Evaluation von Elastizitätsstrategien in der Cloud in Hinblick auf optimale Ressourcennutzung am Beispiel von OpenStack

Bachelorarbeit

Erik Koppenhagen
geb. 08.06.1989 in Rostock

27. September 2013

Christian-Albrechts-Universität zu Kiel
Institut für Informatik
Arbeitsgruppe Software Engineering

Betreut durch: Prof. Dr. Wilhelm Hasselbring
M.Sc. Florian Fittkau

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel,

Zusammenfassung

Die Bachelorarbeit „Evaluation von Elastizitätsstrategien in der Cloud in Hinblick auf optimale Ressourcennutzung am Beispiel von OpenStack“ erforscht und bewertet Skalierungsstrategien für die Lastverteilung in der Cloud. Die Grundlage dafür lieferte das Bachelorprojekt „Kieker in the Cloud“ an der Christian-Albrechts-Universität zu Kiel. Es soll geklärt werden, welche Algorithmen kompatibel mit Kieker sind und die vorhanden Ressourcen optimal ausnutzen.

Um dieses Ziel zu erreichen, werden zunächst in einer theoretischen Evaluation die Strategien analysiert und verglichen. Als Entscheidungskriterien dienen die potentielle Datenverarbeitung und die entstehenden Mehrkosten in Abhängigkeit der CPU Auslastung. Die drei Elastizitätsstrategien, die am besten abschneiden, werden implementiert und in einer OpenStack Cloud getestet.

Die Ergebnisse dieser Evaluationen zeigen, dass Strategien mit Reserveknoten und Berücksichtigung des Abrechnungszeitraumes am besten abschneiden. Es hat sich herausgestellt, dass es keine „beste“ Strategie gibt, sondern nur Strategien, die für einen bestimmten Auslastungsverlauf am besten geeignet sind. Es muss folglich situationsabhängig eine Strategie gewählt werden. Hinweise, welche dies im einzelnen ist, finden sich im Fazit wieder.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1. Motivation

Bis Mitte des 20. Jahrhunderts standen viele Online-Händler und Firmen mit ähnlicher Hardware Architektur vor dem Problem, dass die vorhandenen Ressourcen nicht für Lastspitzen (zum Beispiel zur Weihnachtszeit) ausgelegt waren. Speziell für diese Zeit musste zusätzliche Hardware beschafft werden, die den Rest des Jahres ungenutzt blieb. Die logische Konsequenz daraus war, die zusätzliche Rechenleistung in der ungenutzten Zeit zu vermarkten.

Größere Firmen wie Amazon oder Google entwickelten daraus ein Geschäftsmodell: Die Rechenleistung wird gegen Entgelt anderen zur Verfügung gestellt, ohne dass die Rechner ihren Standort verlassen müssen. Die Kunden können Applikationen oder virtuelle Betriebssysteme nutzen und dynamisch an ihren Bedarf anpassen, ohne dauerhaft neue Hardware erstehen zu müssen. Durch diese Elastizität entsteht der Eindruck beim Kunden, dass die Ressourcen endlos sind [Hof 2006].

Das Kieker Framework wurde unter Prof. Dr. Wilhelm Hasselbring entwickelt. Es beobachtet und dokumentiert die Leistung von Applikationen und analysiert dynamisch das Verhalten zur Laufzeit. Des Weiteren gibt Kieker Aufschluss über die Architektur von Software Systemen und kann diese grafisch darstellen [Rohr u. a. 2008; van Hoorn u. a. 2009; 2012].

Im Rahmen eines Projektmoduls an der Christian-Albrecht-Universität zu Kiel wird Kieker in die Cloud portiert. Die Monitoring- und die Analysekomponente laufen hierbei getrennt auf mehreren Knoten. Je nach Auslastung der Analyse-Instanzen soll eine Skalierung stattfinden. Dies übernimmt ein Load Balancer, der auch für die Zuweisung der Analyseknoten an die Monitoringkomponente zuständig ist.

In der Bachelorarbeit gilt es zu evaluieren, welche Strategien für die Skalierung möglichst effizient Leistung und Wirtschaftlichkeit vereinen.

1.2. Ziele

In diesem Abschnitt werden die Ziele der Bachelorarbeit im Einzelnen vorgestellt und erläutert. Dabei wird als erstes auf die theoretische Evaluation eingegangen, aus dessen Ergebnis sich die praktische Evaluation ergibt.

1.2.1. Ziel 1: Evaluation von Elastizitätsstrategien auf theoretischer Basis in Hinblick auf das Bachelorprojekt

Anhand von verschiedenen Evaluationskriterien werden wir Algorithmen theoretisch gegenüberstellen, damit eine Vorauswahl getroffen werden kann. Die Algorithmen sollen dabei speziell für die Anforderungen des Bachelorprojekts ausgelegt sein.

Algorithmen für die Elastizität einer Cloud

Die anzuwendenden Elastizitätsstrategien bieten beste Leistung, höchste Wirtschaftlichkeit oder eine Kombination aus beidem.

Die leistungsorientierten Algorithmen sind darauf ausgelegt, immer die volle Rechenleistung der Cloud auszunutzen. Sie nehmen keine Rücksicht auf Kosten oder ähnliche Faktoren, sondern beobachten die Auslastung der verschiedenen Knoten. Kriterien wie Datenfluss, Menge der Daten in der Queue oder CPU Auslastung werden hier berücksichtigt.

Die wirtschaftlichen Algorithmen sind darauf ausgelegt, die Kosten für das Betreiben der Anwendungen in der Cloud möglichst gering zu halten. Sie nehmen weniger Rücksicht auf die Auslastung verschiedener Knoten, sondern beachten Abrechnungszeiträume und ähnliches.

Eine Kombination aus beiden bringt Algorithmen hervor, die zwar kosteneffizient operieren, dabei allerdings die Anwendungen in der Cloud so wenig wie möglich ausbremsen. Das Ziel der Bachelorarbeit ist es, einen effizienten Algorithmus dieser Art zu finden, da beide Faktoren eine wichtige Rolle für die praktische Anwendbarkeit darstellen.

1.2.2. Ziel 2: Evaluation der besten Strategien aus vorheriger Evaluation

Die drei Algorithmen, die in der theoretischen Evaluation am besten abgeschnitten haben, werden anschließend implementiert und in einer OpenStack Cloud getestet. Gegebenenfalls müssen sie verbessert oder ausgetauscht werden, wenn sich ihre Leistung in der Praxis als unzureichend erweist.

1.3. Aufbau

Zunächst werden wir im nächsten Kapitel die Grundlagen und Technologien dieser Bachelorarbeit vorstellen und erläutern, um den Einstieg in die Arbeit zu erleichtern und einen kurzen Überblick über die Thematik zu geben.

Danach wird in Kapitel 3 das Projektmodul „Kieker in the Cloud“ vorgestellt, welches die Grundlage dieser Arbeit darstellt. Auch die Entwicklung des Programms zur Lastverteilung in der Cloud ist in diesem Kapitel dargestellt.

Kapitel 4 stellt das Kostenmodell von Amazon Elastic Compute Cloud (kurz: EC2) vor, auf dem die Evaluation basiert. Im darauf folgendem Kapitel werden dann die Strategien erläutert; Die Evaluation der Strategien auf theoretischer Basis in Hinblick auf das Szenario des Projektmoduls sowie die Evaluation der Implementierungen auf praktischer Basis erfolgen danach.

Das letzte Kapitel liefert dann Fazit und Ausblick der Bachelorarbeit. Dabei wird auf die Evaluation, sowie auf offene Themen und Verbesserungsvorschläge, die während der Arbeit aufgetreten sind, eingegangen.

Kapitel 2

Grundlagen und Technologien

Die hier vorgestellten Grundlagen und Technologien wurden genutzt, um diese Bachelorarbeit zu erstellen. Für den Leser wird hier ein Überblick geschaffen, um ihm das Verständnis der Arbeit zu erleichtern.

2.1. Cloud Computing

Cloud Computing ist ein Begriff für ein Verfahren, um sich Rechenleistung, Speicherplatz oder Programme von einer anderen Firma zu leihen. Diese stellt standortfremd Rechner zur Verfügung, die je nach Bedarf von Kunden genutzt und über das Internet überall erreicht werden können. Auf Anfrage können die vorhanden Ressourcen automatisch erweitert werden, ohne dass menschliche Interaktion nötig ist. Diese Dynamik und Elastizität stellt den wichtigsten Unterschied zu einem festen Rechenzentrum dar, da beim Kunden der Eindruck entsteht, dass die Ressourcen endlos seien. Bezahlt wird in der Regel pro Rechner oder Programm für einen bestimmten Zeitraum. Sowohl Firmen als auch Privatanwender können den Service einer Cloud in Anspruch nehmen [Mell und Grance 2011].

2.2. Kieker

Das Kieker Framework wurde unter Prof. Dr. Wilhelm Hasselbring entwickelt. Es dokumentiert und analysiert das Verhalten von großen Softwaresystemen zur Laufzeit und gibt Aufschluss über die Leistung und den Kontrollfluss der beobachteten Software. Durch konfigurierbare Reader und Writer können Analysen online und offline durchgeführt werden. Des Weiteren können zusätzliche Plugins die Architektur des Softwaresystems extrahieren und visualisieren [Rohr u. a. 2008; van Hoorn u. a. 2009; 2012].

2.3. SLAstic

SLAstic ist ein von André van Horn entwickeltes architekturbasierendes Programm zur Verwaltung von Online Kapazitäten um Ressourcen effizienter zu nutzen. Es beobachtet Softwaresysteme zur Laufzeit, analysiert deren Verhalten und trifft Voraussagen über zukünftige Auslastungen. Aufgrund dieser Daten kann SLAstic die Softwaresysteme rekonfigurieren, um sie an die vorhandenen Ressourcen anzupassen.

SLAstic bietet auch die Möglichkeit zur adaptiven Lastverteilung in einer Cloud nach bestimmten Kriterien. Schnittstellen zur Amazon Cloud oder Eucalyptus sind bereits vorhanden [Massow u. a. 2011; Rohr u. a. 2008].

2.4. OpenStack

OpenStack ist eine frei verfügbare, von Rackspace Hosting und der NASA entwickelte Cloud Computing Plattform für Firmen, Regierungen und Privatpersonen. Inzwischen wird das Projekt von großen Firmen wie SUSE Linux oder Dell unterstützt. Ziel von OpenStack ist es, für alle Arten von Clouds eine einfach implementierbare und dennoch umfassende Lösung zu liefern [*OpenStack Homepage*].

Im Rahmen des Bachelorprojektes wurde auf zwei Computern mit 4-Kern-Prozessoren eine OpenStack Cloud mit maximal acht möglichen Knoten betrieben. Ein dritter Computer diente als Verwaltungseinheit und Schnittstelle für den Benutzer.

2.5. Amazon EC2

Amazon ist ein Online-Händler, der Mitte 2002 die „Amazon Web Services“ (AWS) gestartet hat, mit denen Firmen und Privatpersonen ein breites Spektrum an Diensten in der Cloud angeboten wird. Datenverarbeitung, Datenspeicher und Datenbankdienste können dynamisch in Anspruch genommen werden.

„Amazon Elastic Compute Cloud“ (EC2) ist ebenfalls ein Web Service von Amazon, der für die Elastizität und die Anpassung der Rechnerkapazität in der Cloud zuständig ist. Dadurch bietet sich die Möglichkeit, Systeme ausfallsicher zu gestalten [*Amazon Elastic Compute Cloud*].

Kapitel 3

Projektmodul „Kieker in the Cloud“

An der Christian-Albrecht-Universität zu Kiel fand vom 09.04.2013 - 24.07.2013 das Projektmodul „Software Engineering - Kieker in the Cloud“ statt. Inhalt und Aufgabe des Projekts war es Kieker in die Cloud zu portieren.

3.1. Überblick

Das in der Arbeitsgruppe Software Engineering entwickelte Kieker wurde angepasst, um auch in der Cloud betreiben werden zu können. Dies bildete die Grundlage für diese Bachelorarbeit. Abbildung 3.1 beschreibt einen Zustand von Kieker im laufenden Betrieb der Cloud. Die einzelnen Komponenten werden im Folgenden beschrieben.

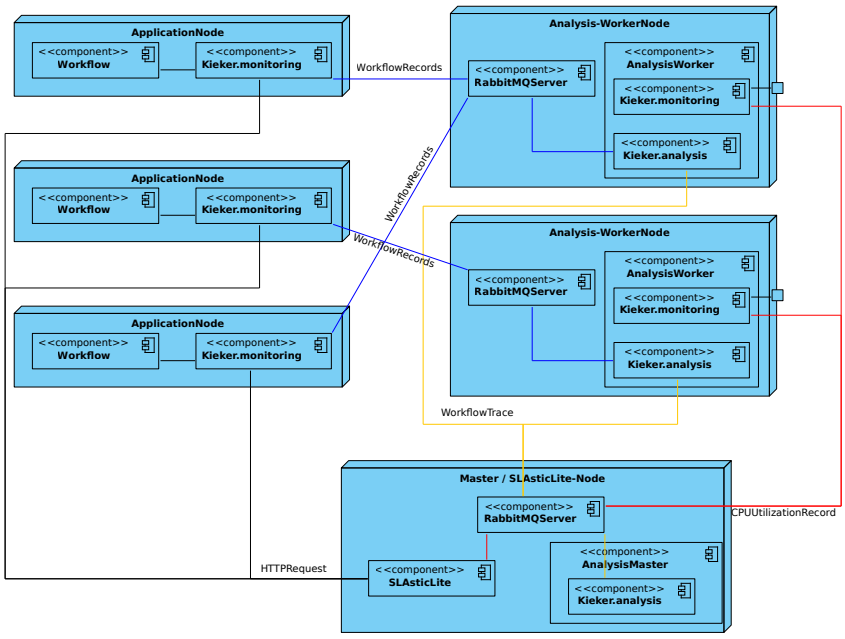

Abbildung 3.1. Kieker in the Cloud

3.1.1. Application Node

Auf einem *Application Node* läuft eine Instanz von *Kieker.monitoring* und ein *JBPM-Workflow*, der eine Annäherungen an π berechnet. Die Monitoring Komponente erstellt für jedes Ereignis innerhalb des Workflows einen *WorkflowRecord* und schickt diesen an einen *RabbitMQ Server* eines *Analysis Workers*. Um die Last gleichmäßig zu verteilen, bekommt die Komponente in regelmäßigen Abständen auf Anfrage eine neue IP eines *RabbitMQ Servers* zugewiesen.

3.1.2. Analysis Worker

Auf einem *Analysis Worker* läuft eine Instanz von *Kieker.analysis* und *Kieker.monitoring* sowie ein *RabbitMQ Server*. Die Analysis Komponente erhält ihre Daten vom *RabbitMQ Server* und wird dabei von der Monitoring Komponente beobachtet. Sie nutzt einen *PartialTraceReconstruction* Filter gefolgt vom *Tee-Filter*, um die verschiedenen *WorkflowRecords* zu gruppieren und als *Workflow(Teil)Traces* an den *Analysis Master* weiterzuleiten. Die *WorkflowTraces* bilden Abschnitte des Ablaufs des Workflows ab. *Kieker.monitoring* sendet dabei die aktuelle CPU Auslastung des Knotens als *CPUUtilizationRecord* an den *RabbitMQ Server* des *Load Balancers*.

3.1.3. Analysis Master

Auf dem *Analysis Master* läuft eine Instanz von *Kieker.analysis*, ein *RabbitMQ Server* sowie ein *Load Balancer*. Aus dem *RabbitMQ Server* können *CPUUtilizationRecords* sowie *WorkflowTraces* ausgelesen werden. Letztere nutzt die Analysis Komponente, um mit drei Filtern (*FinalTraceReconstruction*, *AgglomerationFilter* und *TeeFilter*) die geteilten *WorkflowTraces* zusammenzufügen und zur weiteren Bearbeitung bereitzustellen.

