kapazitäten erheblich verbessert.

Diese positiven Auswirkungen des Protokolls werden anhand mathematischer Analysen und Simulationen in den Abschnitten 5.4 und 5.5 bestätigt. Sie zeigen, dass sich die Dauer, bis alle Anfragen abgearbeitet sind, erheblich verkürzt. Außerdem wird aufgrund der Rückmeldung der Server an die Clients die Menge zu verwerfender Anfragen enorm reduziert.

Trotz der Lastspitzenglättung sind natürlicherweise nicht alle Skalierungsprozesse unerlässlich. Zur automatisierten Entscheidung, wie viele und welche Ressourcen zu allokieren sind, müssen die Abhängigkeiten der ineinandergreifenden Dienstkomponenten bekannt sein. Dafür wurde das Protokoll im zweiten Lösungsschritt erweitert. Wie im sechsten Kapitel dargelegt, erkennt der entwickelte Selbstkalibrierungsmechanismus die Beziehungen der Ressourcen automatisch und bildet sie in einem Abhängigkeitsgraphen ab. Da die einzelnen Auslastungen der Komponenten nicht überwacht werden müssen, minimiert sich der administrative Aufwand der Skalierung. Mithilfe dieses Verfahrens wird das Scheduling verbessert.

Die Testumgebung implementiert fünf Lastszenarien. Das entwickelte Verfahren wurde mit den geläufigen Scheduling-Strategien *Random* und *Round-Robin* verglichen. Die Simulationsresultate belegen, dass der entworfene Algorithmus bei der Anfragenverarbeitung stets besser abschneidet als die Standardverfahren. Die Maschinen benötigen insgesamt halb so viel Zeit. Dabei müssen keine oder nur wenige Tasks erneut bearbeitet werden. Außerdem skaliert das System mit der Erweiterung selbstständig auf ein Auslastungsoptimum, indem es nicht benötigte Kapazität vorausschauend abgibt.

Diese beiden Lösungsteile erzielen demnach ausgezeichnete Ergebnisse, wenn die Last angemessen reserviert ist. Die Aspekte, *wann wie viele* Kapazitäten allokiert werden sollen, sind lösbar. Daneben existieren Fälle, in denen die protokollspezifische Bedarfsreservierung und der optimierte Ressourcen-Schedule nicht umzusetzen sind, weil keine zukünftigen Lastinformationen existieren. Diese Fragestellung umfasst neben den bereits genannten auch den Allokationsaspekt, *wo* Instanzen benötigt werden. Wird die Instanzennutzung nicht aktiv angegeben, muss das Ressourcenmanagement eine Lastvorhersage durch Hochrechnung erfolgen. Der entworfene, im siebten Kapitel beschriebene Algorithmus ermöglicht dafür die Bestimmung des Ressourcenrahmens mithilfe sozialer Netzwerke. Die Prognose künftiger Lastentwicklung erfolgt durch eine Extrapolation der Nutzerdaten. Das entwickelte Verfahren identifiziert zunächst steigenden Lastumfang. Dann werden die Orte des Hauptbedarfs abgeleitet und dort zusätzliche Instanzen gestartet.

Wie die Evaluation zeigt, ist die dafür nötige Korrelationserkennung zwischen der Popularitätskurve einer Anwendung und dem Lastverlauf auf den zugehö-

rigen Rechnern mithilfe des Chi-Quadrat-Tests durchführbar. Die gemessenen Antwortzeiten der Simulationen untermauern, dass die vorgestellte Methode die optimale Ressourcenplatzierung sowie eine deutliche Latenzzeitreduktion bei ortsnaher Diensterbringung ermöglicht.

Zur Berechnung der Allokationsmenge wurde eine Formel entwickelt, die Zeitfaktoren und Gebühren berücksichtigt. Zusätzlich wurden Kostenfunktionen für Nutzer und Anbieter ausgearbeitet. Über diese wurde eine weitere Formel erstellt, die einen Kompromiss zwischen großzügiger Ressourcenallokation für hohe Nutzerzufriedenheit und zurückhaltender Handlungsweise für geringere Betreiberkosten ermittelt. Mit dem Algorithmus lässt sich so zum einen der Ressourcenbedarf ortsbezogen optimal decken. Zum anderen wird langen Wartezeiten während der Diensterbringung entgegen gesteuert.

Eine weitere mögliche Protokollerweiterung kann die Integration von Sonderangeboten der Betreibergebühren einbeziehen. Mittels Observer-Pattern sind die aktuellen Preise des zugrundeliegenden Anbieters überwachbar. Ob eine zurückhaltende oder eher großzügige Allokationsstrategie eingesetzt wird, erfolgt abhängig von eigens definierten Kostenschwellwerten. Diese speziellen Preisangebote werden erst seit kurzem forciert. Da sie noch enorm variieren, wurde von einer konkreten Umsetzung im Rahmen dieser Arbeit abgesehen.

Mit den entwickelten Methoden wird eine effizientere Ressourcenplanung realisierbar, die alle Komponenten bestmöglich auslastet. Dennoch bleiben weitere Herausforderungen bestehen. Sowohl die Instanzenallokation als auch die Umverteilung virtueller Kapazität auf ein und derselben Hardware beanspruchen trotz dieser Verbesserungen große Zeitspannen. Außerdem beherrscht der Anbieter nach wie vor die Systemumgebung maßgeblich, da er ausschlaggebenden Einfluss auf Sicherheitsmechanismen, Gebühren, Provisionierungszeiten und Allokationsorte hat. Er besitzt die Kontrolle über die Daten und kann entscheiden, wo sie repliziert und in welchem Format sie gespeichert werden.

Diese grundlegenden Fakten können auch die entworfenen Lösungsstrategien nicht umgehen. Zielführend wären intensivere Bemühungen um eine einheitliche Standardisierung der elementaren Cloud-Bestandteile. Obwohl Wissenschaftler und IT-Anbieter bereits einige Aspekte in den Grundzügen behandeln, bleiben noch viele Themen offen für weitere Forschung.

Schlussendlich zeigt die Evolutionsgeschichte der Computing-Ära, dass sich die heutige Unternehmenswelt kaum der Cloud-Technologie verschließen kann, um zeitgemäß auf die stetig wachsende Menge an Nutzern reagieren zu können. Die Anbieter müssen die Anpassung des Ressourcenvolumens immer flexibler umsetzen. So wird gerade auch im Zuge des wachsenden Verantwortungsbewusstseins bezüglich der Umwelt Cloud Computing zukünftig eine immer bedeutendere Rolle im IT-Forschungsbereich spielen. Dafür setzt die hier vorgestellte Lösung neue Maßstäbe bei der Verteilung und dynamischen Einbindung virtueller Ressourcen in physikalische Maschinen.

