Energieeffizientes Clustermanagement im Server-Load-Balancing-Bereich, am Fallbeispiel eines Apache-Webserver-Clusters

---

Dissertation

eingereicht von

Diplom-Informatiker Simon Kiertscher

vorgelegt der

Mathematisch-Naturwissenschaftlichen Fakultät
der Universität Potsdam

zur Erlangung des Akademischen Grades

Doktor der Naturwissenschaften

– Dr. rer. nat. –

angefertigt am

Institut für Informatik und Computational Science
der Universität Potsdam

Professur Betriebssysteme und Verteilte Systeme
Betreuung: Prof. Dr. Bettina Schnor

Potsdam, den 3. Oktober 2016

Kiertscher, Simon
kiertscher@cs.uni-potsdam.de
Energieeffizientes Clustermanagement im Server-Load-Balancing-Bereich, am Fallbeispiel eines Apache-Webserver-Clusters
Dissertation, Institut für Informatik und Computational Science
Universität Potsdam, Oktober 2016

Danksagung

Mein größter Dank gilt meiner Frau Juliane, die mich in den letzten Jahren unterstützt und mir viel Kraft gegeben hat. Außerdem danke ich Bettina für eine immer offene Tür, Geduld und viele gute Hinweise und Ratschläge zu dieser Arbeit und den entstandenen Publikationen. Ebenfalls danke ich Klemens für eine immer laufende Hardware, ohne die meine Messungen nicht möglich gewesen wären, und Sabine, Alexandra und Wolfgang für Hilfe bei jeglichen Verwaltungsaufgaben. Ein großer Dank geht auch an alle Kollegen und Mitglieder des Clustertreffs, welche in den letzten Jahren wertvolle Anmerkungen zu der Arbeit geleistet haben. Insbesondere Jörg, Steffen, Sebastian, Claas und Sven sowie den Studierenden Sebastian, Nadine und Jörg für ihre Anteile an dieser Arbeit. Ein besonderer Dank geht an Frau Baller und meine Frau, für unzählige Seiten des Korrekturlesens. Ich danke allen, die meine Entwicklung in den letzten Jahren positiv beeinflusst haben und mit mir kreative Kaffeepausen verbracht haben. Dazu zählen die Kollegen vom PIK Michael, Thomas, Stefan, Klaus und Frank sowie die restlichen Kollegen vom Sonnendeck Marius, Roland, Orkunt, Martin, Javier, Torsten, Benjamin, Johannes, Sebastian, Philipp und Max. Zum Schluss danke ich all meinen Freunden und meiner Familie für ihre Unterstützung und Ablenkung.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendeten Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Potsdam, den 3. Oktober 2016

Simon Kiertscher

Da wies ihn Gott der HERR aus dem Garten Eden, daß er das Feld baute, davon er genommen ist, und trieb Adam aus und lagerte vor den Garten Eden die Cherubim mit dem bloßen, hauenden Schwert, zu bewahren den Weg zu dem Baum des Lebens.

1. Mose 3.23-24

Zusammenfassung

Der Energieverbrauch von Rechenzentren ist in den letzten Jahren auf ein alarmierendes Niveau angestiegen und es ist damit zu rechnen, dass sich dieser Trend fortsetzt. Gleichzeitig wurde in mehreren Studien gezeigt, dass die Auslastung der Rechenzentren nur bei etwa 70 % liegt. Dies entspricht einer verschwendeten Energie, die von mehreren Atomkraftwerken erzeugt werden muss. Einer solchen Verschwendung muss aus ökonomischen und ökologischen Gründen entgegengetreten werden. Da Server noch davon entfernt sind, Energie proportional zu ihrer Auslastung zu verbrauchen (0 % Auslastung ≙ 0 % Verbrauch, 10 % Auslastung ≙ 10 % Verbrauch, usw.), ist die effektivste Methode, um Energie zu sparen, das Abschalten ungenutzter Server.

In dieser Arbeit wird ein Konzept für ein Energiesparsystem im Server-Load-Balancing-Bereich vorgestellt, das die Maschinen eines Clusters dynamisch nach Bedarf an- und abschaltet. Um dieses Konzept zu evaluieren, wird das energieeffiziente Clusterverwaltungssystem *CHERUB* weiterentwickelt. Als *Proof of Concept* wird ein Modul für die Zusammenarbeit von *CHERUB* mit dem Linux-Virtual-Server als Dispatcher und dem Apache-Webserver als Back-End entwickelt und getestet. Diese Kombination wird, stellvertretend für den gesamten Server-Load-Balancing-Bereich, welcher sich durch viele, schnell zu bearbeitende Anfragen auszeichnet, untersucht. Das verwendete Energiesparverfahren arbeitet dabei online und benötigt als Informationen vorab nur die Zeit zum Hochfahren der Maschinen und ein Maß für deren Rechenleistung. Anhand des *CHERUB*-Prototypen werden verschiedene Energiesparalgorithmen untersucht und verglichen, teils durch Messungen in einem realen Clustersetup und teils in einer simulierten Clusterumgebung.

Um die Skalierbarkeit der Strategien nachzuweisen, wird der Simulator *ClusterSim* für beliebig große Apache-Webserver-Cluster entwickelt und validiert. *ClusterSim* arbeitet spurdatenbasiert und ermittelt den Energieverbrauch der simulierten Maschinen abhängig von ihrer Auslastung und auf Basis realer Daten. Um eine realistische Abarbeitung der HTTP-Anfragen zu simulieren, wird in *ClusterSim* sowohl das Linux-Scheduling als auch die Arbeitsweise des Apache Multi-Processing-Modules modelliert. Die Validierung von *ClusterSim* besteht aus mehreren Experimenten, in denen die Simulationsergebnisse mit den korrespondierenden Messergebnissen verglichen werden. Dabei wird festgestellt, dass *ClusterSim* das reale Setup sehr gut nachstellt. Durch eine Schnittstelle zu *CHERUB* ist es mit *ClusterSim* möglich, *CHERUB* in einem simulierten Setup zu testen ohne *CHERUB* selbst simulieren zu müssen.

Mit Hilfe von *ClusterSim* wird eine Parameterstudie durchgeführt. Diese Studie soll zum einen die Skalierbarkeit von Strategie und Software nachweisen und zum anderen den Einfluss von vier verschiedenen Parametern untersuchen. Als Arbeitslast wird ein Lastspitzen-Szenario gewählt, welches eine *Worst-Case*-Situation darstellt. Zur Bewertung der Energiesparstrategie und -Parameter wird die geleistete Dienstqualität, der dafür aufgewandte Energieverbrauch und die Anzahl der Ein/Ausschaltvorgänge herangezogen. Im Gegensatz zu anderen Untersuchungen auf diesem Gebiet, werden die Ergebnisse auch mit einer optimalen Strategie, die ein allwissendes „Orakel" nutzt, verglichen. Die Bewertung der Ergebnisse wird aus der Sicht verschiedener Clusterbetreiber-Strategien durchgeführt, welche die Metriken jeweils unterschiedlich stark gewichten. Die Strategien sind ein Hochleistungsbetrieb (priorisiert Qualität des Dienstes), ein energiesparsamer Betrieb (priorisiert Energieverbrauch), ein ausgewogener Betrieb (priorisiert Qualität des Dienstes und Energieverbrauch) und ein Betrieb mit alter Hardware (priorisiert minimale Anzahl von Ein/Ausschaltvorgängen).

Dabei wird festgestellt, dass vor allem die Zeit für das Hochfahren einer Maschine einen großen Einfluss auf die Ergebnisse hat und schneller bootende Hardware immer bevorzugt eingesetzt werden sollte. Bei den untersuchten Parameterkonfigurationen liefert, mit Ausnahme von einer einzigen Konfiguration, immer die schneller bootende Hardware das bessere Ergebnis. Hardwarehersteller sollten zukünftig Wert darauf legen, ihre Produkte so zu gestalten, dass eine möglichst schnelle Transition von einem sehr tiefen Schlafzustand (bzw. dem Abgeschaltetsein) in einen aktiven Zustand möglich ist. Auf diese Art können alle verwendeten Strategien deutlich optimiert werden.
Als Backup werden Maschinen bezeichnet, die als Leistungsreserve angeschaltet sind, das heißt zusätzlich zu der minimal notwendigen Menge an Maschinen, die zur Bewältigung der aktuellen Last benötigt werden. Es wird gezeigt, dass mit einem genügend großen Backup auch langsamere Hardware noch 10,6 % Energie einsparen und dabei eine Qualität des Dienstes von 92,3 % halten kann. Die Größe des Backups hat insgesamt einen etwa linearen Einfluss auf das Verhältnis zwischen Energieeinsparung und der Qualität des Dienstes. Weiterhin wird nachgewiesen, dass ein aggressives Abschalten im vorliegenden Lastspitzen-Szenario bis zu 12,9 % mehr Energie gegenüber einem langsamen (Stück-für-Stück) Abschalten einsparen kann. Es wird auch gezeigt, dass ein explizites Warten vor einem Bootbefehl überflüssig ist, da die Lastvorhersage ein unnötiges, häufiges An/Abschalten eines Rechners, auch als *State-Flapping* bekannt, bereits verhindert.
Im Rahmen der Parameterstudie wird ein Ranking für die beste Parametrisierung des Systems aufgestellt. Das Ranking wird jeweils für die vier vorgestellten Strategien erstellt. Es zeigt sich, dass *CHERUB* in der Lage ist, bei einem Hochleistungsbetrieb Energieeinsparungen von 13,6 % / 10,6 % (schnell bootende Hardware: 5 Sekunden / langsam bootende Hardware: 3 Minuten) bei gleichzeitiger Qualität des Dienstes von 98,7 % / 92,3 %, zu erzielen. Die Qualität des Dienstes der schnell bootenden Hardware entspricht dabei dem Niveau der optimalen Strategie (98,7 %) ohne den Einsatz von *CHERUB* und zeigt, dass mit guter Hardware eine energieeffiziente Clusterverwaltung ohne Qualitätseinbußen möglich ist. Bei einem besonders preiswerten Betrieb werden Energieeinsparungen von bis zu 32,6 % / 34,3 % bei gleichzeitiger Qualität des Dienstes von 84,6 % / 78 % erzielt. In einem orakelgetriebenen Szenario mit vollständigem Wissen über die Last könnte bei maximaler Qualität des Dienstes 29,9 % / 11 % Energie eingespart werden. Bei einem ausgewogenen Betrieb werden Einsparungen von 28,1 % / 28,7 % bei einer Qualität des Dienstes von 97 % / 84,7 % erreicht und bei der Verwendung von älterer Hardware Einsparungen von 32,6 % / 18,3 % bei einer Qualität des Dienstes von 84,6 % / 85,4 %. Der ausgewogene Betrieb stellt bei den Strategien einen guten Kompromiss zwischen hoher Qualität des Dienstes und hohen Einsparungen dar.
Insgesamt zeigt die Studie, dass eine energieeffiziente Clusterverwaltung auch im Server-Load-Balancing-Bereich mit nur geringen Einbußen bei der Qualität des Dienstes möglich ist und dass die Verwendung von Lastvorhersage mit Hilfe von linearer Regression dabei klassischen, ausschließlich schwellwertbasierten Verfahren überlegen ist.