3.1.4. Cloud & Image

Als Plattform für die Cloud diente *OpenStack* (Version Grizzly). Zusätzlich wurden die Pakete *keystone*, *nova*, *quantum* und *glance* installiert, um Benutzer- und Imageverwaltung sowie Netzwerk- und Virtualisierungsunterstützung zu ermöglichen.

Als Image diente für beide Knoten ein *Ubuntu* 12.04 mit *JDK* und *RabbitMQ*, auf dem die Projektdateien abgelegt wurden. Per *Shellskript* wurde die globale Instanz nach dem Startvorgang angepasst, um Analyse oder Monitoring zu starten.

3.1.5. Lastenverteilung

Als *Load Balancer* wurde das an der Universität entwickelte SLAstic analysiert und eine minimierte Version mit dem Namen SLAstic Lite entwickelt. Die genaue Funktion wird im nächsten Unterkapitel erläutert.

3.2. SLAstic Lite

Im Rahmen des Projektmoduls wurde *SLAstic Lite*, eine minimierte Version des von André van Horn entwickelten *SLAstic*, entwickelt. Die Lite Version ist für das Load Balancing, die IP Verteilung an Kieker, sowie für die Kommunikation mit der Cloud verantwortlich.

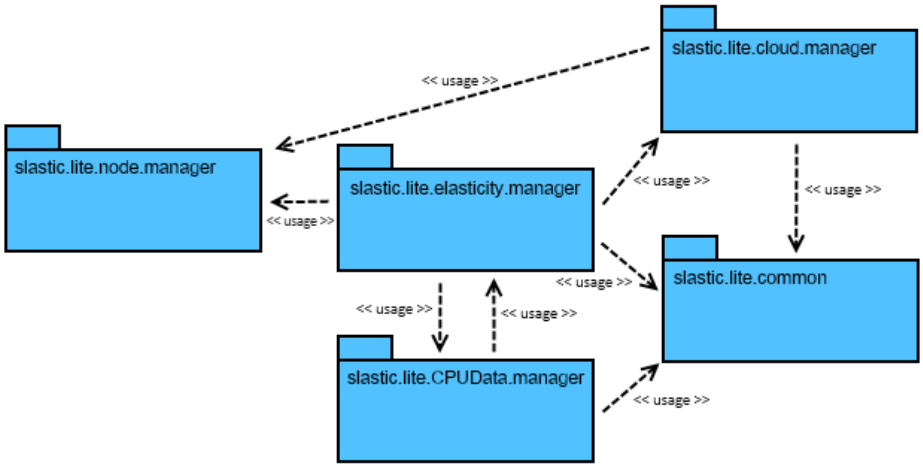

Abbildung 3.2. Paketdiagramm von SLAstic Lite

Abbildung 3.2 zeigt die einzelnen Pakete von *SLAstic Lite* und deren Beziehungen zueinander. Im Folgenden werden die Funktionen der Pakete erläutert:

3.2.1. Elasticity Manager

Der *Elasticity Manager* stellt den Kern von *SLAstic Lite* dar. Er verwaltet die *Elastizitässtrategien* für das intelligente Nutzen der *Analysis Worker* und delegiert zwischen dem *Node Manager*, der Cloud und dem *Data Manager*. Die anfallenden Daten der CPU Auslastung der einzelnen Knoten werden der ausgewählten Strategie entsprechend analysiert und verarbeitet. Das Ergebnis dieser Evaluation kann ein Starten oder Herunterfahren eines *Analysis Workers* bewirken.

SLAstic Lite wurde im Rahmen des Projektes mit einer *Elastizitässtrategie* implementiert. Diese wird im Abschnitt 5.1 erläutert.

3.2.2. Data Manager

Der *CPU Data Manager* empfängt die CPU Auslastung von *Analysis Worker* Knoten über einen *RabbitMQ Server* und verpackt die Daten zusammen mit weiteren wichtigen Informationen, wie der IP Adresse oder der errechneten durchschnittlichen Auslastung über einen bestimmten Zeitraum, in einem Container und leitet diesen an den *Elasticity Manager* weiter.

3.2.3. Node Manager

Der *Node Manager* hält eine synchronisierte Liste aller bekannten *Analysis Worker* Knoten in der Cloud. Die Daten liegen in einem bestimmten Format vor, welches von einem Interface vorgegeben wird. Knoten in der Liste können abgefragt, blockiert, entfernt oder neu hinzugefügt werden.

SLAstic Lite wurde im Rahmen des Projektes mit einer IP-Verteilungs-Strategie implementiert. Diese reagiert auf HTTP Anfragen vom *Kieker Monitoring* auf *Application Nodes* über einen *Webservice*. Als Anwort wird eine IP Adresse per Round-Robin Verfahren ausgewählt und zurück gesendet.

3.2.4. Cloud Manager

Der *Cloud Manager* gibt durch ein Interface vor, welche Methoden jede implementierte Cloud zur Verfügung stellen muss.

SLAstic Lite wurde im Rahmen des Projektes mit der *OpenStack* Cloud implementiert. Die Daten der Cloud implementieren das Interface des *Node Managers*. Beim Starten und Herunterfahren von Knoten werden Fehler abgefangen und entsprechend reagiert. Zusätzlich werden öffentliche IP Adressen erstellt und an *Analysis Worker* gebunden, da *OpenStack* beim Starten nur interne IPs vergibt.

3.2.5. Common

Das *Common* Paket enthält allgemeine Komponenten wie die Main-Methode zum Starten von *SLAstic Lite*, eine Möglichkeit eine SSH-Verbindung aufzubauen und einer Methode um auf Linux Systemen Befehle in der Konsole auszuführen.

Kapitel 4

Kostenmodelle von Amazon EC2

Um ein Kostenmodell für die Evaluation entwickeln zu können, werden wir die verschiedenen Angebote von Amazon EC2 analysieren und die geeignetsten Eigenschaften übernehmen.

4.1. Modelle

Amazon unterscheidet bei der Vermietung von Cloud Kapazitäten zwischen verschiedenen Modellen, deren Abrechnung pro Stunde pro Instanz erfolgt. Angebrochene Stunden werden voll in Rechnung gestellt. Je nach Standort der Serverfarm variieren diese Preise. Im Folgenden werden wir die wichtigsten Modelle vorstellen [*Amazon Elastic Compute Cloud*].

4.1.1. On-Demand-Instances

On-Demand-Instances können ohne langfristige vertragliche Bindung in Anspruch genommen werden. Sie sind für kurzfristige Projekte oder Testzwecke geeignet. Durch diese kurzfristige Mietung sind die Preise zum Teil über fünfmal höher als bei anderen Modellen. Ein Auszug der Preise für das Gebiet der EU befindet sich auf Abbildung 4.1.

4.1.2. Reserved Instances

Reserved Instances können für eine Vertragslaufzeit von einem oder drei Jahren gemietet werden. Für jede genutzte Instanz muss im Vorfeld ein bestimmter Betrag entrichtet werden. Durch diese Vorauszahlung sinkt der Preis pro Stunde pro Instanz. Die Angebote unterscheiden sich je nach späterer Auslastung des Knotens und Laufzeit des Vertrages. Bei der Mietung von Instanzen mit hoher Auslastung werden auch Stunden in Rechnung gestellt, in denen die Instanzen ungenutzt bleiben. Ab einer bestimmten Höhe von Vorabzahlungen gewährt Amazon verschiedene Rabatte.

On-Demand-Instance – Preise

Region: EU (Irland)

	Linux/UNIX-Nutzung
On-Demand Instances – Standard	
Small (Standard)	$0,065 pro Stunde
Medium	$0,130 pro Stunde
Large	$0,260 pro Stunde
Extra Large	$0,520 pro Stunde
2. Generation standardmäßiger On-Demand-Instances	
Extra Large	$0,550 pro Stunde
Double Extra Large	$1,100 pro Stunde
On-Demand Instances – Micro	
Micro	$0,020 pro Stunde
On-Demand Instances – High-Memory	
Extra Large	$0,460 pro Stunde
Double Extra Large	$0,920 pro Stunde
Quadruple Extra Large	$1,840 pro Stunde
On-Demand Instances – High-CPU	
Medium	$0,165 pro Stunde
Extra Large	$0,660 pro Stunde
Cluster Compute-Instances	
Eight Extra Large	$2,700 pro Stunde
On-Demand-Instances – High-Memory Cluster	
Eight Extra Large	$3,750 pro Stunde
Cluster GPU-Instances	
Quadruple Extra Large	$2,36 pro Stunde
On-Demand Instances – High-I/O	
Quadruple Extra Large	$3,410 pro Stunde
On-Demand Instances– High-Storage	
Eight Extra Large	$4,900 pro Stunde

Abbildung 4.1. Auszug des Kostenmodells von Amazon EC2

4.1.3. Spot Instances

Spot Instances ermöglichen Kunden das Nutzen von überschüssigen Amazon EC2 Instanzen. Es besteht die Möglichkeit, für diese Instanzen ein Gebot abzugeben. Sollte das Gebot den aktuellen Preis für eine Spot Instanz überschreiten, kann man diese nutzen. Der aktuelle Preis wird in Echtzeit über Angebot und Nachfrage berechnet.

Kapitel 5

Strategien zur Skalierung in der Cloud

Um einen Überblick über die Thematik zu erhalten, werden wir in diesem Abschnitt nach bereits existierenden Strategien für Elastizität in einer Cloud recherchieren. Diese werden gegebenenfalls angepasst, um dem gewählten Kostenmodell zu entsprechen. Sollte ein Mangel an nutzbaren Strategien bestehen, müssen eigene entwickelt werden. Um auf die Ähnlichkeiten und Unterschiede der Strategien hinzuweisen, werden vereinfachte Namen verwendet.

5.1. Leistungsstrategie

Die *Leistungsstrategie* kann Knoten bei zu geringer Auslastung für die Verteilung an Kieker blockieren und im Leerlauf herunterfahren. Zusätzlich ermöglicht sie es bei Überschreitung eines konfigurierbaren Wertes für die CPU Auslastung neue Knoten zu starten. Es werden jedoch keine Kostenmodelle berücksichtigt.

5.2. Intelligente Leistungsstrategie

Die *Intelligente Leistungsstrategie* ist eine Erweiterung der *Leistungsstrategie*. Sie kann ebenfalls neue Knoten bei sehr hoher CPU Auslastung starten, allerdings wird dabei der Beginn des Abrechnungszeitraums markiert. Bei zu geringer Auslastung eines Knotens wird die Instanz für die Verteilung blockiert, deren Abrechnungszeitraum als nächstes endet, und im Leerlauf herunterfahren.

5.3. Auslastungszeitraum-Strategie

Die *Auslastungszeitraum-Strategie* erweitert die *Leistungsstrategie*. Sie kann Knoten bei zu geringer Auslastung für die Verteilung blockieren und im Leerlauf herunterfahren. Zusätzlich ermöglicht sie es bei sehr hoher CPU Auslastung über einen konfigurierbaren Zeitraum neue Knoten zu starten. Dadurch werden bei kurzen Lastspitzen keine neuen Knoten gestartet. Es werden keine Kostenmodelle berücksichtigt.

5.4. Intelligente Auslastungszeitraum-Strategie

Die *Intelligente Auslastungszeitraum-Strategie* erweitert die *Auslastungszeitraum-Strategie* mit den Eigenschaften der *Intelligente Leistungsstrategie*. Bei zu geringer Auslastung eines Knotens wird die Instanz für die Verteilung blockiert, deren Abrechnungszeitraum als nächstes endet, und im Leerlauf herunterfahren. Zusätzlich ermöglicht sie es bei Überschreitung eines konfigurierbaren Wertes für die CPU Auslastung über einen konfigurierbaren Zeitraum neue Knoten zu starten. Dabei wird der Beginn des Abrechnungszeitraums markiert. Dadurch werden bei kurzen Lastspitzen keine neuen Knoten gestartet.

5.5. Reservestrategie

Die *Reservestrategie* kann Knoten bei zu geringer Auslastung für die Verteilung blockieren und hält sie dann für den Rest des Abrechnungszeitraums in Reserve. Zusätzlich ermöglicht sie es bei sehr hoher CPU Auslastung Knoten aus der Reserve zu reaktivieren. Ist die Reserve erschöpft, wird ein neuer Knoten gestartet und dabei der Beginn des Abrechnungszeitraums markiert.

5.6. Intelligente Reservestrategie I

Die *Intelligente Reservestrategie I* blockiert bei zu geringer Auslastung eines Knotens die Instanz für die Verteilung, deren Abrechnungszeitraum als nächstes endet, und hält sie dann für den Rest des Abrechnungszeitraums in Reserve. Zusätzlich ermöglicht sie es bei sehr hoher CPU Auslastung Knoten aus der Reserve zu reaktivieren. Ist die Reserve erschöpft, wird ein neuer Knoten gestartet und dabei der Beginn des Abrechnungszeitraums markiert.

5.7. Intelligente Reservestrategie II

Die *Intelligente Reservestrategie II* verhält sich wie die *Intelligente Reservestrategie I*, jedoch wird der Knoten blockiert, dessen Abrechnungszeitraum als letztes endet. Alle anderen Eigenschaften sind identisch.

5.8. Auslastungszeitraum-Strategie mit Reserve

Die *Auslastungszeitraum-Strategie mit Reserve* kombiniert die Eigenschaften der *Auslastungszeitraum-Strategie* und der *Reservestrategie*. Sie blockiert Knoten bei zu geringer Auslastung für die Verteilung und hält sie dann für den Rest des Abrechnungszeitraums in Reserve. Zusätzlich ermöglicht sie es bei sehr hoher CPU Auslastung über eine konfigurierbare Zeitspanne Knoten aus der Reserve zu reaktivieren. Ist die Reserve erschöpft, wird ein neuer Knoten gestartet und dabei der Beginn des Abrechnungszeitraums markiert.

5.9. Intelligente Auslastungszeitraum-Strategie mit Reserve I

Die *Intelligente Auslastungszeitraum-Strategie mit Reserve Strategy I* erweitert die *Intelligente Auslastungszeitraum-Strategie* mit der *Intelligente Reservestrategie I*. Bei zu geringer Auslastung blockiert sie den Knoten für die Verteilung, dessen Abrechnungszeitraum als nächstes endet, und hält ihn dann für den Rest des Abrechnungszeitraums in Reserve. Zusätzlich ermöglicht sie es bei sehr hoher CPU Auslastung Knoten aus der Reserve zu reaktivieren. Ist die Reserve erschöpft, wird über eine konfigurierbare Zeitspanne beobachtet, ob der Schwellenwert überschritten bleibt. Tritt dieses Ereignis ein, wird ein neuer Knoten gestartet und dabei der Beginn des Abrechnungszeitraums markiert.

5.10. Intelligente Auslastungszeitraum-Strategie mit Reserve II

Die *Intelligente Auslastungszeitraum-Strategie mit Reserve II* verhält sich wie die *Intelligente Auslastungszeitraum-Strategie mit Reserve I*, jedoch wird der Knoten blockiert, dessen Abrechnungszeitraum als letztes endet. Alle anderen Eigenschaften sind identisch.

Kapitel 6

Evaluation der Strategien auf theoretischer Basis in Hinblick auf das Szenario des Projektmoduls

Die Strategien werden unter verschiedenen Aspekten miteinander verglichen und auf Kompatibilität mit Kieker bzw. SLAstic Lite bewertet. Als erstes folgt eine Beschreibung des Evaluationsszenarios. Danach gehen wir auf das Auswertungsschema ein. Nachdem alle Strategien einzeln evaluiert wurden, erfolgt eine Zusammenfassung der Ergebnisse.

Im Rest der Arbeit wird das Kostenmodell auf periodische Zeitabrechnung eingeschränkt. Als einen Abrechnungszeitraum wählen wir eine Stunde. Damit vereinfachen wir die Evaluation und orientieren uns an dem Auszug des Kostenmodells von Amazon EC2.