Abbildungsverzeichnis

2.1 Überblick Cloud-Stack 8

3.1 Architektur von AzureWatch 30

4.1 Denkbare Lastmuster 37
4.2 Geld- und Leistungsfluss IaaS-, PaaS-, SaaS-Anbieter und Nutzer 40
4.3 Ablauf anfragende Clients und Cloud-Dienste 42

5.1 Beziehung der System-Komponenten 48
5.2 Darstellung Interaktion zwischen Client und Service 49
5.3 Protokollablauf bei Überlast 51
5.4 Client-interne Aktivitäten 53
5.5 SLM-interne Aktivitäten 54
5.6 Grobgranulare Architektur der Ressourcenmanagement-Lösung . 56
5.7 Umsetzung auf der Microsoft Azure Platform 57
5.8 Ressourcenmanagement ohne Adaption 58
5.9 Ressourcenmanagement mit Adaption durch Verzögerung 60
5.10 Antwortzeiten in Zeiten hoher Last 63
5.11 Verwurfsraten in Zeiten hoher Last 63
5.12 Antwortzeiten im Smartphone-Szenario 65
5.13 Verwurfsraten im Smartphone-Szenario 66
5.14 Antwortzeiten in Sensornetzen 68
5.15 Verwurfsraten in Sensornetzen 69

6.1 Lastgebirge ohne Anfragenverzögerung 74
6.2 Lastgebirge mit Anfragenverzögerung und Skalierungsprozessen 75
6.3 Beispielhafte Dienstumgebung und gespeicherte Abstraktion . . 76
6.4 Graph der Dienstumgebung beim SLM 77
6.5 Skalierungsvorgang mittels Abhängigkeitsgraph 80
6.6 Implementierung des automatischen Skalierungsprozesses 82
6.7 Lastszenarien der Simulationen 84
6.8 Lastgebirge der simulierten Lastszenarien 85
6.9 Lastdarstellung aktiver Maschinen auf Email-Servern 88
6.10 VM-Lastgebirge beim *Auto*-Scheduling auf Email-Servern 89
6.11 Lastdarstellung aktiver Maschinen auf Ticket-Servern 91
6.12 Lastdarstellung aktiver Maschinen auf Suchmaschinen 93
6.13 Lastdarstellung aktiver Maschinen auf News-Servern 95
6.14 Lastdarstellung aktiver Maschinen im Forschungsumfeld 97

6.15 Legende aller Arbeitslastszenarien 97
6.16 Scheduling-Simulation für die Arbeitslastszenarien 1 und 2 . . . 98
6.17 Scheduling-Simulation für die Arbeitslastszenarien 3-5 99

7.1 Muster für Popularitätsverläufe 103
7.2 Popularitätskurve 104
7.3 Ablauf der Schätzung des Ressourcenbedarfs 107
7.4 Algorithmus zur Allokation der Ressourcen 112
7.5 Überblick über den Systemablauf 115
7.6 Sequenzierung durch den Tweet-Server 117
7.7 Streuung der Tweets nach fünf Stunden 118
7.8 Verhältnis Lastkurve zu Tweet-Anzahl 119
7.9 Verarbeitungs- und Übertragungszeit bei Verschiebung 125
7.10 Visuelle Darstellung der Kurvenparameter 127
7.11 Verhalten der Kostenfunktionen 133

Tabellenverzeichnis

5.1 Simulationswerte Szenario 1: Periodischer File-Download 62
5.2 Simulationswerte Szenario 2: Permanenter Daten-Download . . . 65
5.3 Simulationswerte Szenario 3: Ereignisgetriebene Nachrichtenverbreitung 67

6.1 Simulationswerte für Business-Lastmuster 87
6.2 Simulationswerte auf Ticket- oder Auktionshaus-Servern 90
6.3 Simulationswerte für Last auf Suchmaschinen- oder Batch-Job-Servern 92
6.4 Simulationswerte für Lastspitzen bei Servern für News-Seiten . . 94
6.5 Simulationswerte schwankender Arbeitslast im Forschungsumfeld 96

7.1 Beispielwerte für Tabelle $A_{\text{Kalifornien}}$ 114
7.2 Round Trip Delays zu den Standorten der Rechenzentren 120
7.3 Round Trip Delay Verhältnisse zur Region West Europa 121
7.4 Antwortzeiten der Region West und Nord Europa 122
7.5 Antwortzeiten der Regionen Ost USA und Ost Asien 123
7.6 Verbesserungsfaktoren der Antwortzeiten nach Ressourcenverschiebung 124

Literaturverzeichnis

[1] Adelsberger, H. und A. Drechsler: *Ausgewählte Aspekte des Cloud-Computing aus einer IT-Management-Perspektive: Cloud Governance, Cloud Security und Einsatz von Cloud Computing in jungen Unternehmen*. Doktorarbeit, Universität Duisburg Essen, Dezember 2010.

[2] Aggarwal, M., R. Kent und A. Ngom: *Genetic Algorithm based Scheduler for Computational Grids*. In: *Proceedings of the 19th International Symposium on High Performance Computing Systems and Applications (HPCS'05)*, S. 209 – 215, 2005.

[3] Alakeel, A.: *A Guide to Dynamic Load Balancing in Distributed Computer Systems*. International Journal of Computer Science and Information Security, 10(6):153–160, 2010.

[4] Alam, K., E. Keresteci, B. Nene und T. Swanson: *Multi-Tenant Applications on Windows Azure: Dokumentation*. `http://cloudninja.codeplex.com/releases/view/65798`, 2011. [Stand: Dezember 2013].

[5] Alexa Internet, I.: *Alexa The Web Information Company*. `http://www.alexa.com/siteinfo/flickr.com`. [Stand: Dezember 2013].

[6] Amazon: *Amazon Elastic Block Store (EBS)*. `http://aws.amazon.com/de/ebs/`. [Stand: Dezember 2013].

[7] Amazon: *Amazon Web Services*. `http://aws.amazon.com/de/`. [Stand: Dezember 2013].

[8] Amazon: *CloudWatch*. `http://aws.amazon.com/de/cloudwatch/`. [Stand: Dezember 2013].