Inhaltsverzeichnis

1 Einleitung

Rechenzentren mit hunderten von Computern sind aus der modernen Informationsgesellschaft nicht mehr wegzudenken. Sie sind die Grundlage, ohne die es keine Web- oder Cloud-Dienste geben würde. Sie werden für die Wettervorhersage, Simulationen von komplexen physikalischen Abläufen, militärische Simulationen, geheimdienstliche Überwachung, wissenschaftliche Anwendungen aller Art etc. benötigt und auch soziale Medien wie Facebook oder Twitter, Online-Shops wie Amazon oder Suchmaschinen wie Google wären ohne sie undenkbar.
Das *Cluster Computing*, welches sich in den Rechenzentren abspielt, unterteilt sich dabei im Wesentlichen in zwei verschiedene Arten. Das *High-Performance-Computing* (HPC) und das *Server-Load-Balancing* (SLB). Dabei bearbeitet ein HPC-Cluster besonders rechenintensive Probleme, um diese möglichst parallel, in einer akzeptablen Zeit zu lösen (Wettervorhersage etc.) und ein SLB-Cluster bearbeitet tausende von kleinen Anfragen, die innerhalb von Sekunden bearbeitet werden müssen (soziale Medien etc.).
Obwohl der SLB-Bereich viele verschiedene Dienste umfassen kann (Webserver, Domain Name System (DNS), Voice over IP (VoIP) etc.), ähneln sich alle in der SLB-Eigenschaft, viele, wenig rechenintensive Aufgaben bewältigen zu müssen. Aus diesem Grund wird sich diese Arbeit auf das Fallbeispiel eines Apache-Webservers als Dienst beschränken, wobei davon ausgegangen werden kann, dass erzielte Ergebnisse auf alle möglichen SLB-Dienste übertragbar sind.
Ein typisches SLB-Setup ist in Abbildung 1.1 zu sehen. Es zeigt, wie Anfragen aus dem Internet an eine öffentliche virtuelle IP (VIP) gesendet werden, hinter der sich ein *Dispatcher* (engl. Zuteiler) verbirgt. Dieser verteilt die eingehende Last, abhängig von der gewählten Schedulingstrategie, auf die Rechner im Back-End, welche dann die eigentliche Arbeit erledigen.
Die Größe, die solche Cluster erreichen können, ist dabei immens. Für das HPC-Feld erscheint halbjährlich eine Liste, die die 500 aktuell schnellsten Systeme der Welt beinhaltet. Die *Top500* [1] wird anhand des Benchmarks *High Performance Linpack* (HPL) ermittelt, der zufällige, dichte, lineare Gleichungssysteme mit 64-Bit-Arithmetik löst. Platz eins wurde dabei von 2013 bis 2016 von einer Maschine aus China gehalten (Tianhe-2), welche aus 16.000 einzelnen Computern, auch Knoten genannt, besteht. In Betrieb hat Tianhe-2 einen Energiebedarf von 17,8 MW.
Im SLB-Bereich gibt es keine vergleichbare Auflistung von Cluster-Systemen. Dies liegt vor allem daran, dass die Cluster meist profitorientierten Unternehmen gehören und diese die Architektur und den Betrieb ihrer Cluster aus Wettbewerbsgründen geheimhalten. Im Netz finden sich trotzdem Informationen, die darauf schließen lassen, dass hier ähnliche Größenordnungen wie im HPC-Feld vorliegen. Tabelle 1.1 zeigt eine Auswahl von bekannten Zahlen und Schätzungen von großen Unternehmen im SLB-Bereich. Es ist davon auszugehen, dass die Anzahl der Server, die in der Tabelle gelistet sind, sich auf mehrere Rechenzentren aufteilt.
Eine so große Menge an Servern verursacht, wie auch im HPC-Feld, einen hohen Energiebedarf und damit hohe laufende Kosten und schädigt die Umwelt. Während des Betriebs kommen zusätzlich Kosten für die Kühlung und Infrastruktur hinzu, welche häufig noch einmal genauso hoch sind wie die Kosten für den Betrieb der Server selbst [3, 4, 5]. Neben dem Energieverbrauch im Betrieb

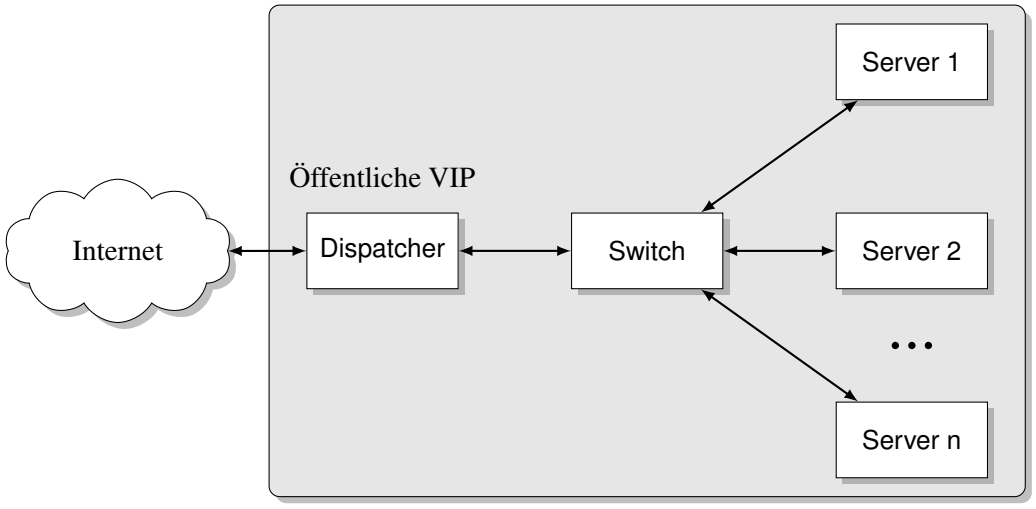


Abbildung 1.1: Typisches Server-Load-Balancing Setup.

Unternehmen	Anzahl der Server	Quelle
Microsoft	> 1 Million	CEO Steve Ballmer (Juli, 2013)
Facebook	*"hundreds of thousands of servers"*	Director, Network Engineering Najam Ahmad (Juni, 2013)
Akamai Technologies	127.000	Aussage des Unternehmens (Juli, 2013)
Intel	75.000	Aussage des Unternehmens (August, 2011)
Google	900.000	Schätzung von Jonathan Koomey anhand des Stromverbrauchs
	45.000	Einzelnes Rechenzentrum, 2005 gebaut
Amazon	40.000 (Web Service EC2)	Analyse von Randy Bias (2009)
eBay	54.011	DSE dashboard (Juli, 2013)

Tabelle 1.1: Unternehmensdaten laut [2].

benötigt auch die Herstellung der Hardware große Mengen an Energie, von der Gewinnung der Rohstoffe, über die Fertigung, bis hin zum Transport.

1.1 Motivation

Eine Studie [6] der University of Stanford hat 2015 eine Arbeit gestützt [7], die im Jahr 2008 vom Uptime Institute und McKinsey&Company veröffentlicht wurde. Diese belegt, dass 30 % aller Server weltweit *„komatös"* sind. Die Autoren meinen damit, dass die Server angeschaltet sind, dadurch Energie verbrauchen, aber nichts tun. Diese Verschwendung lässt sich auf einen Verbrauch von etwa 4 GW beziffern (zum Vergleich, der Verbrauch von Tianhe-2 lag bei 17,8 MW). Laut der Internationalen Atom Energie Behörde (IAEA) generiert der aktuell leistungsfähigste, in Betrieb befindliche Atomkraftwerk-Block der Welt [8] (Charleville, Frankreich) 1,5 GW Energie. Das Kraftwerk selbst besteht aus zwei Blöcken (beide 1,5 GW stark) und ist seit 1996 in Betrieb.