6.1. Evaluationsszenario

Im Folgenden verwenden wir ein Evaluationsszenario werden, welches die Eigenschaften jeder Strategie berücksichtigt. Abbildung 6.1 zeigt die CPU Auslastung in Abhängigkeit von der Zeit. Es wird angenommen, dass immer der Knoten heruntergefahren wird, der als letztes gestartet wurde. Ausnahmen bilden hierbei Strategien, die explizit anders vorgehen.

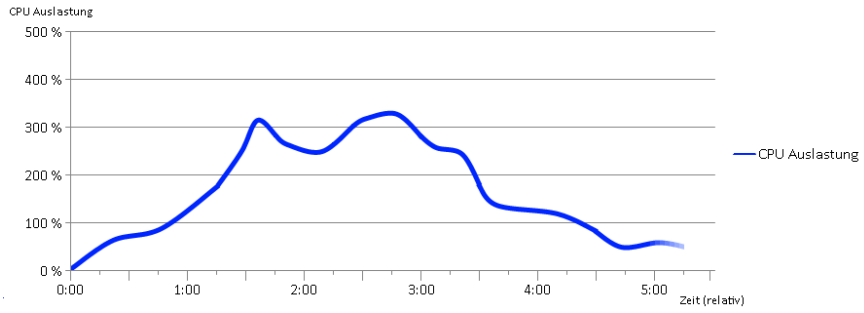

Abbildung 6.1. Evaluationsszenario für die theoretische Evaluation

6. Evaluation der Strategien auf theoretischer Basis in Hinblick auf das Szenario des Projektmoduls

Zu Beginn des Szenarios steigt die Lastkurve stetig an und bildet einen spitzen Peak bei circa 330%. Danach fällt sie auf 240% und erreicht dann erneut für einen längeren Zeitraum die 330%. Nach insgesamt circa drei Zeiteinheiten fällt die Kurve wieder ab und bleibt nach viereinhalb Zeiteinheiten unter 80%. Das gesamte Szenario dauert $5\frac{1}{2}$ Zeiteinheiten.

6.2. Parameter

In der Konfigurationsdatei von *SLAstic Lite* sind Einstellungen vorgegeben, die die Funktionsweise der Strategien beeinflussen. In diesem Abschnitt werden die für die Evaluation verwendeten Werte erläutert.

<high> 90% Bei einer durchschnittlichen Gesamtauslastung der Knoten von 90% oder höher wird versucht, ein neuer Knoten zu starten.

<low> 10% Sollte der Wert von <*high*> nicht überschritten sein, so wird überprüft, ob ein Knoten zu 10% oder weniger ausgelastet ist. Trifft dieses Ereignis ein, wird ein Knoten entsprechend der Strategie für die weitere Verteilung an *Analysis Worker* gesperrt.

<idle> 5% Erreicht ein für die Verteilung gesperrter Knoten eine Auslastung von 5% oder weniger, so wird dieser heruntergefahren.

load limit 3 Der *Data Manager* muss mindestens 3 CPU Auslastungsdaten eines Knotens erhalten, eher die Durchschnittsauslastung dieses Knotens an die ausgewählte Strategie geschickt wird.

peak duration 15 Strategien mit Auslastungszeitraum ignorieren für 15 Minuten einen Anstieg der Last. Sollte die durchschnittliche Gesamtauslastung nach diesem Zeitraum weiterhin angestiegen sein, so wird ein neuer Knoten gestartet.

6.3. Auswertungsschema

Um die Algorithmen miteinander vergleichen zu können, werden für jede Strategie zwei Graphen erstellt. Abbildung 6.2 zeigt die Wertebereiche der Graphen in Abhängigkeit von der Zeit. Die Farben geben Aufschluss darüber, wie gut der derzeitige Wert ist.

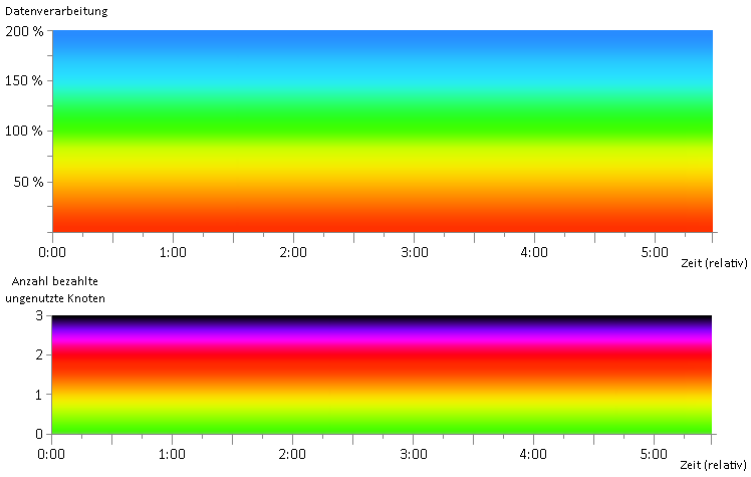

Abbildung 6.2. Auswertungsschema für die theoretische Evaluation

Kriterium 1: Datenverarbeitung
Der erste Graph gibt an, wie viel Daten theoretisch verarbeitet werden können. Ist die Kurve bei 100%, so werden alle anfallenden Daten in Echtzeit verarbeitet. Bei niedrigeren Werten sind nicht genug Knoten gestartet, um die Datenmenge verarbeiten zu können.

Kriterium 1.1: Grundleistung Die Grundleistung zeigt, ob alle anfallenden Daten verarbeitet werden können. Sie ist das primäre Bewertungskriterium und berechnet sich durch das Integral aller Werte kleiner gleich 100% der Kurve. Sind die Werte größer, werden 100% angenommen. Ihr maximaler Wert beträgt 550:

$$D_g = \int_0^{5,5} x \, dx \text{ für } x <= 100$$

Kriterium 1.2: Reserveleistung Auslastungen über 100% kommen zustande, wenn mehr Knoten als nötig für die Analyse vorhanden sind. Dies geschieht bei Reservestrategien, die Knoten nicht sofort herunterfahren. Somit ist Reserveleistung immer mit Mehrkosten verbunden. Für die Evaluation ist dieser Wert deshalb nur das tertiäre Entscheidungsmerkmal. Um die Kurven miteinander vergleichen zu können, wird das Integral der Kurve berechnet. Je größer das Integral, desto höher die Reserveleistung;

$$D_r = \int_0^{5,5} x \, dx \text{ für } x > 100$$

6. Evaluation der Strategien auf theoretischer Basis in Hinblick auf das Szenario des Projektmoduls

Kriterium 2: Mehrkosten
Der zweite Graph gibt an, wie viele nicht benötigte Knoten bezahlt werden müssen. Ist die Kurve bei 0, so sind keine überflüssigen Knoten aktiv. Bei höheren Werten werden Knoten entweder in Reserve gehalten oder der Abrechnungszeitraum von heruntergefahrenen Knoten ist noch nicht zu Ende. Für die Evaluation ist die Anzahl der bezahlten ungenutzten Knoten das sekundäre Entscheidungsmerkmal. Um die Kurven miteinander vergleichen zu können, wird das Integral der Kurve berechnet. Je kleiner das Integral, desto weniger Mehrkosten fallen an. Im Idealfall ist K = 0:

$$K = \int_0^{5,5} y \, dy$$

6.4. Auswertung

Im Folgenden werden die Strategien in zwei Kategorien unterteilt und deren Eigenschaften bestimmt. Danach wird jede Strategie evaluiert.

6.4.1. Kategorien

Die zu evaluierenden Strategien lassen sich in zwei Kategorien unterteilen, die jeweils gleiche Merkmale aufweisen. Daher lässt sich die Evaluation des Knotenverlaufes zusammenfassen. Anhand der Abbildungen 6.3 und 6.4 wird der unterschiedliche Verlauf der Knoten der beiden Kategorien deutlich.

Kategorie A

Strategien der Kategorie A haben eine Grundleistung von 550. Sie starten neue Knoten bei anfallender Last, auch bei kurzen Lastspitzen. Dadurch entstehen neue Abrechnungszeiträume, die gegebenenfalls nicht vollständig genutzt werden.

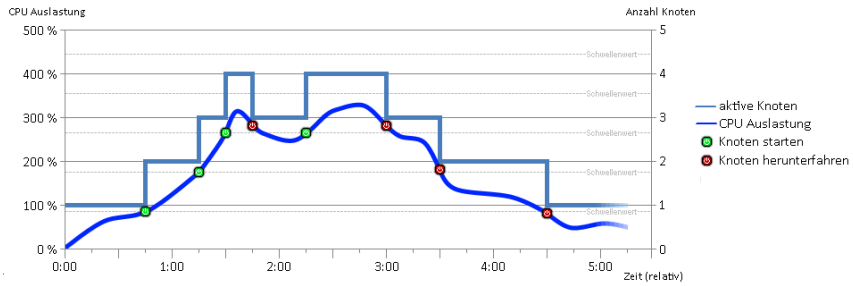

Abbildung 6.3. Strategieverlauf von Strategien der Kategorie A

Die Strategien in dieser Kategorie sind

- *Leistungsstrategie*
- *Intelligente Leistungsstrategie*
- *Reservestrategie*
- *Intelligente Reservestrategie I*
- *Intelligente Reservestrategie II*

Kategorie B

Strategien der Kategorie B starten neue Knoten bei anfallender Last erst verzögert. Deshalb ist es ihnen möglich, kurze Lastspitzen zu ignorieren und dadurch neue Abrechnungszeiträume zu vermeiden. Da die Strategien dieser Kategorie verzögert reagieren, ist auch ihre Grundleistung niedriger als die von Strategien der Kategorie A

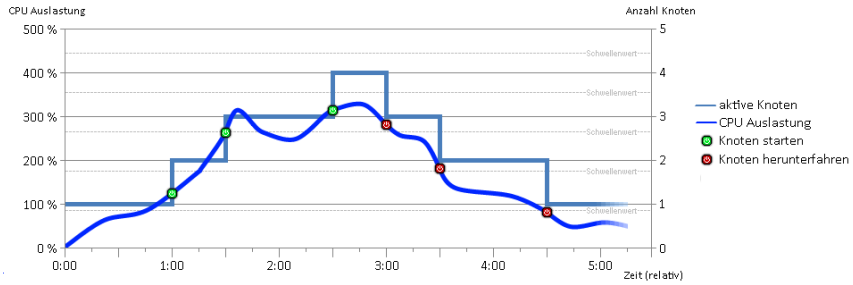

Abbildung 6.4. Strategieverlauf von Strategien der Kategorie B

Die Strategien in dieser Kategorie sind

- *Auslastungszeitraum-Strategie*
- *Intelligente Auslastungszeitraum-Strategie*
- *Auslastungszeitraum-Strategie mit Reserve*
- *Intelligente Auslastungszeitraum-Strategie mit Reserve I*
- *Intelligente Auslastungszeitraum-Strategie mit Reserve II*

6. Evaluation der Strategien auf theoretischer Basis in Hinblick auf das Szenario des Projektmoduls

6.4.2. Leistungsstrategie

Die *Leistungsstrategie* reagiert sofort auf erhöhte Last und startet dementsprechend neue Knoten. Sobald die Last sinkt, werden nicht mehr gebrauchte Knoten heruntergefahren. Da keine Knoten in Reserve gehalten werden, können wir aus Abbildung 6.3 schließen, dass sich die Leistungskurve stets bei 100% befindet.

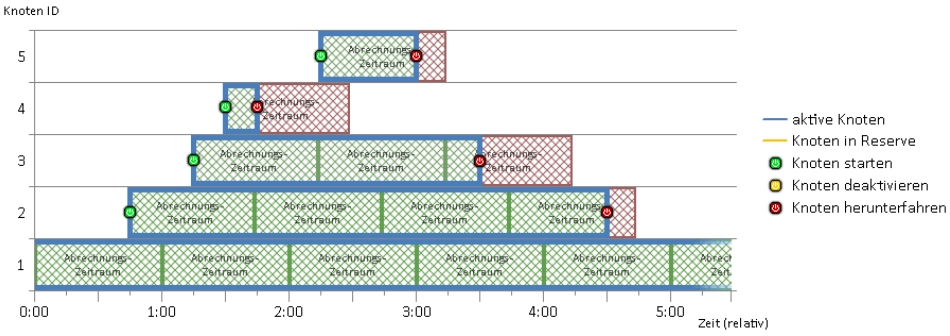

Abbildung 6.5. Knotenverlauf der *Leistungsstrategie*

Anhand des Knotenverlaufs (Abbildung 6.5) können wir bestimmen, wann eine bestimmte Anzahl von Knoten bezahlt wird, ohne dass diese genutzt werden.

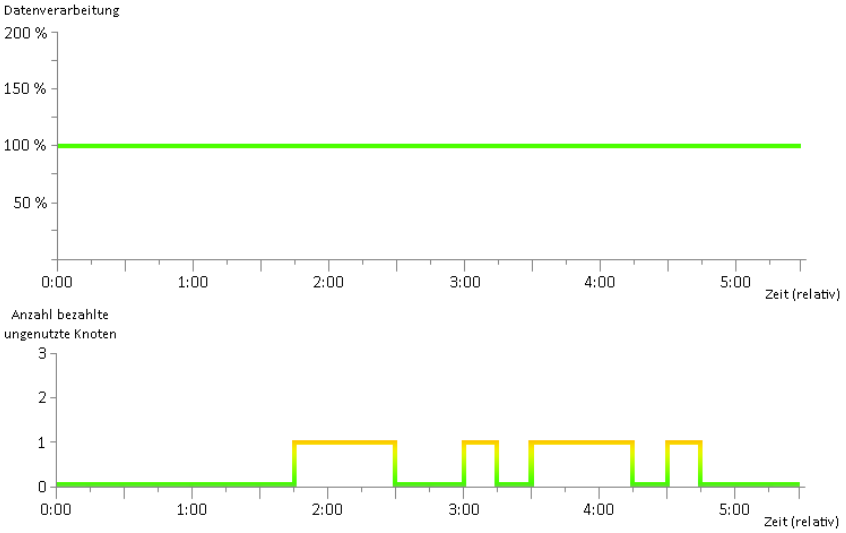

Abbildung 6.6. Auswertung der *Leistungsstrategie*

Die Auswertung der einzelnen Kriterien erfolgt durch die Berechnung der Integrale der Kurven aus Abbildung 6.6. Die *Leistungsstrategie* reagiert auf jeden Lastanstieg und erreicht deshalb die volle Grundleistung. Da sie keine Reserve unterstützt, ist ihre Reserveleistung 0. Ohne Berücksichtigung von Abrechnungszeiträumen entstehen erhöhte Mehrkosten.

Auswertung Kriterium 1: Datenverarbeitung

Kriterium 1.1: Grundleistung

$D_g = \int_0^{5,5} x \, dx$ für $x <= 100$
$= 5,5t * 100\%$
$= 550$

Kriterium 1.2: Reserveleistung

$D_r = \int_0^{5,5} x \, dx$ für $x > 100$
$= 0$

Auswertung Kriterium 2: Mehrkosten

$K = \int_0^{5,5} y \, dy$
$= 0,75t * 1k + 0,25t * 1k + 0,75t * 1k + 0,25t * 1k$
$= 2$

6.4.3. Intelligente Leistungsstrategie

Die *intelligente Leistungsstrategie* reagiert sofort auf erhöhte Last und startet dementsprechend neue Knoten. Sobald die Last sinkt, werden nicht mehr gebrauchte Knoten heruntergefahren. Da keine Knoten in Reserve gehalten werden, können wir aus Abbildung 6.3 schließen, dass sich die Leistungskurve stets bei 100% befindet.