[9] Amazon: *Elastic Compute Cloud (EC2)*. `http://aws.amazon.com/de/ec2/`. [Stand: Dezember 2013].

[10] Amazon: *Relational Database Service (RDS)*. `http://aws.amazon.com/de/rds/`. [Stand: Dezember 2013].

[11] Amazon: *Simple Storage Service (S3)*. `http://aws.amazon.com/de/s3/`. [Stand: Dezember 2013].

[12] Amazon: *SimpleDB*. `http://aws.amazon.com/de/simpledb/`. [Stand: Dezember 2013].

[13] Amazon: *CloudWatch Dokumentation*. `http://awsdocs.s3.amazonaws.com/AmazonCloudWatch/latest/acw-dg.pdf`, 2013. [Stand: Dezember 2013].

[14] Andress, H.: *Chi-Quadrat-Verteilung*. `http://eswf.uni-koeln.de/glossar/chivert.htm`, 2003. Uni Köln, Lehrstuhl für Sozial- und Wirtschaftsforschung. [Stand: Dezember 2013].

[15] Armbrust, M., A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica und M. Zaharia: *A View of Cloud Computing*. Communications of the ACM, 53(4):50–58, 2010.

[16] Armbrust, M., A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin und M. Zaharia: *Above the Clouds: A Berkeley View of Cloud Computing*, 2009. Technischer Bericht: UCB/EECS-2009-28. EECS Department, University of California at Berkley.

[17] Assunção, M., A. Di Costanzo und R. Buyya: *Evaluating the Cost-benefit of using Cloud Computing to Extend the Capacity of Clusters*. In: *Proceedings of the 18th ACM International Symposium on High Performance Distributed*

Computing (HPDC'09), S. 141–150, NY, USA, 2009.

[18] Baun, C., M. Kunze, J. Nimis und S. Tai: *Cloud Computing: Web-basierte dynamische IT-Services*. Springer Berlin / Heidelberg, 2. Aufl., Februar 2011.

[19] Berl, A., E. Gelenbe, M. Girolamo, G. Giuliani, M. Meer, H. Dang und K. Pentikousis: *Energy-Efficient Cloud Computing*. The Computer Journal, 53(7):1045–1051, 2010.

[20] Birman, K., G. Chockler und R. van Renesse: *Toward a Cloud Computing Research Agenda*. SIGACT News, 40:68–80, June 2009.

[21] Braden, R., L. Zhang, S. Berson, S. Herzog und S. Jamin: *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification*. RFC Editor, United States, 1997.

[22] Brandic, I.: *Towards Self-Manageable Cloud Services*. In: *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC '09)*, S. 128–133. IEEE Computer Society, 2009.

[23] Brewer, E.: *Towards Robust Distributed Systems*. In: *Proceedings of the 19th Annual Symposium on Principles of Distributed Computing (PODC'00)*. ACM, 2000.

[24] Brewer, E.: *CAP Twelve Years Later: How the "Rules" have changed*. Computer, 45(2):23–29, 2012.

[25] Bundesdatenschutzgesetz: BDSG: *Der Bundesbeauftragte für den Datenschutz und die Informationsfreiheit*. `http://www.bfdi.bund.de/cae/servlet/contentblob/409518/publicationFile/25234/BDSG.pdf`. [Stand: Dezember 2013].

[26] Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V.: *Deutscher ITK-Markt wächst um 1,6 Prozent*. `http://www.bitkom.org/70752_71380.aspx`, 2012. [Stand: Dezember 2013].

[27] Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V.: *Umsatz mit Cloud Computing steigt auf fast 8 Milliarden Euro*. `http://www.bitkom.org/de/markt_statistik/64086_75301.aspx`, 2013. [Stand: Dezember 2013].

[28] Buyya, R., C. Yeo, S. Venugopal, J. Broberg und I. Brandic: *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, 2009.

[29] Cachin, C., I. Keidar und A. Shraer: *Trusting the Cloud*. SIGACT News, 40(2):81–86, 2009.

[30] Cardosa, M., M. Korupolu und A. Singh: *Shares and Utilities based Power Consolidation in Virtualized Server Environments*. In: *Proceedings of the 11th International Symposium on Integrated Network Management (IM'09)*, S. 327–334. IEEE, 2009.

[31] Chaisiri, S., B. Lee und D. Niyato: *Optimization of Resource Provisioning Cost in Cloud Computing*. IEEE Transactions on Services Computing, 99(PrePrints), 2011.

[32] Chaisiri, S., B. Lee und D. Niyato: *Optimization of Resource Provisioning Cost in Cloud Computing*. IEEE Transactions on Services Computing, 5(2):164–177, 2012.

[33] Chappell, D.: *Enterprise Service Bus: Theory in Practice*. O'Reilly Media, 2004.

[34] Chen, J., W. Li, A. Lau, J. Cao und K. Wang: *Automated Load Curve Data Cleansing in Power Systems*. IEEE Transactions on Smart Grid, 1(2):213–221, 2010.

[35] Chieu, T., A. Mohindra, A. Karve und A. Segal: *Dynamic Scaling of Web*

Applications in a Virtualized Cloud Computing Environment. In: *Proceedings of the 6th International Conference on e-Business Engineering (ICEBE '09)*, S. 281–286. IEEE Computer Society, 2009.

[36] Chisnall, D.: *The Definitive Guide to the Xen Hypervisor*. Prentice Hall PTR, 2008.

[37] Chow, R., P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka und J. Molina: *Controlling Data in the Cloud: Outsourcing Computation without Outsourcing Control*. In: *Proceedings of the 16th Workshop on Cloud Computing Security (CCSW'09)*, S. 85–90, NY, USA, 2009. ACM.

[38] Conrad, M.: *Strom sparen bei PC, Drucker & Co. Tipps zum Sparen und für den Neukau*. http://www.unserener.de/downloads/UE_Stromsparen_PC+Drucker_web.pdf, 2013. Ministerium für Umwelt, Forsten und Verbraucherschutz. [Stand: Dezember 2013].

[39] Czajkowski, K., I. Foster, C. Kesselman, V. Sander und S. Tuecke: *SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems*. In: *Proceedings of the 8th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'02)*, Bd. 2537, S. 153–183, 2002.

[40] Database.com. http://www.salesforce.com/de/database/. [Stand: Dezember 2013].

[41] Database.com: *Pricing*. http://www.database.com/pricing, 2011. [Stand: Dezember 2013].