Eine Studie von Barroso und Hölzle (Google) [9] aus dem Jahr 2007 unterstützt ebenfalls die Aussage, dass Ressourcen nicht effizient genutzt werden. Sie belegen durch eine halbjährige Studie in einem 5.000 Server großen Rechenzentrum, dass die durchschnittliche CPU-Auslastung während der Zeit der Studie zum größten Teil zwischen 10 % - 50 % Auslastung lag.
Ineffizienter Betrieb von Ressourcen betrifft nicht nur große Rechenzentren, sondern Clusterbetreiber aller Größenordnungen. Das Institut für Informatik und Computational Science der Universität Potsdam hat bei dem von ihm betriebenen *Zuse*-Cluster ebenfalls beobachtet, dass die Auslastung stark schwankt. Bei *Zuse* handelt es sich um ein Cluster mit 28 homogenen Knoten und ist damit, im Vergleich zu den Top500-Maschinen, ein eher kleines Cluster.
Als großes Ziel wird in [9] das *Energy-Proportional Computing* aufgezeigt, in dem die geleistete Arbeit einer Maschine (bzw. eines Clusters oder eines ganzen Rechenzentrums) im Verhältnis zu ihrem Energieverbrauch steht. Eine Maschine würde dann keinen Strom verbrauchen, wenn sie inaktiv wäre bzw. bei 10 % Last, 10 % Energie verbrauchen, bei 20 % Last, 20 % Energie verbrauchen und so weiter. Tatsächlich sind Maschinen aber weit davon entfernt, energie-proportional zu sein. Laut Autoren ist die CPU momentan die einzige Komponente, die durch DVFS und andere Technologien (siehe Kapitel 2.1, ab Seite 9), sehr nah an dieses Ziel kommt. In modernen Maschinen macht die CPU aber nur etwa 25 % des gesamten Energiebedarfs aus. Andere Komponenten, die maßgeblich am Verbrauch beteiligt sind, sind Lüfter, Netzwerk, Festplatten, Speicher und Netzteil (ineffiziente AC-DC Umwandlung) [10].
Das Problem von ineffizienten Produkten im IT-Bereich geht so weit, dass im Jahr 2013 die EU Verordnung Nr. 617/2013 [11] verabschiedet wurde. Diese beinhaltet Vorgaben für ein *Ökodesign* von Desktop Rechnern, Notebooks, Workstations und kleineren Servern (eine genaue Aufschlüsselung findet sich in [11]). Für diese wird festgelegt, wie hoch ihr Energieverbrauch maximal sein darf. Die Verordnung schreibt ebenfalls die Effizienz von Netzteilen vor. Die Verordnung schätzt auf Basis von Vorstudien, dass mit den beschriebenen einzuhaltenden Grenzwerten, zwischen 2011 und 2020, etwa 93 TWh eingespart werden können. Die Verordnung stellt damit zwar einen wichtigen Schritt in Richtung *Energy-Proportional Computing* dar, sie schließt allerdings explizit Systeme, wie z.B. Blade-Server, aus, welche für den Betrieb in Rechenzentren verwendet werden.
Bis *Energy-Proportional Computing* vollständig umgesetzt ist, müssen andere Wege gefunden werden, um die Verschwendung von Energie deutlich zu reduzieren. In der Literatur gibt es zwei konträre Verfahren, um die Verschwendung zu reduzieren, *slow-and-steady* (engl. langsam aber stetig, siehe z.B. [12, 13, 14]) und *sprint-to-halt* [1] (engl. Sprint zum Stillstand, siehe z.B. [15, 16, 17, 18]).
Slow-and-steady verfolgt den Ansatz, Last so langsam wie möglich auf heruntergetakteter CPU zu rechnen, ohne dabei dessen Frist zu überschreiten. Verringern des Taktes und der Versorgungsspannung hat den Vorteil einen quadratischen Einfluss auf den Energieverbrauch zu haben (mehr dazu in Abschnitt 2.1.4, Seite 12), dieser Zusammenhang soll bei *slow-and-steady* ausgenutzt werden, um Energie zu sparen.
Sprint-to-halt sieht das Gegenteil vor. Die Arbeit soll so schnell wie möglich verrichtet werden, um danach nicht benötigte Komponenten abzuschalten (oder den gesamten Rechner).
Welches der beiden Verfahren tatsächlich einen Nutzen für die Energieeffizienz hat, hängt von der verwendeten Hardware ab und kann laut Miyoshi et al. [19] berechnet werden (mehr dazu in

[1] Es gibt hier viele weitere Bezeichnungen, wie *sprint-to-shutdown*, *race-to-halt*, *sprint-to-idle*, *sprint-to-rest* etc.

Takt	Leistung	Verbrauch	Dauer	Effizienz
2,262 GHz	62,85 GFlops	20 Wh	286,43 s	250,03 MFlops/Watt
1,596 GHz	44,87 GFLops	22 Wh	401,23 s	227,31 MFlops/Watt

Tabelle 1.2: Energieeffizienz einer CPU bei voller und niedriger Leistung.

Abschnitt 2.1.9, Seite 15).

Typische Server haben nicht nur leistungsstarke Multicore-CPUs verbaut, sondern auch größere Mengen an Arbeitsspeicher und Festplattenspeicher, Netzwerkkarten, Netzteile und z.B. Grafikkarten. All diese Komponenten verbrauchen neben der CPU ebenfalls beträchtliche Mengen an Energie. Daher muss bei Servern *sprint-to-halt* genutzt werden um Energie einzusparen, da *slow-and-steady* durch den hohen Energieverbrauch der anderen Komponenten einen negativen Effekt auf den Energieverbrauch hat. Dieses Verhalten hat sich auch bei eigenen Testmessungen an einer Maschine bewahrheitet, welche mit einem Intel Xeon E5520 (Dell PowerEdge R610) und 6 GB RAM ausgerüstet ist. Gemessen wurde dies mit dem *High Performance Linpack*, die Ergebnisse sind in Tabelle 1.2 zu sehen. Wie sich zeigt, benötigt die CPU mit voller Leistung gegenüber der heruntergetakteten CPU nur etwa 70 % der Zeit, um die Berechnungen zu beenden. Die Leistung in MFlops[2] pro Watt zeigt deutlich, dass die Maschine weniger energieeffizient ist, wenn sie heruntergetaktet wird. *Slow-and-steady* benötigt somit nicht nur mehr Energie, sondern auch wesentlich mehr Zeit[3].

Rechner, die gar keine Arbeit zu verrichten haben, haben zusätzlich das Problem, dass sie, abhängig von ihrer Hardwarekonfiguration, auch im inaktiven Zustand, trotz Drosselung, typischerweise bis zu 50 % ihres maximalen Energieverbrauchs aufweisen. Auch dieses Verhalten wurde bereits bei eigener Hardware beobachtet.

Im Rahmen einer Diplomarbeit wurde daher im Jahr 2010 der Daemon *CHERUB* [20, 21, 22] entwickelt, welcher Maschinen in einem HPC-Cluster dynamisch, nach Bedarf an- und abschaltet. *CHERUB* wurde modular entwickelt. Ziel war es, durch verschiedene Module mit verschiedenen Ressourcen-Management-Systemen (RMS) interagieren zu können. So ist ein breiter Einsatz möglich, da pro RMS nur noch ein Modul für die Interaktion mit dem jeweiligen RMS entwickelt werden muss.

Ziel dieser Arbeit ist es, ein effizientes, skalierbares Modul für den SLB-Bereich zu entwickeln. Damit verbunden ist die Analyse der Unterschiede der Bereiche und die damit verbundene Anpassung der Verwaltungsstrategie. Dabei gestaltet sich die Anwendung im SLB-Bereich als weitaus schwieriger im Vergleich zum HPC-Bereich, bei dem für An/Abschaltentscheidungen nur in die Warteschlange der Anstehenden Jobs geschaut werden musste. Im SLB-Bereich müssen nun insbesondere die Auslastung der einzelnen Maschinen sowie die Menge der anstehenden Last genauer berücksichtigt werden.

[2] Flops - floating point operations per second (engl. Gleitkommaoperationen pro Sekunde)

[3] Es kann argumentiert werden, dass HPL eine sehr rechenintensiver Anwendung ist und der Faktor Zeit weniger ins Gewicht fällt, wenn ein anderer Benchmark für den selben Test verwendet werden würde.

1.2 Gliederung

Die restliche Arbeit gliedert sich wie folgt:

Kapitel 2 (Seite 9) befasst sich mit den Grundlagen. Es werden relevante Technologien und Standards vorgestellt, die wesentlich für das Energiemanagement eines Rechners sind. Außerdem wird ebenfalls verwendete Software vorgestellt, darunter auch das bereits erwähnte *CHERUB*. Für ein besseres Verständnis der verwandten Arbeiten und eine Selbsteinordnung werden anschließend grundlegende Eigenschaften von Algorithmen beschrieben.

In Kapitel 3 (Seite 37) werden relevante verwandte Arbeiten beschrieben und klassifiziert.

Nachdem alle Grundlagen und relevante verwandte Arbeiten vorgestellt wurden, widmet sich Kapitel 4 (Seite 58) der Analyse der Anforderungen und Herausforderungen, die ein Energiesparsystem (ESS) wie *CHERUB* im SLB-Feld bewältigen muss, um effizient zu funktionieren. Darauf aufbauend wird ein Konzept und die daraus resultierenden Änderungen an *CHERUB* vorgestellt.

Kapitel 5 (Seite 64) stellt anschließend die Rahmenbedingungen für die ersten Experimente vor. Das verwendete Setup wird vorgestellt und auf seine Leistung hin untersucht. Im selben Rahmen wird ebenfalls ein Vergleich zwischen den beiden Schedulingstrategien Round-Robin und Least-Connection durchgeführt. Anschließend wird die später immer wieder auftretende Arbeitslast ausführlich diskutiert. Die Arbeitslast entspricht einem realen Vorbild und spiegelt eine Lastspitzen-Situation wider. Hier werden ebenfalls die verwendeten Metriken für die Evaluation vorgestellt.

Das anschließende Kapitel 6 (Seite 78) umfasst Experimente, welche unterschiedliche, grundlegende Verfahren untersucht, auf Basis derer *CHERUB* seine Entscheidungen trifft. Unterteilt wird dabei in einfache schwellwertbasierte Ansätze und solche, die Lastvorhersage nutzen. Alle Verfahren wurden in *CHERUB* implementiert und an einem realen System getestet.

Nachdem im vorherigen Kapitel das effizienteste Verfahren ermittelt wurde, dies aber nur in einem kleinen 2-Knoten-Cluster getestet werden konnte, wird nun in Kapitel 7 (Seite 96) anhand von Funktionstests untersucht, ob *CHERUB* auch performant genug ist, um größere Cluster (100+ Knoten) zu verwalten.