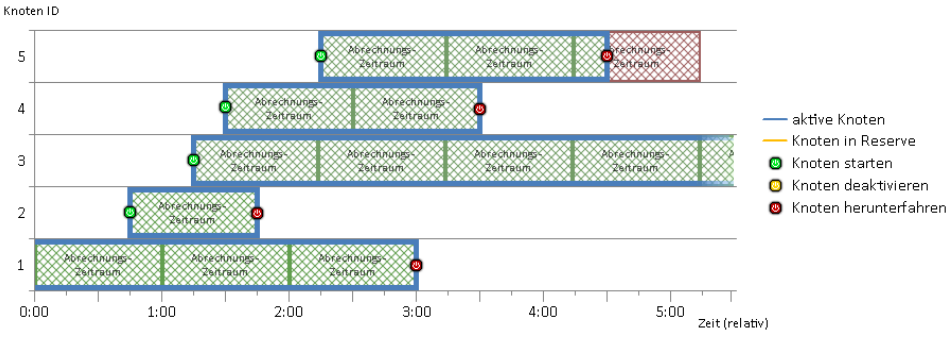

Abbildung 6.7. Knotenverlauf der *intelligenten Leistungsstrategie*

Anhand des Knotenverlaufs (Abbildung 6.7) können wir bestimmen, wann eine bestimmte Anzahl von Knoten bezahlt wird, ohne dass diese genutzt werden. Da diese Strategie die Abrechnungszeiträume berücksichtigt, fallen die Mehrkosten niedrig aus.

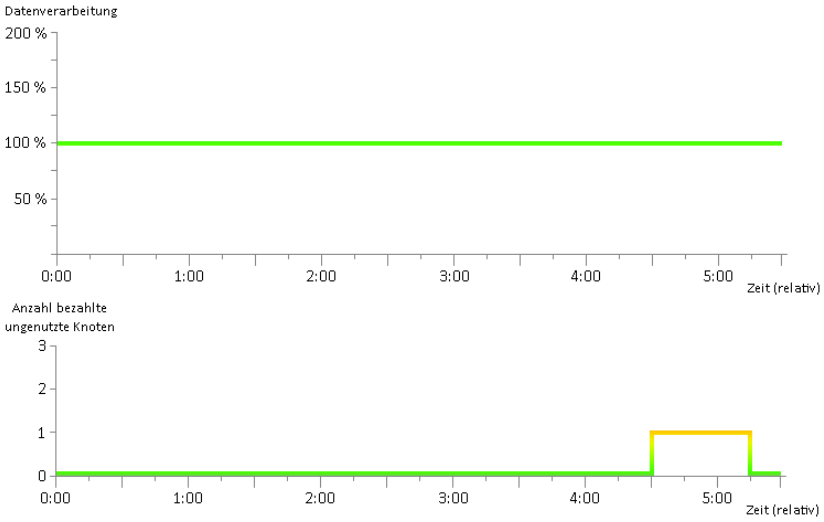

Abbildung 6.8. Auswertung der *intelligenten Leistungsstrategie*

Die *intelligente Leistungsstrategie* erreicht wie die *Leistungsstrategie* volle Grundleistung. Da sie keine Reserve unterstützt, ist ihre Reserveleistung 0. Durch die Berücksichtigung von Abrechnungszeiträumen entstehen geringere Mehrkosten.

Auswertung Kriterium 1: Datenverarbeitung

Kriterium 1.1: Grundleistung

$D_g = \int_0^{5,5} x \, dx$ für $x <= 100$
$= 5,5t * 100\%$
$= 550$

Kriterium 1.2: Reserveleistung

$D_r = \int_0^{5,5} x \, dx$ für $x > 100$
$= 0$

Auswertung Kriterium 2: Mehrkosten

$K = \int_0^{5,5} y \, dy$
$= 0,75t * 1k$
$= 0,75$

6.4.4. Auslastungszeitraum-Strategie

Die *Auslastungszeitraum-Strategie* reagiert verzögert auf erhöhte Last und umgeht so kurze Lastspitzen. Sobald die Last sinkt, werden nicht mehr gebrauchte Knoten heruntergefahren. Da keine Knoten in Reserve gehalten werden, können wir aus Abbildung 6.4 schließen, dass sich die Leistungskurve zum Teil unter 100% befindet.

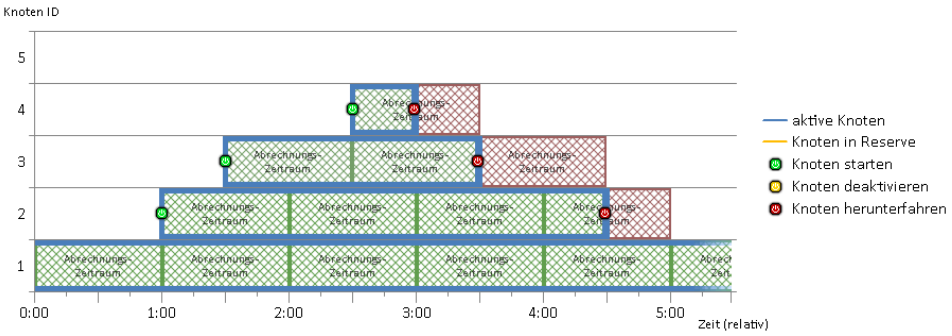

Abbildung 6.9. Knotenverlauf der *Auslastungszeitraum-Strategie*

Anhand des Knotenverlaufs (Abbildung 6.9) können wir bestimmen, wann eine bestimmte Anzahl von Knoten bezahlt wird, ohne dass diese genutzt werden. Durch das versetzte Starten der Knoten beginnen die Abrechnungszeiträume zu anderen Zeiten als bei der *Leistungsstrategie*.

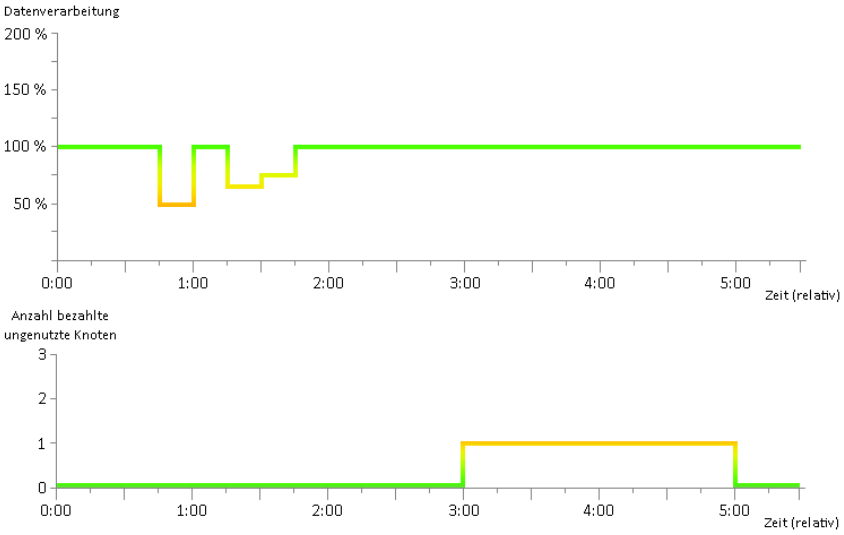

Abbildung 6.10. Auswertung der *Auslastungszeitraum-Strategie*

Die *Auslastungszeitraum-Strategie* regiert nur verzögert auf ansteigende Last und erreicht deshalb nicht die volle Grundleistung. Da sie keine Reserve unterstützt, ist ihre Reserveleistung 0. Ohne Berücksichtigung von Abrechnungszeiträumen entstehen erhöhte Mehrkosten.

Auswertung Kriterium 1: Datenverarbeitung

Kriterium 1.1: Grundleistung

$D_g = \int_0^{5,5} x \, dx$ für $x <= 100$
$= 0{,}75t * 100\% + 0{,}25t * 50\% + 0{,}25t * 100\% + 0{,}25t * 66\frac{2}{3}\% + 0{,}25t * 75\% + 3{,}75t * 100\%$
$= 0{,}25t * 50\% + 0{,}25t * 66\frac{2}{3}\% + 0{,}25t * 75\% + 4{,}75 * 100\%$
$= 522\frac{5}{6}$

Kriterium 1.2: Reserveleistung

$D_r = \int_0^{5,5} x \, dx$ für $x > 100$
$= 0$

Auswertung Kriterium 2: Mehrkosten

$K = \int_0^{5,5} y \, dy$
$= 2t * 1$
$= 2$

6.4.5. Intelligente Auslastungszeitraum-Strategie

Die *intelligente Auslastungszeitraum-Strategie* reagiert verzögert auf erhöhte Last und umgeht so kurze Lastspitzen. Sobald die Last sinkt, werden nicht mehr gebrauchte Knoten heruntergefahren. Da keine Knoten in Reserve gehalten werden, können wir aus Abbildung 6.4 schließen, dass sich die Leistungskurve zum Teil unter 100% befindet.

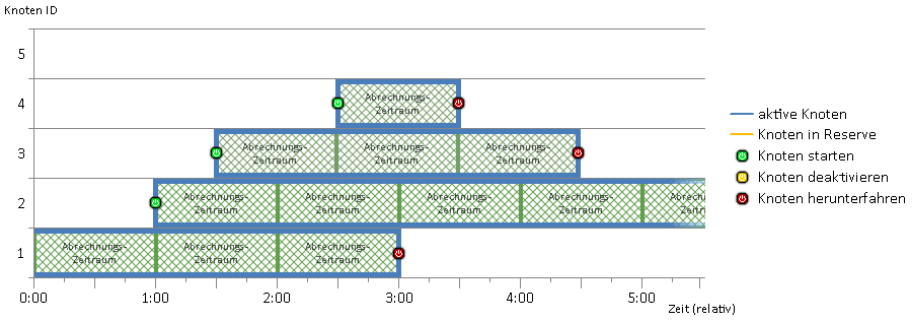

Abbildung 6.11. Knotenverlauf der *intelligenten Auslastungszeitraum-Strategie*

Anhand des Knotenverlaufs (Abbildung 6.9) können wir bestimmen, wann eine bestimmte Anzahl von Knoten bezahlt wird, ohne dass diese genutzt werden. Da diese Strategie die Abrechnungszeiträume berücksichtigt, fallen bei ihr geringere Mehrkosten an als bei der *Auslastungszeitraum-Strategie*.

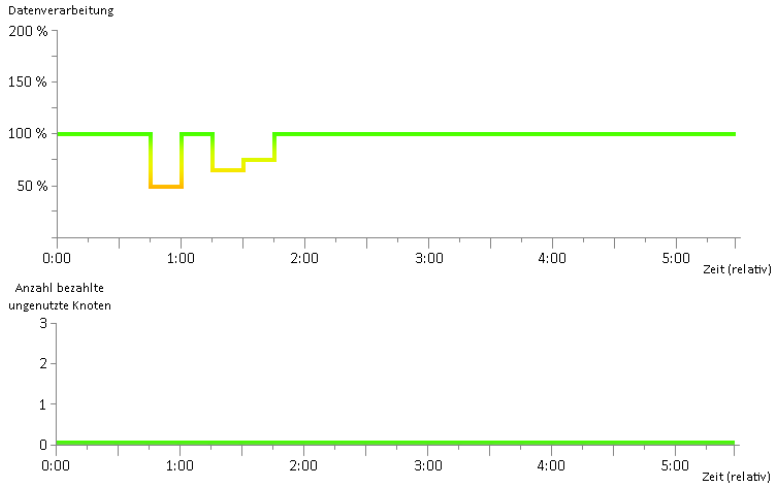

Abbildung 6.12. Auswertung der *intelligenten Auslastungszeitraum-Strategie*

Die *intelligente Auslastungszeitraum-Strategie* erreicht wie die *Auslastungszeitraum-Strategie* nicht die volle Grundleistung. Da sie keine Reserve unterstützt, ist ihre Reserveleistung 0. Durch die Berücksichtigung von Abrechnungszeiträumen entstehen geringere Mehrkosten.

Auswertung Kriterium 1: Datenverarbeitung

Kriterium 1.1: Grundleistung

$D_g = \int_0^{5,5} x \, dx$ für $x <= 100$
= 0,75t * 100% + 0,25t * 50% + 0,25t * 100% + 0,25t * $66\frac{2}{3}$% + 0,25t * 75% + 3,75t * 100%
= 0,25t * 50% + 0,25t * $66\frac{2}{3}$% + 0,25t * 75% + 4,75 * 100%
= $522\frac{5}{6}$

Kriterium 1.2: Reserveleistung

$D_r = \int_0^{5,5} x \, dx$ für $x > 100$
= 0

Auswertung Kriterium 2: Mehrkosten

$K = \int_0^{5,5} y \, dy$
= 0

6.4.6. Reservestrategie

Die *Reservestrategie* reagiert sofort auf erhöhte Last und startet dementsprechend neue Knoten. Sobald die Last sinkt, werden nicht mehr gebrauchte Knoten heruntergefahren oder als Reserve gehalten. Wir können aus Abbildung 6.3 schließen, dass sich die Leistungskurve stets über 100% befindet und eventuell zusätzliche Reserveleistung entsteht.

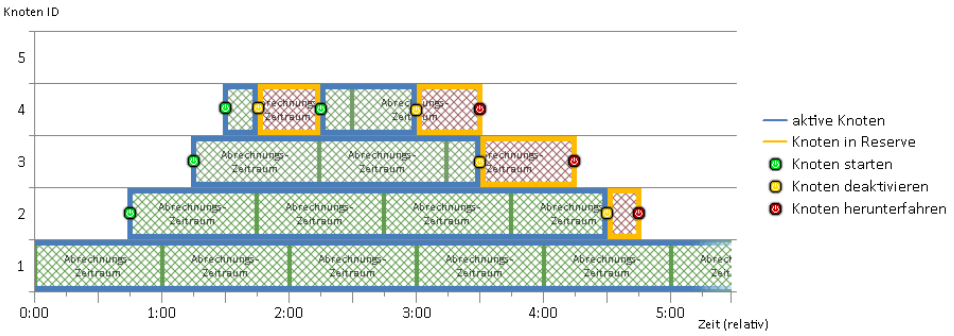

Abbildung 6.13. Knotenverlauf der *Reservestrategie*

Anhand des Knotenverlaufs (Abbildung 6.13) können wir bestimmen, wann eine bestimmte Anzahl von Knoten bezahlt wird, ohne dass diese genutzt werden. Durch das Halten von Knoten in Reserve entstehen andere Abrechnungszeiträume als bei den vorherigen Strategien.

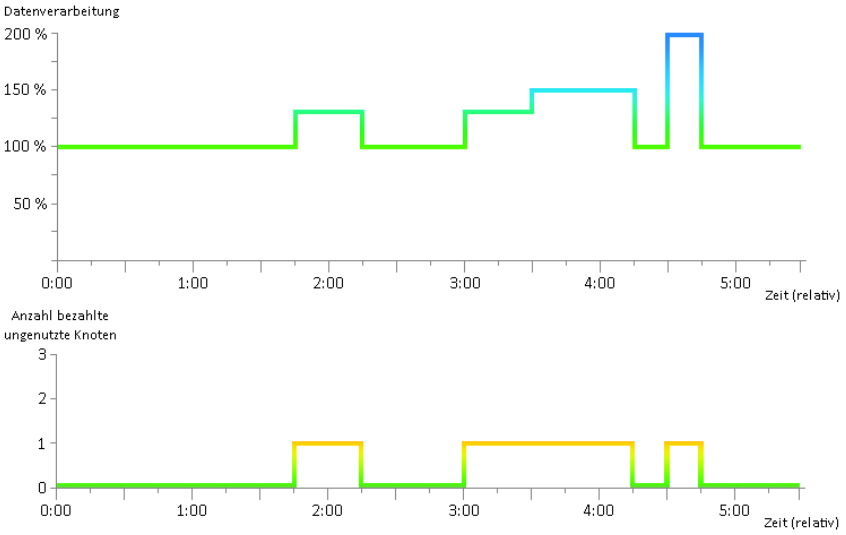

Abbildung 6.14. Auswertung der *Reservestrategie*

Die *Reservestrategie* reagiert auf jeden Lastanstieg und erreicht deshalb die volle Grundleistung. Ohne Berücksichtigung von Abrechnungszeiträumen entstehen erhöhte Mehrkosten, jedoch sind diese immer mit Reserveleistung verbunden.