[42] Deutinger, K.: *Modulare Mechanismen zur Lastbalancierung in Peer-to-Peer Systemen*. Diplomarbeit, Otto-von-Guericke-Universität Magdeburg - Fakultät für Informatik Institut für Technische und Betriebliche Informationssysteme, Deutschland, 2005.

[43] Diestel, R.: *Graphentheorie*. Springer Berlin / Heidelberg, 3 Aufl., 2005.

[44] Dirscherl, H.: *Online-Office: Mail, Kalender & Office von Google - Google Apps*. http://www.pcwelt.de/ratgeber/Online-Office-Mail-Kalender-Office-von-Google-Google-Apps-325400.html, 2010. [Stand: Dezember 2013].

[45] Dong, F. und S. G. Akl: *Scheduling Algorithms for Grid Computing: State of the Art and Open Problems*. Techn. Ber. No. 2006-504, School of Computing, Queen's University, Kingston, Ontario, 2006.

[46] Dornemann, T., E. Juhnke und B. Freisleben: *On-demand Resource Provisioning for BPEL Workflows using Amazon's Elastic Compute Cloud*. In: *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'09)*, S. 140–147. IEEE, 2009.

[47] Ecma International: *The JSON Data Interchange Format*. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf, 2013. Strandard ECMA-404, 1st Edition. [Stand: Dezember 2013].

[48] Ertl, A.: *How much Electricity does a Computer Consume?*. http://www.complang.tuwien.ac.at/anton/computer-power-consumption.html. [Stand: Dezember 2013].

[49] Eucalyptus Systems, Inc.: *Open Source AWS Compatible Private Clouds*. http://www.eucalyptus.com/. [Stand: Dezember 2013].

[50] Experton Group: *Cloud Marktzahlen*. http://www.experton-group.de/press/releases/pressrelease/article/5-huerde-erreicht-aktuelle-marktzahlen-zum-deutschen-cloud-computing-markt.html, 2012. [Stand: Dezember 2013].

[51] Facebook. http://www.facebook.com. [Stand: Dezember 2013].

[52] Feng, W., X. Feng und R. Ge: *Green Supercomputing Comes of Age*. IT

Professional, 10(1):17–23, 2008.

[53] Ferrari, D., A. Gupta und G. Ventre: *Distributed Advance Reservation of Real-Time Connections*. In: *Network and Operating Systems Support for Digital Audio and Video*, Bd. 1018 d. Reihe *Lecture Notes in Computer Science*, S. 16–27. Springer, 1995.

[54] Ferrer, A. J., F. Hernández, J. Tordsson, et al.: *OPTIMIS: A Holistic Approach to Cloud Service Provisioning*. Future Generation Computer Systems, 28:66–77, 2012.

[55] Fershtman, C. und N. Gandal: *Migration to the Cloud Ecosystem: Ushering in a New Generation of Platform Competition*, 2012. Centre for Economic Policy Research, CEPR Discussion Paper No. 8907.

[56] Fielding, R.: *Architectural Styles and the Design of Network-Based Software Architectures*. Doktorarbeit, University of California, 2000.

[57] Fito, J., I. Goiri und J. Guitart: *SLA-Driven Elastic Cloud Hosting Provider*. Euromicro Conference on Parallel, Distributed, and Network-Based Processing, 0:111–118, 2010.

[58] Forrest, W.: *How to cut Data Centre Carbon Emissions?*. `http://www.computerweekly.com/Articles/2008/12/05/233748/how-to-cut-data-centre-carbon-emissions.htm`, 2008. Computerweekly. [Stand: Dezember 2013].

[59] Foster, I.: *There's Grid in them thar Clouds*. `http://ianfoster.typepad.com/blog/2008/01/theres-grid-in.html`, 2008. [Stand: Dezember 2013].

[60] Foster, I., Y. Zhao, I. Raicu und S. Lu: *Cloud Computing and Grid Computing 360-Degree Compared*. In: *Proceedings of the Grid Computing Environments Workshop (GCE'08)*, S. 1–10, 2008.

[61] Foursquare. `https://de.foursquare.com/`. [Stand: Dezember 2013].

[62] Fritsch, W.: *Ressourcen für Softwareentwicklung und Datenhaltung als Service: Salesforce.com erweitert Cloud-Angebote*. `http://www.crn.de/software/artikel-87705.html`, 2010. [Stand: Dezember 2013].

[63] Galstyan, A., K. Czajkowski und K. Lerman: *Resource Allocation in the Grid with Learning Agents*. Journal of Grid Computing, 3:91–100, 2005.

[64] Gandham, S., M. Dawande, R. Prakash und S. Venkatesan: *Energy Efficient Schemes for Wireless Sensor Networks with Multiple Mobile base Stations*. In: *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM'03)*, S. 377–381, 2003.

[65] Geobytes, Inc. `http://www.geobytes.com/iplocator.htm`, 2006. [Stand: Dezember 2013].

[66] Gilbert, S. und N. Lynch: *Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services*. SIGACT News, 33(2), 2002.

[67] Gmach, D., S. Krompass, S. Seltzsam, M. Wimmer und A. Kemper: *Dynamic Load Balancing of Virtualized Database Services Using Hints and Load Forecasting*. In: *Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE Workshops'05)*, Bd. 2, S. 23–37, 2005.

[68] Goel, A., D. Steere, C. Pu und J. Walpole: *SWiFT: A Feedback Control and Dynamic Reconfiguration Toolkit*. In: *Proceedings of the 2nd Conference on USENIX Windows NT Symposium (WINSYM'98)*, Bd. 2, Berkeley, CA, USA, 1998. USENIX Association.

[69] Goel, A., D. Steere, C. Pu und J. Walpole: *Adaptive Resource Management Via Modular Feedback Control*. Techn. Ber. No. 99-03, Oregon Graduate Institute, Computer Science and Engineering, 1999.

[70] Gong, Z., P. Ramaswamy, X. Gu und X. Ma: *SigLM: Signature-Driven Load Management for Cloud Computing Infrastructures*. In: *Proceedings of the*

17th International Workshop on Quality of Service (IWQoS'09), S. 1–9. IEEE, 2009.

[71] Google: *App Engine*. https://developers.google.com/appengine/. [Stand: Dezember 2013].

[72] Google: *Blobstore*. https://developers.google.com/appengine/docs/java/blobstore/?hl=de. [Stand: Dezember 2013].

[73] Google: *Cloud Platform*. https://cloud.google.com/products/. [Stand: Dezember 2013].