Da weder die technischen, noch die finanziellen Mittel zur Verfügung stehen, um *CHERUB* in einem realen Cluster mit mindestens 100 Knoten zu testen, wird in Kapitel 8 (Seite 107) der selbst entwickelte Simulator *ClusterSim* vorgestellt und validiert. *ClusterSim* kann beliebig große Apache-Webserver-Cluster simulieren. Die Simulation stellt dabei das Verhalten des Apache *MPM prefork* (siehe Kapitel 2.2.2.1, Seite 21) Moduls auf Anwendungsebene nach und approximiert auf Betriebssystemebene den $\mathcal{O}(1)$ Scheduler. Er berechnet dabei die verbrauchte Energie anhand der simulierten Auslastung und einem Energieprofil, welches dem Durchschnitt eines realen Datensatzes von 4-Kern-Maschinen entspricht. Durch eine Schnittstelle zu *CHERUB*, kann dieser so mit dem Simulator verbunden werden, dass er sich verhält, als würde er das simulierte Cluster real verwalten.

Kapitel 9 (Seite 134) untersucht die Skalierung der Strategie von *CHERUB*. Mit der Hilfe von *ClusterSim* wird *CHEURB* in einem 100-Knoten großen, virtuellen Cluster getestet. Die Studie komplettiert zum einen die Funktionstests, da hier *CHERUB* noch einmal als Ganzes zum Einsatz kommt, und zum anderen zeigt sie, welche Parameter das Verhalten von *CHERUB* beeinflussen. Eine Analyse der Ergebnisse zeigt, wie die jeweiligen Parameter in Abhängigkeit von Betriebsart und Hardware eines Clusters gesetzt werden sollten, bzw. welche Resultate zu erwarten sind.

In Kapitel 10 (Seite 154) werden abschließend die Grenzen und möglichen Erweiterungen für beispielsweise heterogene Cluster oder Clouds diskutiert und ein abschließendes Fazit gezogen.

1.3 Forschungsbeiträge

Diese Dissertation leistet die folgenden Beiträge:

- Eine umfangreiche Übersicht und Bewertung der Arbeiten im Bereich der energieeffizienten Clusterverwaltung wurde erstellt.

- Verschiedene Methoden zur Ermittlung und Vorhersage von Last im SLB-Bereich und deren Parametrisierung wurden mit Hilfe von Messungen an einem realen Setup auf ihre Effizienz hin untersucht. Dabei wurde der Vorteil von Lastvorhersage mit Hilfe von linearer Regression gegenüber klassischen, ausschließlich schwellwertbasierten Verfahren nachgewiesen.

- Es wurde ein energieeffizientes Clusterverwaltungs-Modul für *CHERUB* entwickelt, welches open source ist und im Zusammenhang mit dem Linux-Virtual-Server (Dispatcher) und dem Apache-Webserver (Back-End) arbeitet. Diese Kombination dient als Fallbeispiel für ein typisches SLB-Setup. Als *Proof of Concept* konnte an ihm gezeigt werden, dass das energieeffiziente Verwalten von SLB-Clustern möglich ist und hohe Energieeinsparungen bei einer hohen Qualität des Dienstes erzielt werden können.

- Ein umfangreicher Simulator (*ClusterSim*) für beliebig große Apache-Webserver wurde entwickelt und validiert. *ClusterSim* arbeitet spurdatenbasiert und ermittelt den Energieverbrauch der simulierten Maschinen abhängig von ihrer Auslastung und auf Basis realer Daten. Um eine realistische Abarbeitung der HTTP-Anfragen zu simulieren, wird in *ClusterSim* sowohl das Linux-Scheduling als auch die Arbeitsweise des Apache Multi-Processing-Modules modelliert. Die Validierung von *ClusterSim* bestand aus mehreren Experimenten, in denen die Simulationsergebnisse mit den korrespondierenden Messergebnissen verglichen wurden. Dabei wurde festgestellt, dass *ClusterSim* das reale Setup sehr gut nachstellt. Durch eine Schnittstelle zu *CHERUB* ist es mit *ClusterSim* möglich, *CHERUB* in einem simulierten Setup zu testen ohne *CHERUB* selbst simulieren zu müssen.

- Anhand einer Parameterstudie wurde mit Hilfe von *ClusterSim* der Einfluss von verschiedenen Parametern auf die Strategie von *CHERUB* und dessen Resultate untersucht. Dabei wurde festgestellt, dass vor allem die Zeit für das Hochfahren einer Maschine einen großen Einfluss auf die Ergebnisse hat und schneller bootende Hardware immer bevorzugt eingesetzt werden sollte. Es konnte ebenfalls gezeigt werden, dass mit einem genügend großen

Backup auch langsamere Hardware noch 10,6 % Energie einsparen und dabei eine Qualität des Dienstes von 92,3 % halten kann. Die Größe des Backups hat insgesamt einen etwa linearen Einfluss auf das Verhältnis zwischen Energieeinsparung und der Qualität des Dienstes. Weiterhin wurde nachgewiesen, dass ein aggressives Abschalten im vorliegenden Lastspitzen-Szenario bis zu 12,9 % mehr Energie gegenüber einem langsamen (Stück-für-Stück) Abschalten einsparen kann. Es wurde auch gezeigt, dass ein explizites Warten vor einem Bootbefehl überflüssig ist, da die Lastvorhersage ein unnötiges, häufiges An/Abschalten eines Rechners, auch als *State-Flapping* bekannt, bereits verhindert.

- Es wurden Rankings für die beste Parametrisierung des Systems erstellt. Das Ranking wurde jeweils für vier verschiedene Strategien erstellt, die ein Clusterbetreiber verfolgen könnte. Darunter die beste Parametrisierung für einen Hochleistungsbetrieb, einen energiesparsamen Betrieb, einen ausgewogenen [4] Betrieb und einen Betrieb mit alter Hardware. Es konnte gezeigt werden, dass für den Hochleistungsbetrieb möglichst viel Backup, ein langsames Abschalten und möglichst schnelles Anschalten verwendet werden sollten. Für einen preiswerten Betrieb sollte möglichst kein Backup verwendet und zusätzlich gewartet werden, bevor Hardware angeschaltet wird. Bei einem ausgewogenen Betrieb sollte ein kleineres Backup Verwendung finden, aber Hardware sollte immer so schnell wie möglich wieder abgeschaltet werden. Bei Verwendung von älterer Hardware sollte ebenfalls auf Backup verzichtet und Hardware nur langsam an- und abgeschaltet werden.

- Anhand von Messungen wurde gezeigt, dass die Performance der verschiedenen Prozesserzeugungsfunktionen unter Python alle nahezu identisch sind. Die schnellste Variante ist die *fork+execl* Kombination.

- Ebenfalls durch Messungen gezeigt wurde, dass das Scheduling-Verfahren Least-Connection gegenüber Round-Robin eine gleichmäßigere Verteilung der Last erzielt, wobei die verwendete Last aus verschiedensten HTTP-Anfragen bestand.

- Mit Hilfe einer Sourcecode-Analyse und Messungen konnte nachgewiesen werden, dass das Netzwerk-Analyse-Werkzeug `tcpdump` bei der Erkennung von HTTP *GET-Requests* auf einem unter Volllast arbeitenden `nginx` Webserver nur 0,004 % der Anfragen verwirft. Damit ist gezeigt, dass `tcpdump` als webserverunabhängige Lösung zur Lasterkennung eingesetzt werden kann.

- Anhand von Messungen wurde gezeigt, dass die Performance von IPMI-1.5 gegenüber IPMI-2.0 etwa 30 % höher ist.

[4] Ausgewogen hinsichtlich Leistung und Energieverbrauch.

1.4 Publikationen, basierend auf dieser Arbeit

Die wichtigsten wissenschaftlichen Erkenntnisse dieser Dissertation wurden bereits in den folgenden Arbeiten zusammengefasst und publiziert:

1. Inhalte aus Kapitel 5 und 6
   Simon Kiertscher und Bettina Schnor
   Energy Aware Resource Management for Clusters of Web Servers
   Erschienen im Konferenzband der 4. *IEEE International Conference on Green Computing and Communications (GreenCom)*, Beijing, China, 2013

2. Inhalte aus Kapitel 7 und 8
   Simon Kiertscher und Bettina Schnor
   Scalability Evaluation of an Energy-Aware Resource Management System for Clusters of Web Servers
   Erschienen im Konferenzband des 18. *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, Chicago, USA, 2015

3. Inhalte aus Kapitel 9
   Simon Kiertscher und Bettina Schnor
   Optimizing Energy Efficiency and Quality of Service in Large Scale Web Server Environments
   Zur Veröffentlichung akzeptiert für *Fachgruppe Betriebssysteme: Herbsttreffen 2016*, Augsburg, Deutschland, 2016

4. Inhalte aus Kapitel 9
   Simon Kiertscher und Bettina Schnor
   Optimizing Energy Efficiency and Quality of Service in Large Scale Web Server Environments
   Eingereicht bei der 7. *IEEE International Conference on Green Computing and Communications (GreenCom)*, Chengdu, China, 2016

2 Grundlagen

Dieses Kapitel vermittelt Grundlagen, die für das weitere Verständnis dieser Arbeit wichtig sind. Es stellt ebenfalls Technologien und Standards vor, die nicht direkt in dieser Arbeit verwendet werden, die aber zusätzlich eingesetzt werden können, um ein System noch energieeffizienter zu gestalten. Das Kapitel stellt außerdem Software vor welche benötigt wird um ein energieeffizientes Webserver Cluster aufzubauen und zu testen. Darunter fällt das RMS *Linux Virtual Server*, der Apache-Webserver, das ESS *CHERUB*, das Überwachungs-Werkzeug `tcpdump`, sowie drei verwendete Werkzeuge zur Lasterzeugung. Es beschreibt die Funktionsweise sowie die wichtigsten Parameter der jeweiligen Software und deren Einfluss. Viele der Parameter werden in den späteren Experimenten immer wieder untersucht und angepasst und sind deshalb ausführlich beschrieben. Abschließend erklärt das Kapitel noch Grundlagen zu Eigenschaften von Algorithmen, welche für das Verständnis des Kapitels über verwandte Arbeiten hilfreich sind.