Auswertung Kriterium 1: Datenverarbeitung

Kriterium 1.1: Grundleistung

$D_g = \int_0^{5,5} x \, dx$ für $x <= 100$
$= 5{,}5t * 100\%$
$= 550$

Kriterium 1.2: Reserveleistung

$D_r = \int_0^{5,5} x \, dx$ für $x > 100$
$= 0{,}5t * 133\frac{1}{3}\% + 0{,}5t * 133\frac{1}{3}\% + 0{,}75t * 150\% + 0{,}25t * 200\% - 2t * 100\%$
$= 1t * 133\frac{1}{3}\% + 0{,}75t * 150\% + 0{,}25t * 200\% - 2t * 100\%$
$= 95\frac{5}{6}$

Auswertung Kriterium 2: Mehrkosten

$K = \int_0^{5,5} y \, dy$
$= 0{,}5t * 1k + 1{,}25t * 1k + 0{,}25t * 1k$
$= 2$

6.4.7. Intelligente Reservestrategie I

Die *intelligente Reservestrategie I* reagiert sofort auf erhöhte Last. Beim Sinken der Last werden nicht mehr gebrauchte Knoten heruntergefahren oder als Reserve gehalten. Wir können aus Abbildung 6.3 schließen, dass sich die Leistungskurve stets über 100% befindet und eventuell zusätzliche Reserveleistung entsteht.

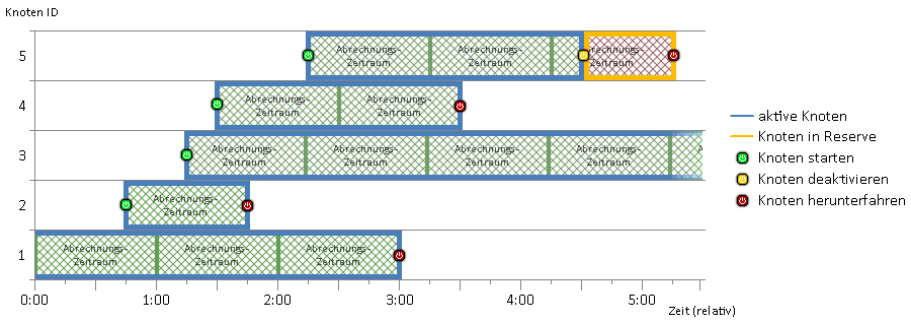

Abbildung 6.15. Knotenverlauf der *intelligenten Reservestrategie I*

Durch den Knotenverlaufs (Abbildung 6.15) können wir bestimmen, wann eine bestimmte Anzahl von ungenutzten Knoten bezahlt wird. Die *intelligente Reservestrategie I* nutzt für die Reserve Knoten, deren Abrechnungszeitraum als nächstes endet.

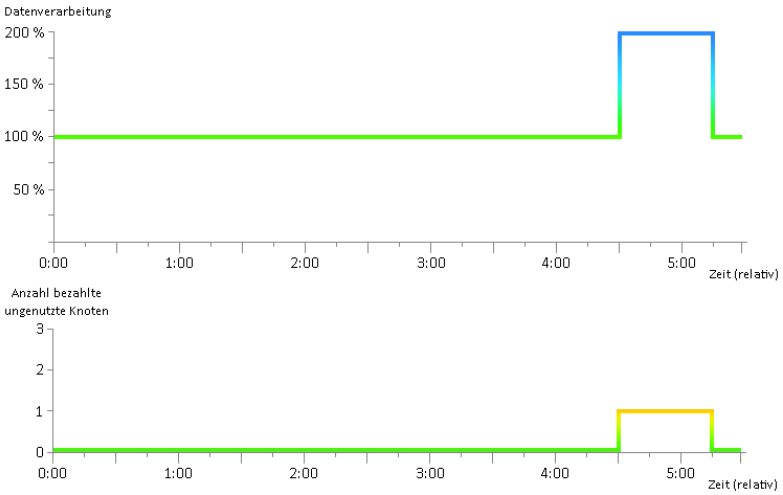

Abbildung 6.16. Auswertung der *intelligenten Reservestrategie I*

Die *intelligente Reservestrategie I* hat im Vergleich zur *Reservestrategie* durch die Berücksichtigung von Abrechnungszeiträumen geringere Mehrkosten, jedoch auch eine geringere Reserveleistung.

Auswertung Kriterium 1: Datenverarbeitung

Kriterium 1.1: Grundleistung

$D_g = \int_0^{5,5} x \, dx$ für $x <= 100$
= 5,5t * 100%
= 550

Kriterium 1.2: Reserveleistung

$D_r = \int_0^{5,5} x \, dx$ für $x > 100$
= 0,75t * 200% - 0,75t * 100%
= 75

Auswertung Kriterium 2: Mehrkosten

$K = \int_0^{5,5} y \, dy$
= 0,75t * 1k
= 0,75

6.4.8. Intelligente Reservestrategie II

Die *intelligente Reservestrategie II* reagiert sofort auf erhöhte Last und startet dementsprechend neue Knoten. Sobald die Last sinkt, werden nicht mehr gebrauchte Knoten heruntergefahren oder als Reserve gehalten. Wir können aus Abbildung 6.3 schließen, dass sich die Leistungskurve stets über 100% befindet und eventuell zusätzliche Reserveleistung entsteht.

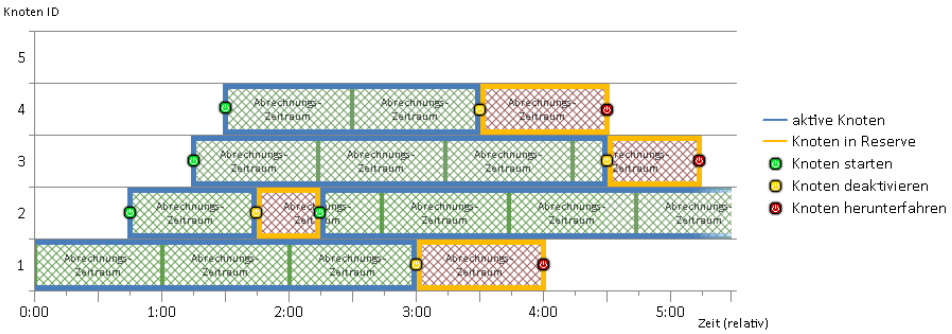

Abbildung 6.17. Knotenverlauf der *intelligenten Reservestrategie II*

Anhand des Knotenverlaufs (Abbildung 6.17) können wir bestimmen, wann eine bestimmte Anzahl von Knoten bezahlt wird ohne genutzt zu werden. Es werden die Knoten für die Reserve genutzt, deren Abrechnungszeitraum als letztes endet.

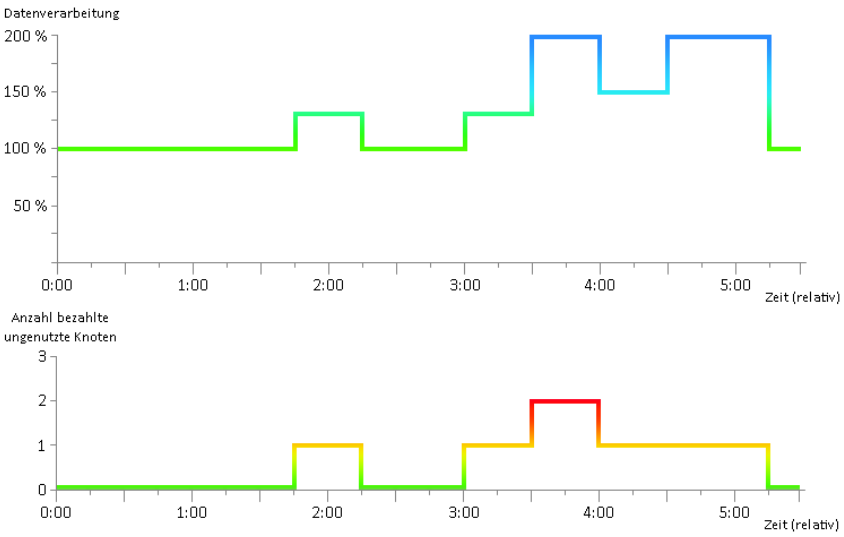

Abbildung 6.18. Auswertung der *intelligenten Reservestrategie II*

Die *intelligente Reservestrategie II* reagiert auf jeden Lastanstieg und erreicht deshalb die volle Grundleistung. Da die Strategie versucht, eine möglichst hohe Reserveleistung zu erzielen, entstehen stark erhöhte Mehrkosten durch ungenutzte Knoten.

Auswertung Kriterium 1: Datenverarbeitung

Kriterium 1.1: Grundleistung

$D_g = \int_0^{5,5} x \, dx$ für $x <= 100$
= 5,5t * 100%
= 550

Kriterium 1.2: Reserveleistung

$D_r = \int_0^{5,5} x \, dx$ für $x > 100$
= 0,5t * $133\frac{1}{3}$% + 0,5t * $133\frac{1}{3}$% + 0,5t * 200% + 0,5t * 150% + 0,75t * 200% - 2,75t * 100%
= 1t * $133\frac{1}{3}$% + 0,5t * 150% + 1,25t * 200% - 2,75t * 100%
= $183\frac{1}{3}$

Auswertung Kriterium 2: Mehrkosten

$K = \int_0^{5,5} y \, dy$
= 0,5t * 1k + 0,5t * 1k + 0,5t * 2k + 1,25t * 1k
= 2,25t * 1k + 0,5t * 2k
= 3,25

6.4.9. Auslastungszeitraum-Strategie mit Reserve

Die *Auslastungszeitraum-Strategie mit Reserve* reagiert verzögert auf erhöhte Last und umgeht so kurze Lastspitzen. Sobald die Last sinkt, werden nicht mehr gebrauchte Knoten heruntergefahren oder als Reserve gehalten. Wir können aus Abbildung 6.4 schließen, dass sich die Leistungskurve zum Teil unter 100% befindet, jedoch eventuell zusätzliche Reserveleistung entsteht.

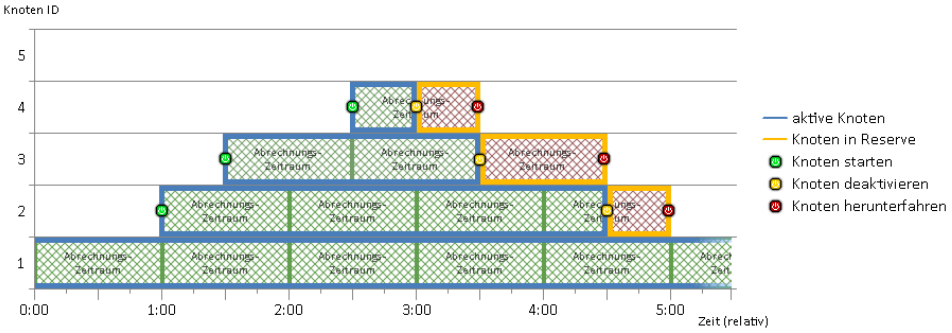

Abbildung 6.19. Knotenverlauf der *Auslastungszeitraum-Strategie mit Reserve*

Anhand des Knotenverlaufs (Abbildung 6.19) können wir bestimmen, wann eine bestimmte Anzahl von Knoten bezahlt wird ohne genutzt zu werden. Das versetzte Starten der Knoten durch Auslastungszeiträume bewirkt verschobene Abrechnungszeiträume.

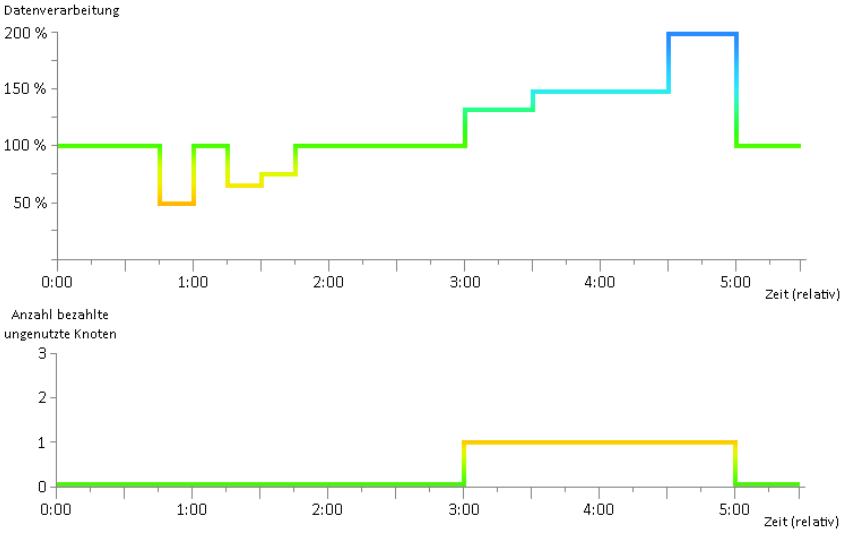

Abbildung 6.20. Auswertung der *Auslastungszeitraum-Strategie mit Reserve*

Die *Auslastungszeitraum-Strategie mit Reserve* regiert nur verzögert auf ansteigende Last und erreicht deshalb nicht die volle Grundleistung. Durch den Einsatz von Reserveknoten können die Mehrkosten in Reserveleistung umgesetzt werden.

6. Evaluation der Strategien auf theoretischer Basis in Hinblick auf das Szenario des Projektmoduls

Auswertung Kriterium 1: Datenverarbeitung

Kriterium 1.1: Grundleistung

$D_g = \int_0^{5,5} x \, dx$ für $x <= 100$

= 0,75t * 100% + 0,25t * 50% + 0,25t * 100% + 0,25t * $66\frac{2}{3}$% + 0,25t * 75% + 3,75t * 100%

= 0,25t * 50% + 0,25t * $66\frac{2}{3}$% + 0,25t * 75% + 4,75 * 100%

= $522\frac{5}{6}$

Kriterium 1.2: Reserveleistung

$D_r = \int_0^{5,5} x \, dx$ für $x > 100$

= 0,5t * $133\frac{1}{3}$% + 1t * 150% + 0,5t * 200% - 2t * 100%

= $116\frac{2}{3}$

Auswertung Kriterium 2: Mehrkosten

$K = \int_0^{5,5} y \, dy$

= 2t * 1k

= 2

6.4.10. Intelligente Auslastungszeitraum-Strategie mit Reserve I

Die *intelligente Auslastungszeitraum-Strategie mit Reserve I* reagiert verzögert auf erhöhte Last und umgeht so kurze Lastspitzen. Sobald die Last sinkt, werden überflüssige Knoten heruntergefahren oder als Reserve gehalten. Aus Abbildung 6.4 folgt, dass keine volle Grundleistung erbracht wird, jedoch eventuell zusätzliche Reserveleistung entsteht.

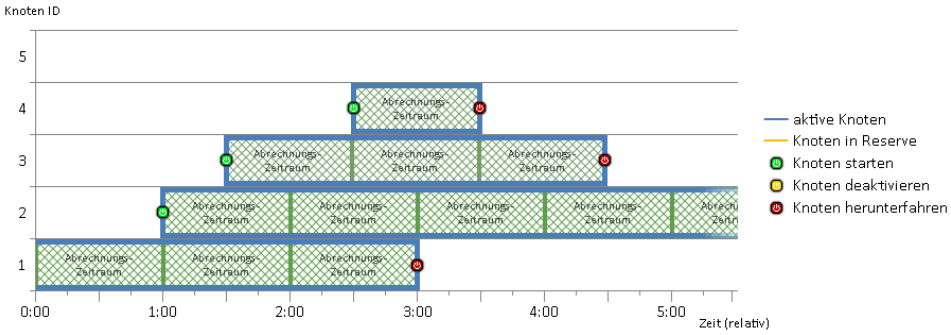

Abbildung 6.21. Knotenverlauf der *intelligenten Auslastungszeitraum-Strategie mit Reserve I*

Anhand des Knotenverlaufs (Abbildung 6.21) können wir bestimmen, wann eine bestimmte Anzahl von Knoten bezahlt wird, ohne dass diese genutzt werden. Da diese Strategie die Abrechnungszeiträume berücksichtigt, fallen bei ihr geringere Mehrkosten an als bei der *Auslastungszeitraum-Strategie mit Reserve*.