[74] Google: *Google Apps for Business*. http://www.google.de/intx/de/enterprise/apps/business/. [Stand: Dezember 2013].

[75] Gruman, G. und E. Knorr: *What Cloud Computing Really Means.*. InfoWorld, Electronic Magazine, April 2008.

[76] Haluschak, B.: *Ratgeber Netzteil – Netzteile und elektrische Leistung*. http://www.pcwelt.de/ratgeber/Netzteile-und-elektrische-Leistung-Ratgeber-Netzteil-314388.html, 2009. PC Welt Ratgeber. [Stand: Dezember 2013].

[77] Hartl, F., G. Koleva und M. Steinkamp: *Google App Engine: Eine genaue Betrachtung der GAE*. Doktorarbeit, Technische Universität München, Deutschland, 2010.

[78] Heiser, J. und M. Nicolett: *Assessing the Security Risks of Cloud Computing*. http://www.globalcloudbusiness.com/SharedFiles/Download.aspx?pageid=138&mid=220&fileid=12, 2008. Gartner Report, ID Number: G00157782, [Stand: Dezember 2013].

[79] Herrmann, K. und K. Geihs: *Integrating Mobile Agents and Neural Networks for Proactive Management*. In: *IFIP TC6 / WG6.1 3rd International Working Conference on New Developments in Distributed Applications and Interoperable Systems*, S. 203–216. Kluwer Academic Publishers, 2001.

[80] Hintemann, R. und K. Fichter: *Energieverbrauch und Energiekosten von Servern und Rechenzentren in Deutschland – Aktuelle Trends und Einsparpotenziale bis 2015*. http://www.bitkom.org/files/documents/Kurzstudie_Borderstep_l_Rechenzentren.pdf, 2012. Borderstep Institut für Innovation und Nachhaltigkeit gemeinnützige GmbH, Berlin. [Stand: Dezember 2013].

[81] Hohenstein, U., R. Krummenacher, L. Mittermeier und S. Dippl: *Choosing the Right Cloud Architecture - A Cost Perspective*. In: *Proceedings of the 2nd International Conference on Cloud Computing and Services Science (CLOSER'12)*, S. 334–344. SciTePress, 2012.

[82] Horn, P.: *Autonomic Computing: IBM's Perspective on the State of Information Technology*, 2001. IBM.

[83] Ihlenfeld, J.: *App Engine 1.5.0: Google erweitert seinen Cloud-Computing-Dienst*. http://www.golem.de/1105/83388.html, 2011. [Stand: Dezember 2013].

[84] Iqbal, W., M. Dailey und D. Carrera: *SLA-Driven Adaptive Resource Management for Web Applications on a Heterogeneous Compute Cloud*. In: *Cloud Computing*, S. 243–253. Springer, 2009.

[85] Jarry, A., P. Leone, O. Powell und J. Rolim: *An Optimal Data Propagation Algorithm for Maximizing the Lifespan of Sensor Networks*. In: *Proceedings of the 2nd IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS'06)*, S. 405–421. Springer Berlin / Heidelberg, 2006.

[86] Jeffery, K. und B. Neidecker-Lutz: *The Future of Cloud Computing: Opportunities for European Cloud Computing Beyond 2010*. http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-report-final.pdf, May 2011. Expert Group Report, Public Version 1.0, [Stand: Dezember 2013].

[87] Jensen, M., J. Schwenk, N. Gruschka und L. Iacono: *On Technical Security Issues in Cloud Computing*. In: *Proceedings of the 2009 IEEE International Conference on Cloud Computing (CLOUD'09)*, S. 109–116, Washington, DC, USA, 2009. IEEE Computer Society.

[88] Jung, J.: *SQL Azure wird ausgereifter*. `http://www.webundmobile.de/Aktuelles/News/SQL-Azure-wird-ausgereifter-4786.html`, 2011. [Stand: Dezember 2013].

[89] Kawamoto, K., J. Koomey, B. Nordman, R. Brown, M. Piette, M. Ting und A. Meier: *Electricity Used by Office Equipment and Network Equipment in the US*. Energy, 27(3):255–269, 2002.

[90] Kemper, A. und A. Eickler: *Datenbanksysteme: Eine Einführung*. Oldenbourg, München, 4 Aufl., 2001.

[91] Kiencke, U.: *Ereignisdiskrete Systeme : Modellierung und Steuerung verteilter Systeme*. Oldenbourg Verlag, München, 2. Aufl., 2006.

[92] Kowalski, R., S. Limber und P. Agatston: *Cyberbullying: Bullying in the Digital Age*. John Wiley & Sons, 2 Aufl., 2012.

[93] Krejcar, O.: *Problem Solving of Low Data Throughput on Mobile Devices by Artefacts Prebuffering*. EURASIP Journal on Wireless Communications and Networking, 2009.

[94] Kwak, H., C. Lee, H. Park und S. Moon: *What is Twitter, a Social Network or a News Media?*. In: *Proceedings of the 19th International Conference on World Wide Web (WWW'10)*, S. 591–600, New York, NY, USA, 2010. ACM.

[95] Kwok, Y. und I. Ahmad: *Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors*. ACM Computing Surveys, 31(4):406–471, Dez. 1999.

[96] Leskovec, J.: *Rhythms of Information Flow through Networks*. `http://sonic.northwestern.edu/wp-content/uploads/2011/03/3-Leskovec-info_flow.pdf`, 2011. [Stand: Dezember 2013].

[97] Li, B. und K. Nahrstedt: *QualProbes: Middleware QoS Profiling Services for Configuring Adaptive Applications*. In: *Proceedings of the IFIP/ACM International Conference on Distributed systems platforms (Middleware'00)*, Bd. 1795, S. 256–272. Springer Berlin / Heidelberg, 2000.

[98] Lighthill, M.: *Introduction to Fourier Analysis and Generalised Functions*. Cambridge University Press, 1996.

[99] Lim, H., S. Babu, J. Chase und S. Parekh: *Automated Control in Cloud Computing: Challenges and Opportunities*. In: *Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds (ACDC'09)*, S. 13–18. ACM, 2009.

[100] Little, J.: *A Proof for the Queuing Formula: $L=\lambda W$*. Operations Research, 9(3):383–387, 1961.

[101] Lucas, J., C. Carrión und B. Caminero: *Flexible Advance-reservation (FAR) for Clouds*. In: Leymann, F., I. Ivanov, M. van Sinderen und B. Shishkov (Hrsg.): *Proceedings of the 1st International Conference on Cloud Computing and Services Science (CLOSER'11)*, S. 610–615. SciTePress, 2011.