2.1 Relevante Standards und Technologien

Energieverbrauch ist nicht nur für die in der Einleitung erwähnten Cluster ein Problem, sondern auch für mobile Geräte. Bei ihnen sind weniger die Kosten der Energie das Problem, sondern die Nutzungsdauer des Gerätes. Daher sind Hardwarehersteller immer daran interessiert, nicht nur die Leistung ihrer Komponenten zu verbessern, sondern auch ihre Energieeffizienz zu erhöhen. Mit diesem Ziel wurden verschiedenste Technologien und Standards entwickelt, die im folgenden vorgestellt werden.

2.1.1 Advanced Configuration & Power Interface (ACPI)

Beim ACPI [24, 23] handelt es sich um einen offenen Industriestandard, der beschreibt, welche Mechanismen und Strukturen notwendig sind, um Komponenten eines Computers vom Betriebssystem aus zu erkennen, zu konfigurieren und zu verwalten. Die federführenden Konzerne hinter ACPI sind Hewlett-Packard, Intel, Microsoft, Phoenix Technologies und Toshiba. Der für den Energieverbrauch wichtigste Aspekt von ACPI ist die Definition der verschiedenen Zustände von Rechnern und Komponenten. Es werden folgende globalen Zustände definiert:

G3 Mechanical Off: In diesem Zustand ist das Gerät vollständig abgeschaltet. Es fließt keinerlei Strom in dem Gerät und es verbraucht daher auch keinen Strom. Um das Gerät betriebsbereit zu machen, muss es vollständig neu gestartet werden. In diesem Zustand kann das Gerät gewartet werden (z.B. defekte Hardware austauschen).

G2 Soft Off: Das Gerät ist abgeschaltet, aber nicht vom Strom getrennt. Um das Gerät betriebsbereit zu machen, muss es vollständig neu gestartet werden.

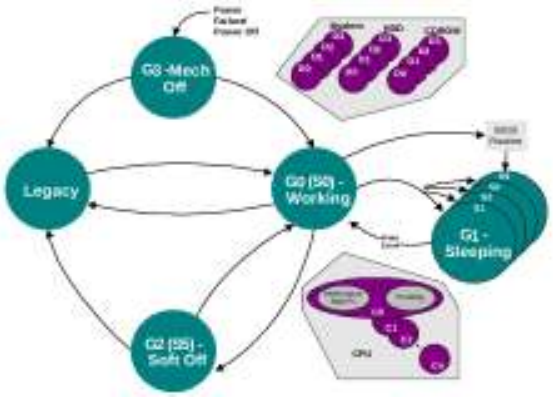


Abbildung 2.1: Definierte ACPI Zustände. Entnommen aus [23].

G1 Sleeping: In diesem Zustand erscheint das Gerät dem Nutzer als abgeschaltet. Es verbraucht ein gewisses Maß an Strom, abhängig von den Komponenten, die noch versorgt werden. Um das Gerät betriebsbereit zu machen, muss das Betriebssystem nicht neu gestartet werden, da große Teile des Zustands gespeichert sind.

G0 Working: In diesem Zustand ist das Gerät voll einsatzbereit. Es können eventuell verschiedene Performance-Profile gewählt werden, in denen Leistung gegen erhöhten Energieverbrauch und umgekehrt getauscht wird.

Zusätzlich werden für einzelne Komponenten eines Rechners sogenannte *Device Power States* definiert. Diese entsprechen im Grunde den globalen Zuständen und werden hier nicht weiter erläutert.
ACPI nimmt eine weitere Unterteilung der globalen Zustände vor, um den verschiedenen Formen der energiesparenden Zustände besser gerecht zu werden. Es unterteilt daher in folgende *Sleep States* (engl. Schlafzustände):

S1: Ein sehr geringer Schlafzustand, in dem selbst Systemkontext, wie CPU-Cache, erhalten bleibt. Eine Rückkehr zu voller Betriebsbereitschaft dauert nur wenige Zyklen.

S2: Wie S1, aber es bleibt kein Systemkontext erhalten. Eine Rückkehr zu voller Betriebsbereitschaft dauert ebenfalls nur wenige Zyklen.

S3 Suspend-to-RAM (STR): In diesem Zustand wird sämtlicher Systemkontext im RAM gespeichert und dort gehalten. Andere Komponenten werden abgeschaltet, wodurch bereits erhebliche Einsparungen stattfinden. Um das System wieder betriebsbereit zu machen, muss der Systemkontext aus dem RAM gelesen werden. Dies dauert erfahrungsgemäß wenige Sekunden.

S4 Suspend-to-DISC (STD): Im Gegensatz zu STR wird hier der Systemkontext auf der Festplatte gesichert. Dadurch wird der Stromverbrauch auf ein Minimum reduziert und entspricht dem Verbrauch von G2. Das Schreiben und Lesen des Systemkontextes dauert länger als bei STR und demzufolge dauert auch das Erreichen und Verlassen des Zustands länger als bei S3

S5 Soft Off: Entspricht G2.

Der ACPI Standard unterteilt G0 weiter in vier verschiedene *Processor Power States* (C0 - C3) und C0 wiederum in beliebig viele *Device and Processor Performance States* (P0 - Pn). Dabei wird empfohlen, nicht mehr als 16 Zustände zu definieren. Grundlegend definieren diese Zustände feingranular die Unterschiede zwischen den einzelnen leichten Schlafzuständen. Dabei gilt, je tiefer der Schlafzustand, desto mehr Energie wird gespart, aber um so länger benötigt die Komponente wieder, um ihre volle Leistung zu erreichen. Sowohl die C- als auch P-Zustände sind für diese Arbeit nicht weiter von Interesse.
Abbildung 2.1 zeigt das Zusammenspiel der verschiedenen, in ACPI spezifizierten, Zustände. Für die Zwecke dieser Arbeit sind die G- und S-Zustände ausreichend und besonders *S3 - STR* wird später noch, mit Hilfe des Simulators *ClusterSim* genauer untersucht (siehe Kapitel 9, Seite 134).

2.1.2 Intelligent Platform Management Interface (IPMI)

Das IPMI [25] ist ebenfalls ein Industriestandard. Dieser wird von Hewlett-Packard, Intel, NEC und Dell spezifiziert. Er beschäftigt sich mit Funktionen zum Überwachen und Steuern von Rechnern, ohne dass diese angeschaltet sein müssen. Funktionen, die IPMI unterstützt, sind z.B. das Auslesen von Sensordaten (Temperatur, Lüfter, Spannung etc.), das Protokollieren von aufgetretenen Fehlern oder das An- und Abschalten des Rechners. Ermöglicht wird dies vom *Baseboard Management Controller* (BMC). Beim BMC handelt es sich um einen Chip, der mit allen wichtigen Sensoren und Komponenten des Rechners verbunden ist und bereits durch die Versorgung durch die Grundspannung, die vorherrscht, wenn ein Rechner am Stromnetz angeschlossen ist, aktiv ist und arbeitet. Durch diese aktive Komponente können die entsprechenden Befehle umgesetzt werden, auch wenn der Rechner selbst noch abgeschaltet ist.
CHERUB verwendet IMPI mit Hilfe des Kommandozeilen-Werkzeugs `ipmitool` [26], um Rechner zu starten oder herunterzufahren. Verfügt ein Rechner nicht über IPMI kann diesem alternativ zum Herunterfahren per SSH der standardmäßige `shutdown` Befehl geschickt werden. Zum Starten eines solchen Rechners kann auf die *Wake on Lan* Technologie zurückgegriffen werden.

2.1.3 Wake on Lan (WOL)

Wake on Lan [27] ist ein Standard, der von AMD und Hewlett-Packard entwickelt wurde. Dieser beschreibt, wie ein Rechner durch ein sogenanntes *Magic Packet* (engl. magisches Paket) zum Starten gebracht wird bzw. wie mit solch einem Paket verfahren werden soll (z.B. beim Routing). Die Idee ist dabei, dass die Netzwerkkarte eines abgeschalteten Rechners weiter Netzwerkpakete verarbeitet und dabei nach einem bestimmten Muster innerhalb der Pakete sucht. Wird dieses Muster entdeckt, schaltet sich der Rechner selbst an. Das Muster, welches gefordert wird, ist in Abbildung 2.2 zu sehen und besteht aus Zieladresse (`DESTINATION`), Quelladresse (`SOURCE`), sonstigem Inhalt (`MISC`), gefolgt von einer Marke, welche durch die sechsfache Wiederholung des Hexadezimalwertes `FF` dargestellt wird. Nach dieser Marke muss die MAC-Adresse des angesprochenen Rechners 16 Mal wiederholt werden. Das Paket kann danach erneut mit beliebigem Inhalt (`MISC`) gefüllt werden und wird mit einer Prüfsumme (`CRC`, für cyclic redundancy check) abgeschlossen.
CHERUB kann, als Alternative zu IPMI, auch WOL verwenden, um abgeschaltete Rechner wieder zu aktivieren.

```
DESTINATION SOURCE MISC FF FF FF FF FF FF 11
22 33 44 55 66 11 22 33 44 55 66 11 22 33 44
55 66 11 22 33 44 55 66 11 22 33 44 55 66 11
22 33 44 55 66 11 22 33 44 55 66 11 22 33 44
55 66 11 22 33 44 55 66 11 22 33 44 55 66 11
22 33 44 55 66 11 22 33 44 55 66 11 22 33 44
55 66 11 22 33 44 55 66 11 22 33 44 55 66 11
22 33 44 55 66 MISC CRC
```

Abbildung 2.2: Das Format eines *Magic Packet* für das Starten eines Rechners mit der MAC-Adresse 11:22:33:44:55:66. Entnommen aus [27].

2.1.4 Dynamic Voltage and Frequence Scaling (DVFS)

DVFS bezeichnet allgemein die Möglichkeit, die Spannung (voltage) und die Taktfrequenz (frequence) einer Komponente (meist CPU) dynamisch verändern zu können. Im Normalfall ist, wenn von DVFS gesprochen wird, das Senken von Spannung und Frequenz gemeint. Es gibt ebenfalls Systeme, die nur eines von beidem verändern können, diese werden dann entsprechend als Dynamic Voltage Scaling (DVS) oder als Dynamic Frequence Scaling (DFS) bezeichnet. Da bei DVFS beides verändert werden kann, ist DVFS flexibler und häufiger in modernen CPUs anzutreffen.