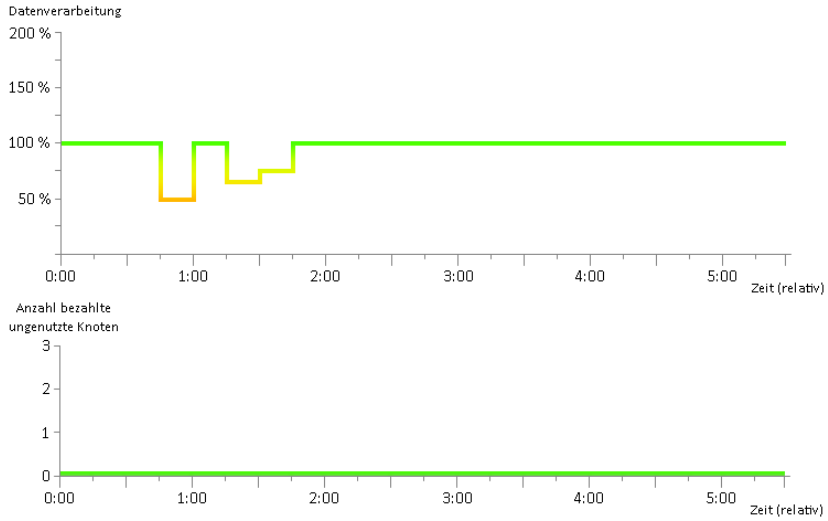

Abbildung 6.22. Auswertung der *intelligenten Auslastungszeitraum-Strategie mit Reserve I*

Da die *intelligente Auslastungszeitraum-Strategie mit Reserve I* auf der *intelligenten Auslastungszeitraum-Strategie* basiert und bei dieser keine Mehrkosten entstehen, ist ihre Evaluation identisch.

Auswertung Kriterium 1: Datenverarbeitung

Kriterium 1.1: Grundleistung

$D_g = \int_0^{5,5} x \, dx$ für $x <= 100$

= 0,75t * 100% + 0,25t * 50% + 0,25t * 100% + 0,25t * $66\frac{2}{3}$% + 0,25t * 75% + 3,75t * 100%

= 0,25t * 50% + 0,25t * $66\frac{2}{3}$% + 0,25t * 75% + 4,75 * 100%

= $522\frac{5}{6}$

Kriterium 1.2: Reserveleistung

$D_r = \int_0^{5,5} x \, dx$ für $x > 100$

= 0

Auswertung Kriterium 2: Mehrkosten

$K = \int_0^{5,5} y \, dy$

= 0

6. Evaluation der Strategien auf theoretischer Basis in Hinblick auf das Szenario des Projektmoduls

6.4.11. Intelligente Auslastungszeitraum-Strategie mit Reserve II

Die *intelligente Auslastungszeitraum-Strategie mit Reserve II* reagiert verzögert auf erhöhte Last und umgeht so kurze Lastspitzen. Sobald die Last sinkt, werden nicht mehr gebrauchte Knoten heruntergefahren oder als Reserve gehalten. Wir können aus Abbildung 6.4 schließen, dass sich die Leistungskurve zum Teil unter 100% befindet, jedoch eventuell zusätzliche Reserveleistung entsteht.

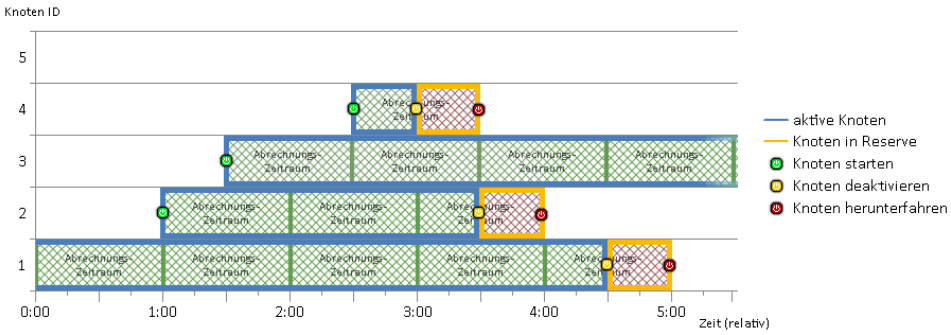

Abbildung 6.23. Knotenverlauf der *intelligenten Auslastungszeitraum-Strategie mit Reserve II*

Anhand des Knotenverlaufs (Abbildung 6.23) können wir bestimmen, wann eine bestimmte Anzahl von Knoten bezahlt wird, ohne dass diese genutzt werden. Es werden immer die Knoten als Reserve gehalten, deren Abrechnungszeitraum als letztes endet.

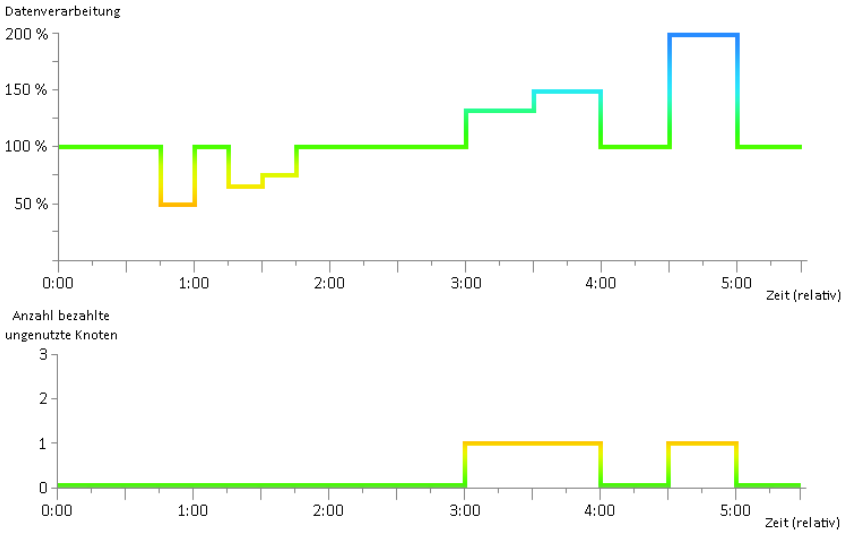

Abbildung 6.24. Auswertung der *intelligenten Auslastungszeitraum-Strategie mit Reserve II*

Die *intelligente Auslastungszeitraum-Strategie mit Reserve II* regiert nur verzögert auf ansteigende Last und erreicht deshalb nicht die volle Grundleistung. Da die Strategie versucht, eine möglichst hohe Reserveleistung zu erzielen, entstehen stark erhöhte Mehrkosten durch ungenutzte Knoten.

Auswertung Kriterium 1: Datenverarbeitung

Kriterium 1.1: Grundleistung

$D_g = \int_0^{5,5} x \, dx$ für $x <= 100$
= 0,75t * 100% + 0,25t * 50% + 0,25t * 100% + 0,25t * $66\frac{2}{3}$% + 0,25t * 75% + 3,75t * 100%
= 0,25t * 50% + 0,25t * $66\frac{2}{3}$% + 0,25t * 75% + 4,75 * 100%
= $522\frac{5}{6}$

Kriterium 1.2: Reserveleistung

$D_r = \int_0^{5,5} x \, dx$ für $x > 100$
= 0,5t * $133\frac{1}{3}$% + 0,5t * 150% + 0,5t * 200% - 1,5t * 100%
= $91\frac{2}{3}$

Auswertung Kriterium 2: Mehrkosten

$K = \int_0^{5,5} y \, dy$
= 1t * 1k + 0,5t * 1k
= 1,5

6.5. Vergleich und Ergebnis

Im Folgenden sind die Ergebnisse der theoretischen Evaluation gewichtet nach den drei Kriterien zu sehen. Da durch die Festlegung der Grundleistung als primäres Kriterium Strategien mit Auslastungszeitraum benachteiligt wurden, sind diese gesondert zu betrachten. Es fällt auf, dass „intelligente“ Strategien besser abschneiden als normale. Eine Ausnahme bildet hier die *intelligente Reservestrategie II*, da diese versucht möglichst lange Knoten als Reserve zu halten.

Tabelle 6.1. Ergebnisse der theoretischen Evaluation

Strategie	**Grundleistung D_g**	**Reserveleistung D_r**	**Mehrkosten K**
intelligente Reservestrategie I	550	75	0,75
intelligente Leistungsstrategie	550	0	0,75
Reservestrategie	550	$95\frac{5}{6}$	2
Leistungsstrategie	550	0	2
intelligente Reservestrategie II	550	$183\frac{1}{3}$	3,25
intelligente Auslastungszeitraum-Strategie mit Reserve I	$522\frac{5}{6}$	0	0
intelligente Auslastungszeitraum-Strategie	$522\frac{5}{6}$	0	0
intelligente Auslastungszeitraum-Strategie mit Reserve II	$522\frac{5}{6}$	$91\frac{2}{3}$	1,5
Auslastungszeitraum-Strategie mit Reserve	$522\frac{5}{6}$	$116\frac{2}{3}$	2
Auslastungszeitraum-Strategie	$522\frac{5}{6}$	0	2

Die kursiv gedruckten Strategien haben am besten in ihrem Bereich abgeschnitten. Sie werden für die praktische Evaluation implementiert und dann in einer OpenStack Cloud unter realen Bedingungen getestet.

6.6. Einschränkungen der Validität

Die erzielten Ergebnisse sind nicht allgemeingültig. Im Folgenden werden Aspekte aufgezeigt, die die Validität der Evaluation einschränken.

6.6.1. Lastkurve

Für die theoretische Evaluation wurde nur eine Lastkurve verwendet. Diese versucht zwar die Besonderheiten jeder Strategie abzudecken, jedoch können andere Lastkurven auch ein anderes Verhalten der Algorithmen hervorrufen. Dadurch kann eine andere Wertungsreihenfolge im direkten Vergleich entstehen, was zu anderen Ergebnissen führt.

6.6.2. Kostenmodell

Um die Strategien auch unter wirtschaftlichen Aspekten miteinander vergleichen zu können, wurde ein Kostenmodell benutzt, das von einer stündlichen Abrechnungsperiode ausgeht. Verkürzt oder verlängert sich der Abrechnungszeitraum, entstehen auch andere Werte der Mehrkosten der Strategien. Bei minutengenauer Abrechnung würden die Mehrkosten vollständig wegfallen.

6.6.3. Hardware

Die Knoten wurden nicht weiter spezifiziert. Sie benötigen keine Zeit zum Hoch- und Herunterfahren und kommunizieren ohne Verlust und Übertragungsverzögerung mit dem *Load Balancer*. Dies sind Idealbedingungen, die in der Realität nicht erreicht werden.

Kapitel 7

Implementierung der drei besten Strategien aus der vorherigen Evaluation

Aus der vorherigen theoretischen Evaluation haben sich drei für das Projektmodul am besten geeignetsten Strategien herausgestellt: Die *Intelligente Reservestrategie I*, die *Intelligente Leistungsstrategie* und die *Intelligente Auslastungszeitraum-Strategie mit Reserve I*. Diese werden in diesem Schritt implementiert. Durch die eigene Entwicklung von SLAstic Lite kann auch der Load Balancer optimiert werden, um die Strategien noch effizienter nutzen zu können.

7.1. Architekturänderungen

Um die Strategien zu implementieren und nutzen zu können, mussten eine Modifikationen an Teilen von SLAstic Lite vorgenommen werden.

Einführung von „Intelligenz“ Um Knoten sinnvoll herunterfahren und als Reserve halten zu können, werden beim Starten von einer neuen Instanz das Datum und die Uhrzeit vermerkt (*INodeData*, *CloudManager*). Diese können auch abgerufen und an die Strategien weitergereicht werden (*NodeManager*, *EventManagerCPULoadDistributor*, *CPUData*, *AverageData*).

Einführung von Reserve Knoten enthalten die Information, ob sie als Backup in Reserve gehalten werden (*INodeData*). Dies muss auch beim weiterleiten der CPU Auslastung an die Strategie berücksichtigt werden (*EventManagerCPULoadDistributor*). Die eigentliche Liste aller Knoten in Reserve hält die entsprechende Strategie selbst, wenn sie die Reservefunktion unterstützt. Beim Starten von neuen Knoten überprüft die Strategie selbst, ob sie noch Knoten in Reserve hat und diese reaktiviert oder den Befehl an den *ElasticityManager* weiterleitet.

Konfiguration In der Konfigurationsdatei von SLAstic Lite (*slasticlite.properties*) wurden die neuen Strategien als optionale Auswahl implementiert. Zusätzlich wurden Werte für die Länge eines Abrechnungszeitraumes und die Dauer eines Auslastungszeitraumes festgelegt. Diese Werte werden dann in den *SettingsContainer* geladen.

Kapitel 8

Evaluation der Implementierungen auf praktischer Basis

Ursprünglich sollten die Strategien an der Christian-Albrechts-Universität zu Kiel in einer OpenStack Cloud getestet werden. Aus Mangel an einem geeigneten Zugang und fehlender Implementierung für die Unterstützung der Amazon Cloud beim Abschluss des Projektes, werden wir für die technische Evaluation ein eigenes entwickeltes Programm nutzen, um eine Cloud zu simulieren.

8.1. VirtuaCloud

VirtuaCloud ist eine in SLAstic Lite integrierte alternative Cloud, um die Auslastung des Systems innerhalb eines Tages zu simulieren. Es interagiert dabei mit dem *EventManager-CPULoadDistributor* indem es CPU Daten liefert, IP Adressen beim *NodeManager* anfragt sowie virtuelle Knoten startet und herunterfährt. Während jeder Simulation wird eine Textdatei erzeugt, die Informationen über den Verlauf der Simulation enthält.

8.1.1. Funktionalität

Abbildung 8.1 zeigt den Ablauf einer Simulation von *VirtuaCloud*. Im Folgenden werden die einzelnen Klassen und ihre spezielle Funktion innerhalb der virtuellen Cloud erläutert.

ServerStarter *(SLAstic Lite)* Der *ServerStarter* aus *SLAstic Lite* unterscheidet zwischen *VirtuaCloud* und anderen Cloud Plattformen. Um eine Simulation zu starten, müssen zum Teil Klassen von *VirtuaCloud* an Stelle der von *SLAstic Lite* initialisiert werden. Zusätzlich muss gewährleistet werden, dass ein Protokoll erzeugt wird.

VirtuaCloudNodeData Die in *VirtuaCloudNodeData* enthaltenen Methoden werden durch das Interface *INodeData* (siehe 3.2.3) vorgegeben. Zusätzlich ist eine toString Methode vorhanden, um die Simulation effektiver für Testzwecke nutzen zu können.