[102] Mahlmann, P. und P. Schindelhauer: *Peer-to-Peer-Netzwerke in der Praxis*. eXamen.press. Springer Berlin / Heidelberg, 2007.

[103] Malawski, M. a., G. Juve, E. Deelman und J. Nabrzyski: *Cost-and Deadline-Constrained Provisioning for Scientific Workflow Ensembles in IaaS Clouds*. In: *Proceedings of the 25th International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*, S. 1–11. IEEE Computer Society Press, 2012.

[104] Mao, M. und M. Humphrey: *Auto-Scaling to Minimize Cost and Meet Appli-*

cation Deadlines in Cloud Workflows. In: *Proceedings of the 24th International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*, S. 1–12. IEEE, 2011.

[105] Mao, M. und M. Humphrey: *A Performance Study on the VM Startup Time in the Cloud*. In: *Proceedings of the 5th International Conference on Cloud Computing (CLOUD'12)*, S. 423–430. IEEE, 2012.

[106] Mao, M., J. Li und M. Humphrey: *Cloud Auto-Scaling with Deadline and Budget Constraints*. In: *Proceedings of the 11th IEEE/ACM International Conference on Grid Computing (GRID'10)*, S. 41–48. IEEE, 2010.

[107] Marinos, A. und G. Briscoe: *Community Cloud Computing*, Bd. 5931 d. Reihe *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2009.

[108] McKinsey & Company: *Clearing the Air on Cloud Computing*. http://www.isaca.org/Groups/Professional-English/cloud-computing/GroupDocuments/McKinsey_Cloud%20matters.pdf, 2009. [Stand: Dezember 2013].

[109] Meddeb, A.: *Internet QoS: Pieces of the Puzzle*. IEEE Communications Magazine, 48(1):86–94, 2010.

[110] Mell, P. und T. Grance: *The NIST Defiition of Cloud Computing (Draft)*. http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145_cloud-definition.pdf, 2011. [Stand: Dezember 2013].

[111] Meyer, D.: *Stromfresser PC: Runter mit den Energiekosten*. http://www.t-online.de/computer/hardware/id_53106252/stromfresser-pc-runter-mit-den-kosten.html, 2013. T-Online Computer. [Stand: Dezember 2013].

[112] Michel, D.: *Databases in the Cloud*. Doktorarbeit, HSR University of Applied Science Rapperswil, Rapperswil, 6. Dezember 2010.

[113] Microsoft: *Einführung in Windows Server AppFabric*. http://msdn.microsoft.com/de-de/library/ee677312%28v=azure.10%29.aspx. [Stand: Dezember 2013].

[114] Microsoft: *How to use the Queue Storage Service*. http://www.windowsazure.com/en-us/develop/net/how-to-guides/queue-service/. [Stand: Dezember 2013].

[115] Microsoft: *How to use the Table Storage Service*. http://www.windowsazure.com/en-us/develop/net/how-to-guides/table-services/. [Stand: Dezember 2013].

[116] Microsoft: *How to use Windows Azure SQL Database in .NET Applications*. http://www.windowsazure.com/en-us/develop/net/how-to-guides/sql-database/?fb=de-de. [Stand: Dezember 2013].

[117] Microsoft: *Introducing Windows Azure*. http://www.windowsazure.com/en-us/develop/net/fundamentals/intro-to-windows-azure/?fb=de-de. [Stand: Dezember 2013].

[118] Microsoft: *Windows Azure*. http://msdn.microsoft.com/en-us/library/windowsazure/dd163896.aspx. [Stand: Dezember 2013].

[119] Microsoft: *Windows Azure*. https://www.windowsazure.com. [Stand: Dezember 2013].

[120] Microsoft: *Windows Azure Compute*. http://www.windowsazure.com/en-us/develop/net/compute/?fb=de-de. [Stand: Dezember 2013].

[121] Microsoft: *Windows Azure Platform*. http://www.microsoft.com/windowsazure/offers/popup/popup.aspx?lang=en&locale=en-us&offer=MS-AZR-0003P. [Stand: Dezember 2013].

[122] Microsoft Windows Azure: *Virtual Machine and Cloud Service Sizes for Windows Azure*. http://msdn.microsoft.com/en-us/library/windowsazur

e/dn197896.aspx. [Stand: Dezember 2013].

[123] Microsoft Windows Azure: *Cloud Services – Hosted Services*. http://msdn.microsoft.com/en-us/library/windowsazure/jj155995.aspx, 2012. [Stand: Dezember 2013].

[124] Microsystems, S.: *JXTA v2.0 Protocols Specification*. https://jxta.kenai.com/, 2004. [Stand: Dezember 2013].

[125] Miers, C., M. Barros, M. Simplicio, N. Gonzalez, P. Evangelista, W. Goya, et al.: *Using Trade Wind to Sail in the Clouds*. In: *Proceedings of the 23rd Conference on Parallel and Distributed Computing and Systems (PDCS'11)*. ACTA Press, 2011.

[126] Misra, S. und A. Mondal: *Identification of a Company's Suitability for the Adoption of Cloud Computing and Modelling its Corresponding Return on Investment*. Mathematical and Computer Modelling, 53(3):504–521, 2011.

[127] Nurmi, D., R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff und D. Zagorodnov: *Eucalyptus: A Technical Report on an Elastic Utility Computing Architecture Linking your Programs to Useful Systems*. In: *UCSB Technical Report*. Citeseer, 2008.

[128] Nygren, E., R. Sitaraman und J. Sun: *The Akamai Network: A Platform for High-Performance Internet Applications*. ACM SIGOPS Operating Systems Review, 44(3):2–19, 2010.

[129] Oracle: *Oracle Exalogic Elastic Cloud X2-2*. http://www.oracle.com/oms/hardware/extremeperformance/assets/ept-ds-exalogic-elastic-cloud-1641451.pdf, 2012. [Stand: Dezember 2013].

[130] Oracle: *Data Warehouse Appliance: Oracle Exadata*. http://www.oracle.com/us/products/database/datawarehousing/overview/index.html, 2013. [Stand: Dezember 2013].

[131] Padala, P., K. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant und K. Salem: *Adaptive Control of Virtualized Resources in Utility Computing Environments*. ACM SIGOPS Operating Systems Review, 41(3):289–302, 2007.