Untertakten/Untervolten: DVFS ist wohl die am meisten verbreitetste Technologie zum Energiesparen und wurde ursprünglich für mobile Geräte entwickelt, um diese länger im Akkubetrieb nutzen zu können. DVFS kann aber auch in nicht mobilen Bereichen genutzt werden. Die Technologie macht sich zu Nutze, dass die CPU immer wieder Leerlaufzeiten ausgesetzt ist. Dies kann z.B. durch speicherintensive Anwendungen der Fall sein oder wenn viele Nutzerinteraktionen stattfinden (Office Anwendungen, Surfen im Internet etc.) oder der Rechner überhaupt nicht verwendet wird. In diesen Leerlaufzeiten wird sowohl die Taktfrequenz als auch die Spannung reduziert, wodurch die CPU weniger Energie verbraucht. Spannung und Frequenz müssen in einem gewissen Verhältnis stehen, damit ein Rechner funktionsfähig bleibt. Senkt man beispielsweise die Spannung bei fester Frequenz zu stark, kann es zu Rechenfehlern oder Abstürzen kommen, da die Transistoren eventuell nicht mehr zuverlässig schalten.
Der Verbrauch von auf CMOS basierenden CPUs wird allgemein hin mit Formel 2.1 berechnet.

$$P = CfV^2 + P_{static} \tag{2.1}$$

Dabei entspricht C der Kapazität der Transistoren, f der Taktfrequenz, V der Spannung und P_{static} dem statischen Anteil. Da die Spannung quadratisch in die Gleichung eingeht, lohnt es sich aus energetischer Sicht, vor allem die Spannung zu senken. Mit Reduzierung der Frequenz ist nur eine lineare Einsparung möglich.
Laut Le Sueur und Heiser [28] ist der Nutzen von DVFS in modernen Systemen allerdings in Frage zu stellen. Durch die sehr kleine Bauweise moderner Chips (kleiner *nm*-Bereich) wird der statische Anteil am Verbrauch, durch z.B. Verluststrom, größer. Durch sogenanntes *prefetching* (engl. vorzeitiges Abrufen) laden moderne CPUs bereits mehr Daten in ihren Cache als notwendig. Dadurch kann in den meisten Fällen das Warten auf spätere Speicherzugriffe verringert werden

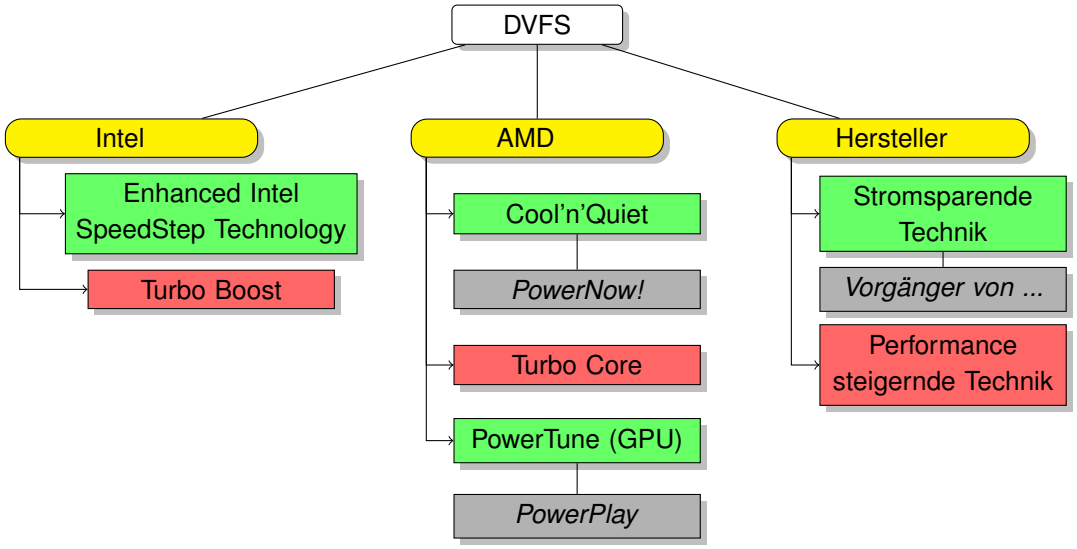


Abbildung 2.3: Übersicht der, auf DVFS basierenden, Technologien.

und DVFS kann diese Wartezeit ebenfalls nicht mehr ausnutzen. Auch die im ACPI-Standard festgelegten Schlafzustände sorgen dafür, dass DVFS keine weiteren Vorteile mehr bringt, wenn diese bereits aktiv sind.

Die Implementierung von DVFS wird von verschiedenen Herstellern verschieden bezeichnet. Intel bezeichnet seine Implementierung als *Enhanced Intel SpeedStep Technology* (EIST) [29] und AMD's Bezeichnung ist *AMD PowerNow! Technology* [30] bzw. dessen Weiterentwicklungen namens *Cool'n'Quiet* [31]. Bei GPUs bezeichnet AMD die Technik als *AMD PowerTune Technology* [32], welche der Nachfolger von *AMD PowerPlay* [33] ist.

Übertakten/Übervolten: Da sich mit DVFS die Spannung bzw. die Frequenz verändern lässt, kann die Technik auch verwendet werden, um die Spannung und die Frequenz zu erhöhen. Dadurch kann die Leistung der Kerne erhöht werden. Wie eingangs erklärt, erhöht sich damit allerdings auch der Energiebedarf ($P = CfV^2 + P_{static}$) und damit auch die thermale Emission. Beide Faktoren machen das Übertakten unattraktiv. Wenn in einem Mehrkernsystem allerdings zeitweise weniger Kerne genutzt werden, als zur Verfügung stehen, kann die damit eingesparte Energie und Fläche (die sich nicht selbst erwärmt, da die entsprechenden Kerne nichts tun) ausgenutzt werden, um die arbeitenden Kerne zu übertakten, ohne dass es zu thermalen Problemen kommt. Durch dieses Vorgehen wird die Energie, die durch die Anzahl nicht arbeitender Kerne potenziell eingespart werden könnte, in Leistung umgesetzt.

Die Umsetzung des beschriebenen Vorgehens wird von Intel als *Turbo Boost* [34] und von AMD als *Turbo Core* [35] bezeichnet.

Abbildung 2.3 gibt eine Übersicht zu den genannten Technologien, welche sich zwar im Detail unterscheiden (z.B. Anzahl der P-States, welche Caches abgeschaltet werden, Nutzung für CPU oder GPU etc.) aber im Grunde alle DVFS umsetzen.

2.1.5 Clock-Throttling

Mit dem *Clock-Throttling* (engl. etwa *Takt Drosselung*) kann, ähnlich wie bei DFS, die Leistung der CPU gedrosselt werden. Im Gegensatz zum DFS wird allerdings nicht die Frequenz des taktgebenden Signals verringert. Bei dieser Technik wird das taktgebende Signal nur in periodischen Abständen an die CPU weitergeleitet (z.B. nur jeder vierte Takt). Wie beim DFS reduziert sich damit die Anzahl der Takt-Signale pro Zeiteinheit, die die CPU erreichen. Dadurch reduziert sich ebenfalls der Energieverbrauch und die thermale Emission.

2.1.6 Clock-Gating

Eine auf tiefster technischer Ebene implementierbare Methode zum Energiesparen stellt das *Clock-Gating* (engl. etwa *getaktete Torschaltung*) dar. Bei dieser Methode können Teile eines Schaltkreises selektiv verwendet werden, indem ihnen eine Torschaltung vorgeschaltet wird. Diese können verhindern, dass der dahinter liegende Teil der Schaltung ein Takt-Signal erhält und daher nicht schaltet. Durch das Verhindern des Schaltens kommt es zu keiner Leistungsaufnahme, wodurch Strom eingespart wird. [36]

2.1.7 Power-Gating

Power-Gating ist dem *Clock-Gating* sehr ähnlich. Im Gegensatz zum *Clock-Gating* werden hier allerdings ganze Teile eines Schaltkreises vollständig vom Strom getrennt (inklusive der Vorspannung/Bias). Dies hat, im Gegensatz zum *Clock-Gating*, größeren Einfluss auf das Design eines Chips, aber umgeht dafür die Verlustleistung, die beim *Clock-Gating* immer noch vorliegt. [37]

2.1.8 Linux Governors

Der Linux-Kernel bietet dem Nutzer die Möglichkeit, die Frequenz des genutzten Rechners selber zu bestimmen, indem DVFS verwendet wird. Zum Einsatz kommen dafür sogenannte *Governors*, von denen es fünf Stück gibt:

1. `Performance`: Verwendet die höchste mögliche Frequenz.
2. `Powersave`: Verwendet die niedrigste mögliche Frequenz.
3. `Userspace`: Verwendet eine vom Nutzer eingestellte Frequenz.
4. `Ondemand`: Die verwendete Frequenz wird sofort der aktuellen Last angepasst.
5. `Conservative`: Wie *Ondemand*, aber ändert die Frequenz nur schrittweise.

Der aktuell verwendete *Governor* findet sich in der Datei `/sys/devices/system/cpu/cpu0/cpufreq/scaling_governor` wieder und kann über diese auch verändert werden. Weitere Details finden sich in [38, 39].

2.1.9 Zusammenfassung und Diskussion: Technologien und Standards

Von den vorgestellten Technologien und Standards wird im Verlauf der Arbeit nur IPMI und ACPI aktiv verwendet. IPMI wird verwendet, um die Maschinen innerhalb eines Clusters an- und abzuschalten und ACPI wird insofern verwendet, als dass in den späteren Simulationen Maschinen verwendet werden, welche STR einsetzen. Die restlichen vorgestellten Techniken sind trotzdem nützlich, da sie, wie bereits in der Einführung erwähnt, zusätzlich zu einem ESS eingesetzt werden können. Sie können vor allem in Situationen mit geringerer Last verwendet werden, während das ESS noch keine Abschaltentscheidung getroffen hat.