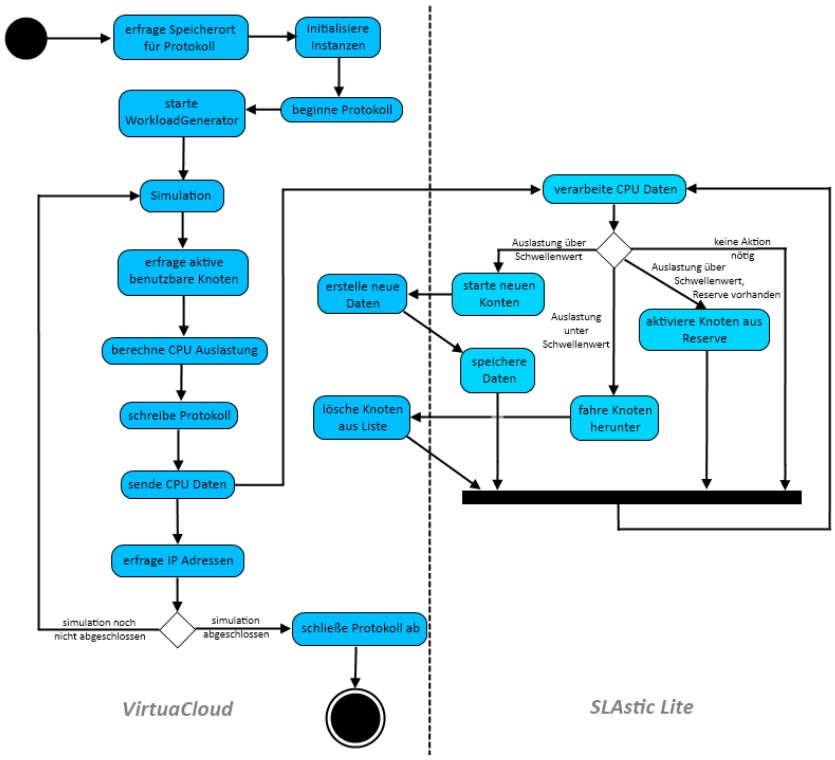

Abbildung 8.1. Aktivitätsdiagramm von *VirtuaCloud*

CloudManagerVirtuaCloud Der *CloudManagerVirtuaCloud* enthält eine Liste aller noch nicht heruntergefahrenen Knoten. Beim Starten eines neuen Knotens werden nach einem vorgegebenen Schema die notwendigen Daten erzeugt und dann ausgegeben. Dabei wird der Knoten auch der Liste des *CloudManagerVirtuaCloud* hinzugefügt. Beim Herunterfahren wird der entsprechende Eintrag aus der Liste gelöscht.

iWorkload Die Lastkurven von *VirtuaCloud* werden über ein Interface implementiert und sind frei austauschbar. Es können also auch eigene Szenarien entwickelt und genutzt werden.

CommunicationRabbitMQ Um *Kieker.monitoring* zu simulieren, enthält diese Klasse eine Methode, um direkt mit dem *CPULoadDistributor* von *SLAstic Lite* zu kommunizieren. Dabei wird für jeden vorhandenen Knoten die entsprechende Auslastung gesendet.

CommunicationWebService Damit *Analysis Worker* auch als *„used“* markiert werden, muss einmal ihre IP Adresse von einem Application Node erfragt worden sein. Die Klasse *CommunicationWebService* kommuniziert direkt mit dem *3.2.3* und stellt eine Methode zur Verfügung, mit der für jeden aktiven Knoten eine IP Anfrage gesendet wird.

TextFactory Der Verlauf jeder Simulation wird automatisch protokolliert. Die *TextFactory* fragt bei ihrer Initialisierung den Benutzer nach dem gewünschten Verzeichnis zum Speichern. Danach werden jede in der Simulation vergangene Minute folgende Daten gespeichert: *aktuelle Uhrzeit*, *Zeit der Simulation*, *Anzahl aktiver Knoten*, *Anzahl Knoten in Reserve*, *gesamte CPU Auslastung* und die *durchschnittliche CPU Auslastung*. Zusätzlich wird jeder aktive Knoten mit *hostname*, *CPU Auslastung* und *Uhrzeit des Hochfahrens* vermerkt.

WorkloadGenerator Der *WorkloadGenerator* enthält einen Thread, der die Simulation durchführt. Er berechnet aus der Workload Funktion die aktuelle CPU Auslastung, kommuniziert über die *Communication* Klassen mit *SLAstic Lite* und sendet der *TextFactory* die notwendigen Daten für das Protokoll. Um aus der gesamten CPU Auaslastung die Auslastung jedes gestarteten Knotens zu berechnen, stehen zwei Funktionen zur Verfügung:

- **accumulate**
  Die Methode erhält die Liste aller aktiven Knoten, die nicht in Reserve sind, sowie die gesamte errechnete CPU Auslastung. Die Knoten der Liste werden beginnend beim Index Null überprüft. Ist ein Knoten nicht *„usable“* erhält er die Auslastung 0.0%. Andernfalls werden den Knoten der Reihe nach 90.0% als Last zugeschrieben, bis die gesamte Auslastung abgearbeitet ist. Übrige Knoten erhalten ebenfalls die Auslastung 0.0%.

- **divide *(beta)***
  Die Methode erhält die Liste aller aktiven Knoten, die nicht in Reserve sind, sowie die gesamte errechnete CPU Auslastung. Die Last wird gleichmäßig auf alle Knoten verteilt. Dabei wird sichergestellt, dass kein Knoten eine Auslastung über 100.0% besitzt und jeder Knoten *„usable“* ist.

  Beim Testen der Methode stellte sich heraus, dass Knoten nur mit großer Verzögerung heruntergefahren wurden, obwohl die selbe Gesamtlast wie bei *accumulate* vorliegt. Des Weiteren wurden teilweise Lastspitzen von Strategien ignoriert, die für maximale Datenverarbeitung ausgelegt sind.

8.2. Evaluationsszenario

Für die praktische Evaluation nutzt *VirtuaCloud* die wird die Funktion einer Kurve eines großen Energiekonzerns benutzt, der auch Kommunikationsdienste anbietet. Die Kurve (Abbildung 8.2) beschreibt den anfallenden Workload im Laufe eines Tages. Damit genug Knoten mit dem anfallenden Workload gestartet werden, wurde der Multiplikator *8* gewählt.

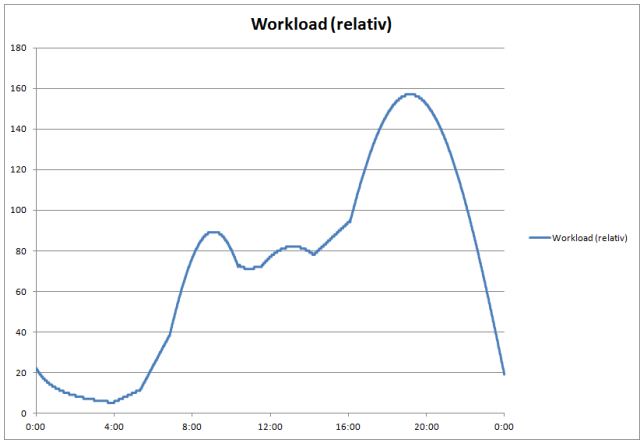

Abbildung 8.2. Simulationskurve eines Tag/Nacht Zyklus

Typischerweise sind in den frühen Morgenstunden nur wenige Nutzer online. Um 4:00 Uhr hat die Kurve ihr Minimum erreicht, danach steigt die Kurve an. Die meisten Leute nehmen nun ihre Arbeit auf. Zur Mittagspause findet ein Einbruch der Kurve statt. Sie steigt zum späten Nachmittag erneut an, bis sie bei ungefähr 20:00 Uhr ihr Maximum erreicht. Danach sinkt die Anzahl der Nutzer.

8.3. Auswertungsschema

Für jede Strategie wird der Knotenverlauf grafisch dargestellt. So kann der Leser die Daten nachvollziehen, die aus den erzeugten Protokollen abgelesen wurden. Abbildung 8.3 zeigt den idealen Knotenverlauf, der durch das Szenario erzeugt wird.

Kriterium 1: Datenverarbeitung
Auf ansteigende Last wird je nach Strategie anders reagiert. Des Weiteren brauchen Knoten eine bestimmte Zeit um hochzufahren oder um aus der Reserve wieder in Bereitschaft versetzt zu werden.

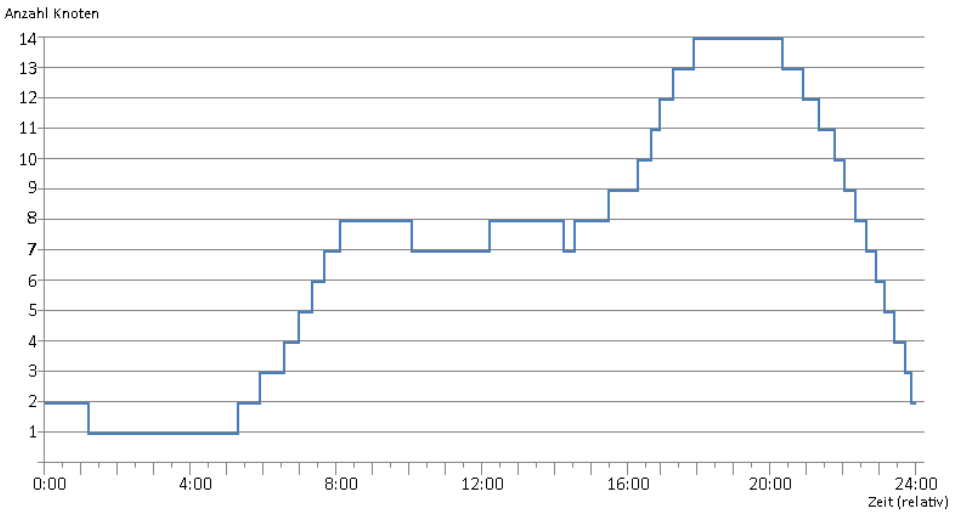

Abbildung 8.3. idealer Knotenverlauf

Kriterium 1.1: Grundleistung Die Grundleistung zeigt, wie viel Prozent der anfallenden Daten verarbeitet werden können. Sie berechnet sich durch der Zeit, die Knoten aktiv und in Benutzung sind, geteilt durch die Zeit, in der eine bestimmte Anzahl von Knoten gefordert ist:

$$D_g = \sum_{i=1}^{n} t_a(i) \div \sum_{i=1}^{n} t_b(i)$$

für n = Anzahl aller Knoten, $t_{b/a}(i)$ = benötigte/aktive Laufzeit des Knotens i

Kriterium 1.2: Reserveleistung Reservestrategien können schneller auf ansteigende Last reagieren, da neue Knoten schneller verfügbar sind. Aus den Simulationsprotokollen wird für jeden Knoten die Zeit abgelesen, die er als Reserve gehalten wird. Je größer der Wert, desto höher die Reserveleistung;

$$D_r = \sum_{i=1}^{n} t_r(i)$$

für n = Anzahl aller Knoten, $t_r(i)$ = Reservezeit des Knotens i

Kriterium 2: Mehrkosten
Die Zeit, die ein Knoten ungenutzt ist, ist ein ungewollter Nebeneffekt von Abrechnungszeiträumen. Je geringer diese Zeit, desto effizienter arbeitet eine Strategie. Die Zeit jedes Knotens lässt sich aus den Protokollen ablesen.

$$K = \sum_{i=1}^{n} t_u(i)$$

für n = Anzahl aller Knoten, $t_u(i)$ = ungenutzte Zeit des Knotens i

8.4. Parameter

Für die praktische Evaluation wurden die selben Parameter verwendet wie für die theoretische Evaluation (siehe Abschnitt 6.2).

8.5. Auswertung

Im Folgenden werden die drei Strategien evaluiert. Dafür wird für jede Strategie das Verfahren *accumulate* simuliert. Die Protokolle der einzelnen Simulationen befinden sich im Anhang.

8.5.1. Intelligente Leistungsstrategie

Der nachfolgende Abschnitt behandelt die praktische Evaluation der *intelligenten Leistungsstrategie*. Eine Erklärung der Funktionsweise befindet sich im Abschnitt 5.2. Die theoretische Evaluation der Strategie ist im Abschnitt 6.4.3 nachzulesen.

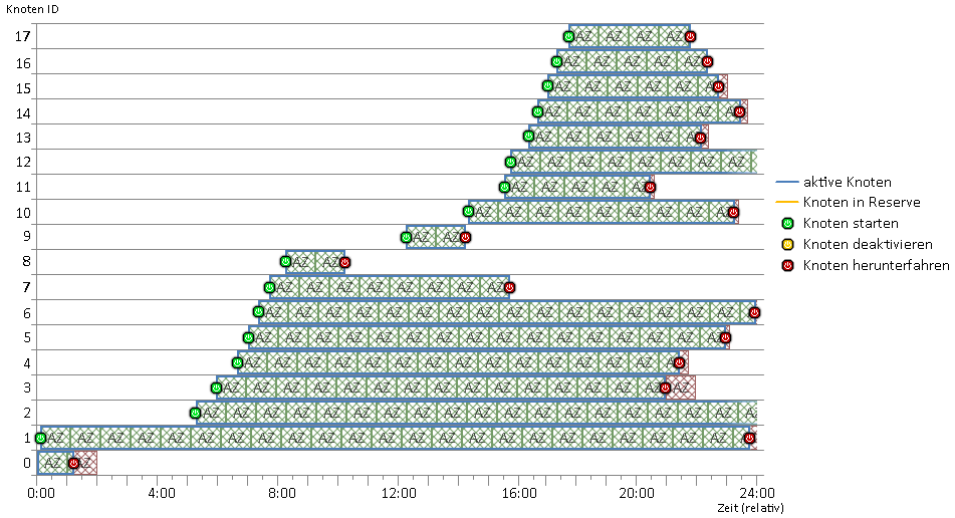

Abbildung 8.4. Knotenverlauf der *intelligenten Leistungsstrategie*

Der Knotenverlauf der *intelligenten Leistungsstrategie* ähnelt sehr dem idealen Kurvenverlauf. Daraus kann eine hohe Datenverarbeitungsrate geschlossen werden. Da diese Strategie keine Reserve unterstützt, kann auch keine Reserveleistung festgestellt werden. Im Gegensatz zur theoretischen Evaluation stellen wir fest, dass Knoten mit einer kleinen Verzögerung gestartet werden. Des Weiteren kann es vorkommen, dass das Herunterfahren eines Knotens den Abrechnungszeitraum überschreitet und einen neuen Zeitraum startet.

Auswertung Kriterium 1: Datenverarbeitung

Kriterium 1.1: Grundleistung

$D_g = \sum_{i=1}^{n} t_a(i) \div \sum_{i=1}^{n} t_b(i)$
= 9773 Min ÷ 9817 Min
= 99,55%

Kriterium 1.2: Reserveleistung

$D_r = \sum_{i=1}^{n} t_r(i)$
= 0 Minuten

Auswertung Kriterium 2: Mehrkosten

$K = \sum_{i=1}^{n} t_u(i)$
= 400 Minuten

8.5.2. Intelligente Reservestrategie I

Der nachfolgende Abschnitt behandelt die praktische Evaluation der *Intelligenten Reservestrategie I*. Eine Erklärung der Funktionsweise befindet sich im Abschnitt 5.6. Die theoretische Evaluation der Strategie ist im Abschnitt 6.4.7 nachzulesen.

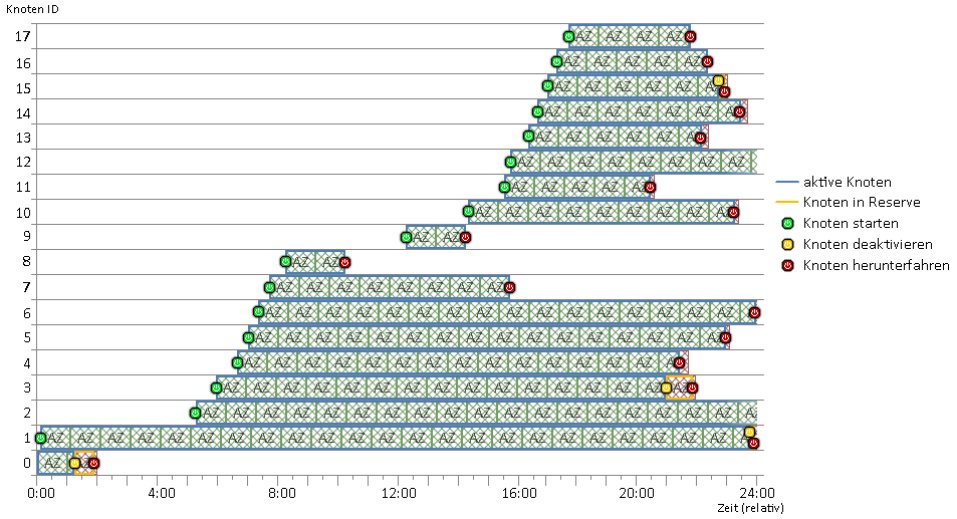

Abbildung 8.5. Knotenverlauf der *intelligenten Reservestrategie 1*

Der Knotenverlauf der *intelligenten Reservestrategie I* ist der selbe, wie der Verlauf der *intelligenten Leistungsstrategie*. Sie unterscheiden sich nur in der Reservehaltung von Knoten. Nicht alle Knoten, die heruntergefahren werden sollen, können von der *intelligenten Reservestrategie I* als Reserve behalten werden, da ihr restlicher Abrechnungszeitraum zu kurz ist.