[132] Paraleap Technologies: *Architecture Overview AzureWatch*. http://blogs.msdn.com/b/ukisvdev/archive/2011/06/14/autoscaling-a-windows-azure-hosted-service-with-azurewatch.aspx. [Stand: Dezember 2013].

[133] Paraleap Technologies: *AzureWatch*. http://www.paraleap.com/AzureWatch. [Stand: Dezember 2013].

[134] Paraleap Technologies: *AzureWatch Security*. http://www.paraleap.com/AzureWatch/Security. [Stand: Dezember 2013].

[135] Paton, N. W., M. De Aragão, K. Lee, A. Fern und R. Sakellariou: *Optimizing Utility in Cloud Computing Through Autonomic Workload Execution*. IEEE Bulletin of the Technical Committee on Data Engineering, 32(1):51–58, 2009.

[136] Pearson, K.: *On the Criterion that a Given System of Derivations from the Probable in the Case of a Correlated System of Variables is such that it can be Reasonably Supposed to have arisen from Random Sampling*. Philosophical Magazine and Journal of Science, 50(5):157–175, 1900.

[137] Pearson, S. und A. Benameur: *Privacy, Security and Trust Issues Arising from Cloud Computing*. In: *Proceedings of the IEEE 2nd International Conference on Cloud Computing Technology and Science (CloudCom'10)*, S. 693–702, 2010.

[138] Petersohn, N.: *Vergleich und Evalution zwischen modernen und traditionellen Datenbankkonzepten unter den Gesichtspunkten Skalierung, Abfragemög-*

lichkeit und Konsistenz. Diplomica Verlag, 2011.

[139] Phillips, D. und P. Young: *Online Public Relations: A Practical Guide to Developing an Online Strategy in the World of Social Media*. PR In Practice. Kogan Page, 2009.

[140] Pring, E., R. Brown, L. Leong, A. Couture, F. Biscotti, B. Lheureux, A. Frank, J. Roster, S. Cournoyer und V. Liu: *Garntner Forecast: Public Cloud Services, Worldwide and Regions, Industry Sectors, 2009-2014*. `http://www.gartner.com/DisplayDocument?ref=clientFriendlyUrl&id=1378513`, 2010. [Stand: Dezember 2013].

[141] Ragan, K.: *The Cloud Wars: $100+ Billion at Stake*, 2008. Technischer Bericht, Merrill Lynch.

[142] Raman, R., M. Livny und M. Solomon: *Matchmaking: Distributed Resource Management for High Throughput Computing*. In: *Proceedings of the 7th International Symposium on High Performance Distributed Computing (HPDC'98)*, S. 140–146, 1998.

[143] Randles, M., D. Lamb und A. Taleb-Bendiab: *A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing*. In: *Proceedings of the 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA'10)*, S. 551–556. IEEE, 2010.

[144] Rescorla, E.: *SSL and TLS: Designing and Building Secure Systems*. Addison-Wesley, New York, USA, 2001.

[145] Rinne, H.: *Taschenbuch der Statistik*. Verlag Harri Deutsch, 3. Aufl., 2003.

[146] Roy, N., A. Dubey und A. Gokhale: *Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting*. In: *Proceedings of the 4th IEEE International Conference on Cloud Computing (CLOUD'11)*, S. 500–507, 2011.

[147] Salesforce.com: *Gute Anwendungen muss man nicht besitzen, um sie zu benutzen*. `http://www.salesforce.com/de/company/`. [Stand: Dezember 2013].

[148] Schiller, J.: *Mobile Communications*. Pearson Education, 2 Aufl., 2003.

[149] Schmidt, W.: *Elektrische Leistung, Arbeit, Stromkostenberechnung*. `http://www1.schmidts-phytech.de/media/physik/elektronik/leistung-arbeit-kosten.pdf`, 2008. [Stand: Dezember 2013].

[150] Schumacher, R.: *Improving Database Performance with Partitioning*. `http://dev.mysql.com/tech-resources/articles/partitioning.html`, 2010. [Stand: Dezember 2013].

[151] Schüttler, M. und A. Essigkrug: *Softwareentwicklung in den Wolken*. Die Zeitschrift für Software-Engineering und -Management, S. 34–36, 2012.

[152] Schwanengel, A.: *Cloud Datenspeicher- und Datenbank-Lösungen – Eine Übersicht zu den Big Playern des Cloud Computing*. Praxis der Informationsverarbeitung und Kommunikation (PIK), 36(2):133–140, May 2013.

[153] Schwanengel, A. und U. Hohenstein: *Challenges with Tenant-Specific Cost Determination in Multi-Tenant Applications*. In: *Proceedings of the 4th International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING'13)*, 2013.

[154] Schwanengel, A., U. Hohenstein und M. C. Jaeger: *Resource Allocation for Cloud SaaS Offerings based on Social Web Applications*. In: *Proceedings of the IEEE International Conference on Cloud Networking (CloudNet'12)*, S. 179–181, 2012.

[155] Schwanengel, A., M. C. Jaeger und U. Hohenstein: *Automated Load Adaptation for Cloud Environments in Regard of Cost Models*. In: *Proceedings of the 2nd International Conference on Cloud Computing and Services Science (CLOSER'12)*, S. 562–567. SciTePress, 2012.

[156] Schwanengel, A. und G. Kaefer: *Light-Weight Load Management Protocol based on Reservation and Feedback Loops*. In: *Proceedings of the 23rd International Conference on Parallel and Distributed Computing and Systems (PDCS'11)*. ActaPress, 2011.

[157] Schwanengel, A., G. Kaefer und C. Linnhoff-Popien: *Improved Throughput and Response Times by a Light-Weight Load Management Protocol*. Journal of Parallel and Cloud Computing (PCC'12), 1(1):1–9, 2012.

[158] Schwanengel, A., G. Kaefer und C. Linnhoff-Popien: *Proactive Automated Dependable Resource Management in Cloud Environments*. In: *Proceedings of the 7th International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP'13)*, 2013.

[159] Schwanengel, A., A. Schönl und C. Linnhoff-Popien: *Location-based Cloud Resource Allocation based on Information of the Social Web*. International Journal of Advanced Cloud Computing and Applied Research, 1:1–12, 2013.

[160] Seltzsam, S., D. Gmach, S. Krompass und A. Kemper: *AutoGlobe: An Automatic Administration Concept for Service-Oriented Database Applications*. In: *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, 2006.

[161] Sharp, R.: *Principles of Protocol Design*. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008.

[162] Siegele, L.: *Let It Rise: A Special Report on Corporate IT*. The Economist, 389:3–14, 2008.