Eine Arbeit von Miyoshi et al. [1] [19] untersucht, unter welchen Umständen sich das Reduzieren der CPU-Leistung mit den vorgestellten Techniken überhaupt lohnt. Dafür schlagen sie das Konzept der *critical power slope* (engl. etwa *kritische Energie-Steigung*) vor. Dabei handelt es sich um eine Funktion die angibt, welches Energieprofil eine Maschine aufweisen müsste, damit es unerheblich ist, welche Frequenz bzw. Spannung zum Berechnen einer Last verwendet wird, die Berechnung der Last würde immer gleich viel Energie benötigen [2]. Die *critical power slope* ergibt sich aus dem Verhältnis zwischen dem Energieverbrauch bei geringster Frequenz, abzüglich dem Energieverbrauch des Schlafzustandes und der Leistung bei geringster Frequenz (Formel 2.2).

$$m_{critical} = \frac{P_{f_{min}} - P_{idle}}{f_{min}} \tag{2.2}$$

Der tatsächliche Anstieg der Maschine kann durch das Verhältnis zwischen maximalem Energieverbrauch, abzüglich des Energieverbrauchs bei geringster Frequenz, gegenüber der maximalen Leistung, abzüglich der minimalen Leistung, berechnet werden (Formel 2.3).

$$m = \frac{P_f - P_{f_{min}}}{f - f_{min}} \tag{2.3}$$

Liegt der tatsächliche Anstieg (m) der verwendeten Maschine unter der berechneten *critical power slope* ($m_{critical}$) ist es, aus energetischer Sicht, sinnvoll, die Last möglichst schnell zu berechnen und danach in einen Schlafzustand zu wechseln. Liegt der Anstieg hingegen über der *critical power slope*, sollte die Last möglichst langsam berechnet werden. Da das Energieprofil bei Systemen mit DVFS oder DVS nicht linear, sondern quadratisch verläuft, müssen Formel 2.2 und 2.3 für diesen Fall angepasst werden. Der Anstieg muss in diesen Fällen pro Abschnitt zwischen den einzelnen möglichen Frequenzen angegeben werden. Durch den quadratischen Verlauf kann die energetisch optimale Frequenz auch eine Frequenz innerhalb des möglichen Frequenzspektrums sein und nicht nur die minimal oder maximal mögliche Frequenz. Entscheidend bleibt das Verhältnis zwischen Energieverbrauch in aktivem Zustand und Energieverbrauch im Schlafzustand.

[1] Mitautor ist u.a. Charles Lefurgy, von dessen mit IBM verbundener Arbeitsgruppe noch weitere Arbeiten in Kapitel 3 (Seite 37) vorgestellt werden.

[2] Bei schnelleren Frequenzen wird immer angenommen, dass der Rechner danach noch in einem Schlafzustand weiter Energie verbraucht, bis so viel Zeit vergangen ist, wie es benötigt, bis die Arbeit auch mit der zu vergleichenden Frequenz berechnet ist. Es wird angenommen, dass die niedrigste benötigte Frequenz, um innerhalb der erforderlichen Fristen zu bleiben, bekannt ist und mindestens diese Frequenz genutzt wird.

2.2 Verwendete Software

Diese Arbeit macht Gebrauch von bestehender Software, die hier erklärt werden soll, so dass ein weiteres Verständnis der Arbeit möglich ist. Vorgestellt werden das RMS/der Lastverteiler Linux Virtual Server, der Apache-Webserver, das ESS *CHERUB* (welches Kern dieser Arbeit ist), das Netzwerküberwachungswerkzeug `tcpdump`, das Konzept der TCP-Backlog-Queue aus dem Linux Kernel und abschließend drei verwendete Werkzeuge zur Lasterzeugung.

2.2.1 Linux Virtual Server

Bei dem Linux Virtual Server (LVS) Projekt [40, 41, 42, 43], handelt es sich um ein freies Software-Projekt für virtuelle Server unter Linux. Der virtuelle Server erscheint dabei dem Nutzer wie eine einzelne, leistungsstarke Maschine, obwohl sie aus beliebig vielen Maschinen bestehen kann. Um dies zu realisieren, verwendet LVS eine Maschine als Lastverteiler, der als *Dispatcher*, *LinuxDirector* oder *load balancer* bezeichnet wird. Dieser fungiert als *Front-End* und ist mit dem Internet verbunden. Der *Dispatcher* nimmt eingehende Anfragen entgegen und verteilt diese an Maschinen, welche als sogenannte *Real Server* im *Back-End* über LAN oder WAN mit dem *Dispatcher* verbunden sind. Auf den *Real Servers* befindet sich der eigentliche Dienst (z.B. ein Apache-Webserver), welcher angeboten wird. Sie bearbeiten die zu ihnen weitergeleiteten Anfragen und antworten den Nutzern. LVS unterstützt TCP- und UDP-Dienste, aber ist auch in der Lage ICMP Pakete zu routen.
Der *Dispatcher* stellt in solch einem Setup einen Single-Point-of-Failure dar, welcher, wenn er ausfällt, den gesamten Dienst zum Erliegen bringt. Ein Produktivsystem wird deshalb meist auch als *Hochverfügbarkeits-Cluster*, oder auch *HA-Cluster* (für engl. High-Availability-Cluster) genannt, konzipiert. Um dies zu verwirklichen, wird der *Dispatcher* redundant angelegt und mit entsprechender Software versehen. Diese erkennt, ob der *Dispatcher* ausgefallen ist und kann im Notfall für diesen einspringen. Damit die Dienste so wenig wie möglich von dem Ausfall betroffen werden, ist es nötig, Verbindungen zwischen dem *Dispatcher* und der Reservemaschine zu synchronisieren, so dass die bestehenden Verbindungen bei einem Ausfall nicht neu aufgebaut werden müssen. Mögliche Software, mit der ein LVS-Cluster zu einem HA-Cluster erweitert werden kann, ist z.B. *Piranha* [44], *Keepalived*[3] [45], *Ultra Monkey* [46] oder *heartbeat* [47] in Kombination mit *mon* [48] und *coda* [49] oder in Kombination mit dem *ldirectord* [50]. Diese Arbeit setzt keine HA-Komponenten ein, da diese nur für Produktivsysteme sinnvoll sind. Ein LVS-Cluster mit HA-Komponenten könnte wie in Abbildung 2.4 aussehen.

2.2.1.1 Virtual Server Konfiguration

LVS verfügt über drei verschiedene Möglichkeiten, Anfragen vom *Dispatcher* an die *Real Server* und von den *Real Servern* zurück an die Nutzer zu senden.

Virtual Server via NAT (VS/NAT) Die einfachste und in dieser Arbeit verwendete Variante wird via *Network Address Translation* (NAT) realisiert und auch als *masquerading* bezeichnet. Für diese Variante überprüft der *Dispatcher* jedes eingehende Paket, ob für die Ziel-Adresse und den Ziel-Port ein Dienst konfiguriert ist. Ist dies der Fall, wird anhand der Schedulingstrategie ein

[3] Nicht zu verwechseln mit der (fast) gleichnamigen Funktion aus HTTP/1.1

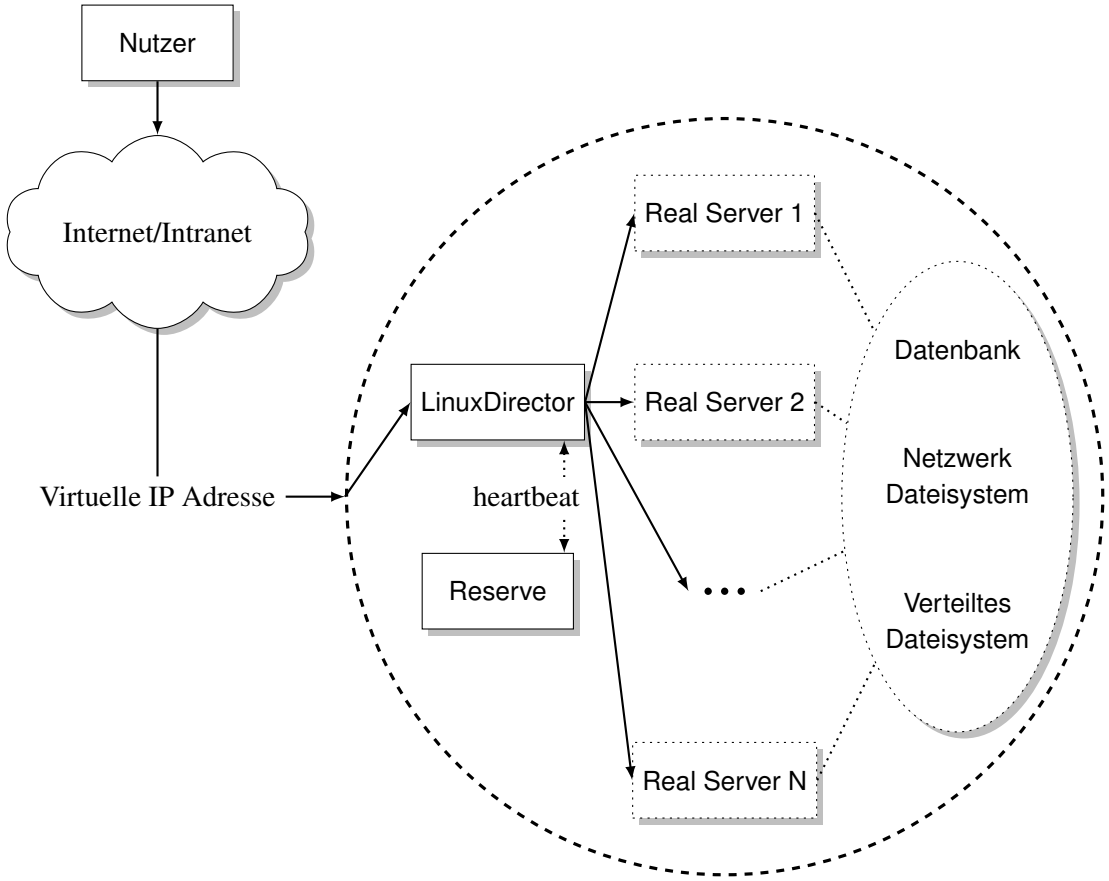


Abbildung 2.4: LVS-Cluster mit HA-Komponenten, nach [51].