Auswertung Kriterium 1: Datenverarbeitung

Kriterium 1.1: Grundleistung
$D_g = \sum_{i=1}^{n} t_a(i) \div \sum_{i=1}^{n} t_b(i)$
= 9773 Min ÷ 9817 Min
= 99,55%

Kriterium 1.2: Reserveleistung
$D_r = \sum_{i=1}^{n} t_r(i)$
= 116 Minuten

Auswertung Kriterium 2: Mehrkosten
$K = \sum_{i=1}^{n} t_u(i)$
= 400 Minuten

8.5.3. Intelligente Auslastungszeitraum-Strategie mit Reserve I

Der nachfolgende Abschnitt behandelt die praktische Evaluation der *Intelligente Auslastungszeitraum-Strategie mit Reserve I*. Eine Erklärung der Funktionsweise befindet sich im Abschnitt 5.9. Die theoretische Evaluation der Strategie ist im Abschnitt 6.4.10 nachzulesen.

Durch das verzögerte Starten von Knoten verschiebt sich der Knotenverlauf (Abbildung 8.6) der *intelligenten Auslastungszeitraum-Strategie mit Reserve 1* nach rechts. Es entsteht eine andere Reserveleistung, da sich die Abrechnungszeiträume verschieben. Eine weitere Besonderheit ist die Anzahl der gestarteten Knoten. Durch die Verzögerung und Verschiebung werden Lastspitzen ignoriert und es kann die Reserve genutzt werden. Dadurch werden nur 15 verschiedene Knoten anstelle von 17 gestartet.

Auswertung Kriterium 1: Datenverarbeitung

Kriterium 1.1: Grundleistung
$D_g = \sum_{i=1}^{n} t_a(i) \div \sum_{i=1}^{n} t_b(i)$
= 9563 Min ÷ 9817 Min
= 97,41%

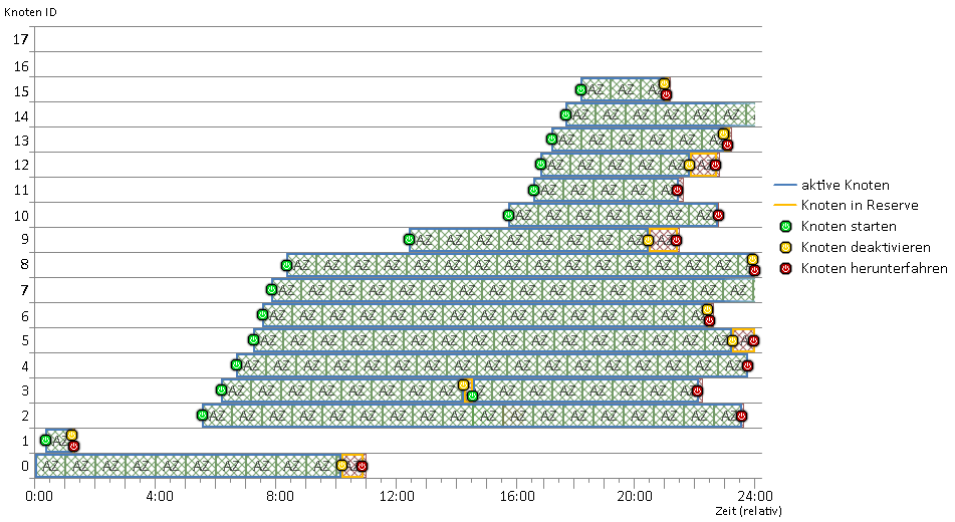

Abbildung 8.6. Knotenverlauf der *intelligenten Auslastungszeitraum-Strategie mit Reserve 1*

Kriterium 1.2: Reserveleistung

$D_r = \sum_{i=1}^{n} t_r(i)$
= 242 Minuten

Auswertung Kriterium 2: Mehrkosten

$K = \sum_{i=1}^{n} t_u(i)$
= 585 Minuten

8.6. Vergleich und Ergebnis

Tabelle 8.1 zeigt die Ergebnisse der praktischen Evaluation in der Reihenfolge ihres Auftretens.

Tabelle 8.1. Ergebnisse der praktischen Evaluation

Strategie	**Grundleistung D_g / Reserveleistung D_r**	**Mehrkosten K**
intelligente Reservestrategie I	99,55% / 116 Min	400 Min
intelligente Leistungsstrategie	99,55% / 0 Min	400 Min
intelligente Auslastungszeitraum-Strategie mit Reserve I	97,41% / 242 Min	585 Min

Es ist zu beobachten, dass durch die (im Gegensatz zur theoretischen Evaluation) hohe Last der Unterschied der Grundleistung aller Strategien minimal geworden ist. Die Reserveleistung der *intelligenten Auslastungszeitraum-Strategie mit Reserve I* ist dabei jedoch mehr als doppelt so hoch wie die der *intelligenten Reservestrategie I*, bei einem 1,5-fachen Anstieg der Mehrkosten.

8.7. Einschränkungen der Validität

Die erzielten Ergebnisse sind nicht allgemeingültig. Im Folgenden werden Aspekte aufgezeigt, die die Validität der Evaluation einschränken.

8.7.1. Simulierte Cloud

VirtuaCloud simuliert eine Cloud, geht jedoch nicht auf alle Besonderheiten ein. Durch die direkte Integration in *SLAstic Lite* entfallen Kommunikationswege und Reaktionszeiten von Hardware. In einer realen Cloud können die Evaluierungen von den in dieser Arbeit erzielten Ergebnissen abweichen oder sich gänzlich unterscheiden.

8.7.2. Workload

Die in der Evaluation erzielten Ergebnisse sind nur für die verwendete Workload Funktion gültig. Andere Lastverläufe könnten die speziellen Eigenschaften mancher Strategien besser nutzen. Dadurch kann eine andere Wertungsreihenfolge im direkten Vergleich entstehen, was zu anderen Ergebnissen führt.

8.7.3. Beschränkung der Kriterien

Die für die praktische Evaluation verwendeten Kriterien sind eine Auswahl an möglichen Merkmalen, die zu vergleichen wären. Unter Berücksichtigung andere Aspekte könnte man zu einem anderem Ergebnis gelangen.

Kapitel 9

Verwandte Arbeiten

Dieses Kapitel stellt wissenschaftliche Arbeiten und Text vor, deren Themengebiet sich mit dem dieser Bachelorarbeit überschneidet oder ihm ähnelt,

9.1. Cloud User-Centric Enhancements of the Simulator CloudSim to Improve Cloud Deployment Option Analysis

Benutzerfokussierte Verbesserungen der Cloud Simulationssoftware *„CloudSim“*
Erläutert die Funktion des Programms und zeigt Verbesserungen auf, die dem Benutzer Abschätzungen ermöglichen, welche Modelle einer Cloud für ihn am geeignetsten ist. Dabei wird auf Kostenmodelle sowie auf die Leistung eingegangen [Fittkau u. a. 2012].

9.2. Search-Based Genetic Optimization for Deployment and Reconfiguration of Software in the Cloud

Softwareportierung in die Cloud anhand von *CDO*s
Für die Portierung von Software in die Cloud müssen Optionen zur Verteilung (*cloud deployment options, CDO*) gegeneinander abgewogen werden. Der Simulator *CDOSim* kann die verschiedenen Faktoren grob gegeneinander abwägen. Es wird versucht, eine Annäherung an das Optimierungsproblem mit dem genetischen Algorithmus *CDOXplorer* zu finden [Frey u. a. 2013].

9.3. Programming Directives for Elastic Computing

Vorgehensweisen zur Aufwandsreduzierung für elastische Cloud Software
Beschreibt Ansätze und Vorgehensweisen, um eine Aufwandsminimierung bei der Programmierung von elastischer Cloud Software zu erreichen. Dafür werden Beispiele mit der eigenen entwickelten Programmiersprache *SYBL (Simple-Yet-Beautiful Language)* vorgestellt und erläutert [Dustdar u. a. 2012].

Kapitel 10

Fazit und Ausblick

Im letzten Kapitel dieser Arbeit werden die Ergebnisse der theoretischen und der praktischen Evaluation zusammengefasst und eine Schlussfolgerung daraus gezogen. Ferner wird ein Ausblick auf Verbesserungen und mögliche zukünftige Arbeiten gegeben.

10.1. Fazit

Bereits die theoretische Evaluation hat gezeigt, dass Strategien, die den Abrechnungszeitraum berücksichtigen, wesentlich kosteneffizienter sind. Im Gegensatz zu gewöhnlichen Strategien ließen sich zum Teil über die Hälfte der Mehrkosten einsparen.

Strategien mit Auslastungszeitraum erschienen in der theoretischen Evaluation Leistungsschwach; die Einsparung bei Lastspitzen schien zu gering. In der Simulation zeigte sich, dass bei größer werdender Last die Leistungsunterschiede zunehmend minimaler werden.

Reservestrategien haben weder Vor- noch Nachteil gegenüber anderen Strategien aufgezeigt. Die Ursache hierfür liegt in der Simulation. Das Starten eines neuen Knotens nahm genau so viel Zeit in Anspruch wie das Reaktivieren eines Knotens. Dies trifft in der Realität jedoch nicht zu.

Wir kommen zu dem Schluss, dass Strategien, die Abrechnungszeiträume beachten und Reserve ermöglichen, am besten für Kieker in der Cloud geeignet sind. Dabei ist es situationsabhängig, ob eine Strategie mit Auslastungszeitraum verwendet werden sollte. Ist die entstehende Last weitgehend unbekannt, ist die *Intelligente Reservestrategie I* zu bevorzugen.

10.2. Ausblick

Die evaluierten Strategien benötigen eine Feinabstimmung für das System, mit dem sie verwendet werden. Zeiträume zum Hoch- und Herunterfahren sowie die Datenübertragung können zu einem Abweichen der Strategie von ihrer eigentlichen Funktionsweise führen.

Um *VirtuaCloud* effektiver zum Testen und Analysieren von Elastizitätsstrategien zu verwenden, könnte man vor dem Start der Simulation einen Multiplikator festlegen, der dem Benutzer ermöglicht zu bestimmen, wie viel Last maximal erzeugt werden soll.

Des Weiteren kann die Auswahl der Workload Kurve durch ein GUI geschehen. Man hätte eine Übersicht über alle vorhanden Funktionen und könnte auch ein Bild der Kurve anzeigen lassen.

Das von *VirtuaCloud* erstellte Protokoll besitzt auch Möglichkeiten zur Verbesserung. Zusätzlich könnte ein Wert angezeigt werden, der angibt, wie viele Knoten bei der aktuellen Last benötigt werden. Nach Abschluss der Simulation könnte noch eine Übersicht erfolgen, welcher Knoten wann wie lange aktiv war.

Literaturverzeichnis

[*Amazon Elastic Compute Cloud*] Amazon Web Services, Inc. Amazon Elastic Compute Cloud. Letzter Zugriff: 01. August 2013. URL: http://aws.amazon.com/de/ec2/. (Siehe Seiten 6 und 11)

[Dustdar u. a. 2012] S. Dustdar, Y. Guo, R. Han, B. Satzger und H.-L. Truong. Programming directives for elastic computing. *IEEE Internet Computing* 16.6 (2012), Seiten 72–77. (Siehe Seite 55)

[Fittkau u. a. 2012] F. Fittkau, S. Frey und W. Hasselbring. Cloud user-centric enhancements of the simulator cloudsim to improve cloud deployment option analysis. In: *Service-Oriented and Cloud Computing*. Lecture Notes in Computer Science. DOI: 10.1007/978-3-642-33427-6_15. Springer Berlin / Heidelberg, 2012, Seiten 200–207. URL: http://eprints.uni-kiel.de/15398/. (Siehe Seite 55)

[Frey u. a. 2013] S. Frey, F. Fittkau und W. Hasselbring. Search-based genetic optimization for deployment and reconfiguration of software in the cloud. In: *35th International Conference on Software Engineering (ICSE 2013)*. IEEE Press, 2013, Seiten 512–521. URL: http://eprints.uni-kiel.de/19237/. (Siehe Seite 55)

[Hof 2006] R. D. Hof. Jeff bezos' risky bet. *Bloomberg Businessweek Magazine* (Nov. 2006). URL: http://www.businessweek.com/stories/2006-11-12/jeff-bezos-risky-bet. (Siehe Seite 1)

[Massow u. a. 2011] R. Massow, A. van Hoorn und W. Hasselbring. Performance simulation of runtime reconfigurable component-based software architectures. In: *Proceedings of the 5th European Conference on Software Architecture (ECSA '11)*. Herausgegeben von I. Crnkovic, V. Gruhn und M. Book. Lecture Notes in Computer Science. Springer-Verlag, 2011, Seiten 43–58. (Siehe Seite 6)

[Mell und Grance 2011] P. Mell und T. Grance. The NIST definition of cloud computing. *Special Publication 800-145* (Sep. 2011). URL: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf. (Siehe Seite 5)

[*OpenStack Homepage*] Rackspace Cloud Computing. OpenStack Homepage. Letzter Zugriff: 01. August 2013. URL: http://www.openstack.org/. (Siehe Seite 6)

[Rohr u. a. 2008] M. Rohr, A. van Hoorn, J. Matevska, N. Sommer, L. Stoever, S. Giesecke und W. Hasselbring. Kieker: continuous monitoring and on demand visualization of java software behavior. In: *Proceedings of the IASTED International Conference on Software Engineering 2008 (SE'08)*. Herausgegeben von C. Pahl. Anaheim, CA, USA: ACTA Press, 2008, Seiten 80–85. (Siehe Seiten 1, 5, 6)

[Van Hoorn u. a. 2009] A. van Hoorn, M. Rohr, W. Hasselbring, J. Waller, J. Ehlers, S. Frey und D. Kieselhorst. Continuous Monitoring of Software Services: Design and Application of the Kieker Framework. Forschungsbericht. Kiel University, 2009. (Siehe Seiten 1 und 5)

[Van Hoorn u. a. 2012] A. van Hoorn, J. Waller und W. Hasselbring. Kieker: a framework for application performance monitoring and dynamic software analysis. In: *Proceedings of the 3rd joint ACM/SPEC International Conference on Performance Engineering (ICPE 2012)*. ACM, 2012, Seiten 247–248. (Siehe Seiten 1 und 5)

Anhang

Dieses Kapitel listet den Inhalt aller angehängter Datenträger.

Daten der CD

Die CD enthält mehrere Ordner, die jeweils bestimmte Daten enthalten.

Images Enthält alle in der Bachelorarbeit verwendeten Grafiken.

Logs Enthält die Protokolle der Simulationen sowie die Änderungen an SLAstic Lite.

PDF Enthält das Proposal, die Präsentation und die Bachelorarbeit als PDF Dateien.

SLAstic Lite Enthält *SLAstic Lite* und alle implementierten Strategien.

Verwandtes Enthält verwandte wissenschaftliche Arbeiten oder Texte.