[163] Skype. `http://www.skype.com/intl/en/home`. [Stand: Dezember 2013].

[164] Sriram, I. und A. Khajeh-Hosseini: *Research Agenda in Cloud Technologies: Technical Aspects of Cloud Computing*. CoRR, abs/1001.3259, 2010.

[165] Stackoverflow: *Why does Azure Deployment take so long?*. `http://stackoverflow.com/questions/5080445/why-does-azure-deployment-take-so-long`, 2011. [Stand: Dezember 2013].

[166] Stahlknecht, P. und U. Hasenkamp: *Einführung in die Wirtschaftsinformatik*. Springer-Lehrbuch Series. Springer Berlin / Heidelberg, 2005.

[167] Stang, K.: *Projektmanagement, Anforderungsanalyse und externe Qualitätsicherung: IT-Projekte durch umfassendes Anforderungsmanagement erfolgreich gestalten*. vdf Hochschulverlag AG an der ETH Zürich, 2002.

[168] Steere, D., A. Goel, J. Gruenberg, D. McNamee, C. Pu und J. Walpole: *A Feedback-Driven Proportion Allocator for Real-Rate Scheduling*. In: *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI'99)*, S. 145–158. USENIX Association, 1999.

[169] Takahashi, T., Y. Kadobayashi und H. Fujiwara: *Ontological Approach toward Cybersecurity in Cloud Computing*. In: *Proceedings of the 3rd International Conference on Security of Information and Networks (SIN'10)*, S. 100–109. ACM, 2010.

[170] Tanenbaum, A.: *Computernetzwerke*. Peearson Studium, 4. Aufl., 2003.

[171] Tseng, C., G. Lee, R. Liu und T. Wang: *HMRSVP: A Hierarchical Mobile RSVP Protocol*. Wireless Networks, 9:95–102, 2003.

[172] Twitter. `https://twitter.com/`. [Stand: Dezember 2013].

[173] Urgaonkar, B., P. Shenoy, A. Chandra, P. Goyal und T. Wood: *Agile Dynamic Provisioning of Multi-Tier Internet Applications*. ACM Transactions on Autonomous and Adaptive Systems (TAAS'08), 3(1):1, 2008.

[174] Vaquero, L. M., L. Rodero-Merino, J. Caceres und M. Lindner: *A Break in the Clouds: Towards a Cloud Definition*. Computer Communication Review, 39(1):50–55, 2008.

[175] Voss, W.: *Taschenbuch der Statistik*. Fachbuchverlag Leipzig, 2. Aufl., 2000.
[176] Vouk, M.: *Cloud Computing – Issues, Research and Implementations*. Journal of Computing and Information Technology, 16(4):235–246, 2004.
[177] Wartala, R.: *Hadoop - Zuverlässige, verteilte und skalierbare Big-Data-Anwendungen*. Open Source Press, München, Deutschland, 2012.
[178] Wei, K., D. Wen-wu und W. Lin: *Research on Emergency Information Management based on the Social Network Analysis*. In: *Proceedings of the International Conference on Management Science and Engineering (ICMSE'11)*, S. 1302–1310, 2011.
[179] Weinmann, J.: *Mathematical Proof of the Inevitability of Cloud Computing*. `http://www.joeweinman.com/resources/Joe_Weinman_Inevitability_Of_Cloud.pdf`, 2011. [Stand: Dezember 2013].
[180] Wilder, B.: *Cloud Architecture Patterns: Using Microsoft Azure*. Sebastopol: O'Reilly Media Inc., 2012.
[181] Wu, Z., X. Liu, Z. Ni, D. Yuan und Y. Yang: *A Market-Oriented Hierarchical Scheduling Strategy in Cloud Workflow Systems*. The Journal of Supercomputing, 63(1):256–293, 2013.
[182] Yang, B. B. und H. Garcia-Molina: *Designing a Super-Peer Network*. In: *Proceedings of the 19th International Conference on Data Engineering (ICDE'03)*, S. 49–60, 2003.
[183] Yang, J. und J. Leskovec: *Modeling Information Diffusion in Implicit Networks*. In: *Proceedings of the 10th IEEE International Conference on Data Mining (ICDM'10)*, S. 599–608. IEEE, 2010.
[184] Yang, J. und J. Leskovec: *Patterns of Temporal Variation in Online Media*. In: *Proceedings of the 4th ACM International Conference on Web Search and Data Mining (WSDM'11)*, S. 177–186. ACM, 2011.
[185] Yang, J. und J. Leskovec: *Defining and Evaluating Network Communities based on Ground-Truth*. In: *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics (MDS'12)*. ACM, 2012.
[186] Yazir, Y., C. Matthews, R. Farahbod, S. Neville, A. Guitouni, S. Ganti und Y. Coady: *Dynamic Resource Allocation in Computing Clouds using Distributed Multiple Criteria Decision Analysis*. In: *Proceedings of the 3rd International IEEE Conference on Cloud Computing (CLOUD'10)*, S. 91–98. IEEE, 2010.
[187] Ye, N.: *QoS-Centric Stateful Resource Management in Information Systems*. Information Systems Frontiers, 4:149–160, 2002.
[188] Ye, N., E. Gel, X. Li, T. Farley und Y. Lai: *Web Server QoS Models: Applying Scheduling Rules from Production Planning*. Computers and Operations Research, 32(5):1147–1164, 2005.
[189] Younge, A., G. von Laszewski, L. Wang, S. Lopez-Alarcon und W. Carithers: *Efficient Resource Management for Cloud Computing Environments*. In: *Proceedings of the International Green Computing Conference*, S. 357–364. IEEE, 2010.
[190] Zhang, L., S. Deering und D. Estrin: *RSVP: A New Resource ReSerVation Protocol*. IEEE Network, 7(5), 1993.
[191] Zhang, Q., L. Cheng und R. Boutaba: *Cloud Computing: State-of-the-Art and Research Challenges*. Journal of Internet Services and Applications, 1:7–18, 2010.
[192] Zhao, Y., . Raicu und I. Foster: *Scientific Workflow Systems for 21st Century, New Bottle or New Wine?*. In: *IEEE Congress on Services-Part I*, S. 467–471. IEEE, 2008.
[193] Zheng, H. und J. Nieh: *SWAP: A Scheduler with Automatic Process De-*

pendency Detection. In: *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation (NSDI'04)*, Bd. 1, S. 145–158. USENIX Association, 2004.