Real Server ausgewählt, welcher die Anfrage beantworten soll. Die Adresse und der Port des Pakets werden auf diesen *Real Server* umgeschrieben und an ihn weiter geschickt. Für jede so entstehende Verbindung legt LVS einen Eintrag in einer internen Hash-Tabelle an, so dass weitere Pakete, welche zu derselben Verbindung gehören, an den gleichen *Real Server* weiter gesendet werden können. Beantwortete Anfragen werden zurück an den *Dispatcher* geschickt, welcher die Anfragen auf die ursprüngliche Adresse übersetzt und sie dann an den Nutzer zurück sendet. Die Verbindung wird aus der Hash-Tabelle entfernt, wenn die Verbindung beendet wird oder eine Zeitüberschreitung eintritt. [52]

Virtual Server via IP Tunneling (VS/TUN) Als *IP tunneling*, oder auch *IP encapsulation* (engl. für IP-Kapselung), wird die Technik bezeichnet, bei der ein IP-Paket in ein weiteres IP-Paket eingebettet wird und somit zu dessen Nutzlast wird. Der Vorteil von dieser Technik ist, dass die *Real Server* sich in einem beliebigen, auch geographisch anders gelegenem Netz befinden können und nicht im selben Netz oder am selben Ort wie der *Dispatchers* sein müssen. Um das zu ermöglichen, müssen die *Real Server* allerdings ein *IP tunneling* Protokoll konfiguriert haben, bei dem das dafür konfigurierte Tunnel-Device mit der vom *Dispatcher* verwendeten virtuellen IP versehen sein muss. Die Konfiguration einer solchen Installation gestaltet sich somit wesentlich umfangreicher,

aber hat dafür auch die erwähnten Vorteile.
Der Ablauf unterscheidet sich von VS/NAT. Eine Anfrage, die beim *Dispatcher* ankommt, wird, abhängig von der Schedulingstrategie, via *IP tunneling* an einen ausgewählten *Real Server* versendet. Dieser entpackt die eigentliche Anfrage und kann sie nun direkt beantworten, ohne dass der *Dispatcher* erneut tätig werden muss. Dieses Vorgehen entlastet den *Dispatcher*, wodurch diese Konfiguration performanter ist als VS/NAT und für größere Setups eingesetzt werden sollte. [53]

Virtual Server via Direct Routing (VS/DR) Um *direct routing* zu ermöglichen, müssen *Dispatcher* und *Real Server* alle mit einem ununterbrochenen LAN-Segment verbunden sein (z.B. einem Switch). Zusätzlich müssen alle *Real Server* die virtuelle IP Adresse des *Dispatchers* konfiguriert haben. Damit es nicht zu einem ARP-Konflikt kommt, müssen die Interfaces der *Real Server* als *hidden* markiert werden. So reagieren sie nicht auf eingehende ARP-Anfragen [4]. Zu einem ARP-Konflikt kann es unter Umständen auch in einer VS/TUN-Umgebung kommen, hier müssen die Interfaces auch entsprechend konfiguriert werden. [55]
Da der *Dispatcher* als einziger auf ARP-Anfragen antwortet, bekommt er auch als einziger Anfragen von Nutzern zugestellt. Er wählt, wie bei den anderen Verfahren auch, einen der *Real Server* zur Bearbeitung aus und sendet ihm das Paket weiter, indem er einzig die MAC-Adresse des Pakets durch die des *Real Servers* austauscht und das Paket wieder in das Netzwerk sendet. Durch die geänderte MAC-Adresse erhält der entsprechende *Real Server* das Paket und nimmt es auf dem dafür konfigurierten Interface an. Nach Bearbeitung kann er, wie bei VS/TUN auch, das Paket direkt an den Nutzer zurückschicken, ohne dass der *Dispatcher* noch einmal involviert wird. Dieses Vorgehen macht VS/DR ebenfalls performant, aber VS/DR ist im Gegensatz zu VS/TUN durch die Bedingung, dass alle Maschinen an einem ununterbrochenen LAN-Segment angeschlossen sein müssen, sehr limitiert.

2.2.1.2 Scheduling-Verfahren

Unabhängig von der verwendeten Methode zur Virtualisierung des Servers, kann aus verschiedenen Scheduling-Verfahren gewählt werden. Diese entscheidet darüber, welche Anfrage an welchen *Real Server* gesendet wird.

- **Round-Robin (RR):** Anfragen werden der Reihe nach an die vorhandenen *Real Server* verteilt. Hat jeder eine Anfrage erhalten, wird wieder von vorne begonnen. So erhalten alle *Real Server* immer gleich viele Anfragen, unabhängig von ihrer Auslastung.

- **Weighted Round-Robin (WRR):** Anfragen werden, abhängig von den konfigurierten Gewichten, reihum (wie bei RR) an die vorhandenen *Real Server* verteilt. Dabei bekommen Server mit höheren Gewichten entsprechend der Gewichtung mehr Anfragen zugeteilt.

- **Least-Connection (LC):** Anfragen werden an die Maschine mit den wenigsten aktiven Verbindungen gesendet. Dies ist nützlich bei stark variierender Last, ist aber nicht geeignet, um Server mit verschiedener Leistung zu verwalten. Verbindungen verbringen nach ihrem Ende standardmäßig noch 2 Minuten im TCP-Zustand *TIME_WAIT*. LVS betrachtet diese

[4] Das Verhalten ist abhängig vom verwendeten Kernel. Eine Erläuterung der verschiedenen Methoden zur Umsetzung eines Interfaces, welches nicht auf ARP-Anfragen reagiert, befindet sich in [54]

Verbindungen als aktiv, da diese immer noch Pakete annehmen können. Eine leistungsstarke Maschine könnte somit n Anfragen fertig bearbeitet haben und hätte gleichzeitig noch n aktive Verbindungen, alle in *TIME_WAIT*. Eine leistungsschwache Maschine mit n Anfragen hingegen hätte dieselbe Anzahl an Verbindungen, obwohl sie eventuell noch an allen n Anfragen arbeitet. Beide Maschinen würden bei LC nun gleichermaßen neue Anfragen zugewiesen bekommen, obwohl die leistungsschwache Maschine bereits überfordert ist.

- **Weighted Least-Connection (WLC):** Dieses Verfahren funktioniert wie LC, aber verwendet zusätzlich Gewichte für die konfigurierten Server. Anhand dieser Gewichte werden zusätzliche Anfragen an stärker gewichtete Maschinen vergeben. Durch die richtige Gewichtung kann das in LC beschriebene Problem in heterogenen Systemen behoben werden.

- **Shortest Expected Delay (SED):** SED ist WLC recht ähnlich. Hier wird der *Real Server* ausgewählt, welcher die kürzeste Bearbeitungszeit verspricht. Diese wird mit $C_i + 1/U_i$ berechnet, wobei C_i die Anzahl der momentanen Verbindungen und U_i die Gewichtung des *Real Servers i* ist.

- **Never Queue:** Dieses Verfahren wählt zuerst einen *Real Server* aus, der momentan keine Anfragen zu bearbeiten hat. Gibt es keinen solchen *Real Server*, wird das SED-Verfahren angewendet.

Für komplexere Cluster-Architekturen, die z.B. *Squid Caching Proxies* [56] vor ihre *Real Server* schalten, gibt es noch die Verfahren **Locality-Based Least-Connection**, **Locality-Based Least-Connection with Replication**, **Destination Hashing** und **Source Hashing**. Diese berücksichtigen vor allem die Lokalität von Anfragen, so dass Caches besser ausgenutzt werden können. Da in dieser Arbeit eine einfache Cluster-Architektur mit homogenen Servern verwendet wird, wird darauf verzichtet, die zusätzlichen Scheduling-Verfahren näher zu erläutern. Der Fokus der Arbeit liegt daher auf dem RR- und dem LC-Verfahren.

2.2.1.3 IP Virtual Server (IPVS)

Bei IPVS handelt es sich um die Kernel-Erweiterung, welche die erklärten Funktionalitäten ermöglicht. Dafür klinkt sich IPVS in den Kernel ein (ein sogenannter *hook*) und greift die Pakete dort ab oder schreibt sie um. IPVS implementiert für LVS einen sogenannten *Layer-4 Switch*, da dieser, bezugnehmend auf das OSI-Modell, auf Basis von Informationen bis zur 4. Schicht (IP-Adresse und Port) sein Routing vornimmt. Da nur Informationen bis Schicht 4 für das Routing verwendet werden, ist das Routing zwar performant, dafür müssen aber alle *Real Server* denselben Inhalt anbieten. In dieser Arbeit wird IPVS in der Version 1.2.1 verwendet.

2.2.1.4 Kernel TCP Virtual Server (KTCPVS)

Bei KTCPVS handelt es sich, wie bei IPVS, ebenfalls um eine (nicht völlig ausgereifte) Kernel-Erweiterung. Ebenfalls bezugnehmend auf das OSI-Modell, implementiert KTCPVS für LVS einen *Layer-7 Switch*, da es Routing-Entscheidungen auf Basis von Informationen bis zu Schicht 7 treffen kann. Da es sich hier um die Applikationsschicht handelt, ist dieses Routing zwar weniger performant als bei IPVS, dafür ist das Routing aber kontextabhängig und die *Real Server*

```
1 | [root@ib1 ~]# ipvsadm
2 | IP Virtual Server version 1.2.1 (size=4096)
3 | Prot LocalAddress:Port: Scheduler Flags
4 |   -> RemoteAddress:Port               Forward Weight  ActiveConn InActConn
5 | TCP ib1.leibniz.cluster:http lc
6 |   -> ib7.ib.cluster:http              Masq    0       0          0
7 |   -> ib8.ib.cluster:http              Masq    1       552        17998
```

Abbildung 2.5: Beispielhafte Ausgabe von `ipvsadm`.

müssen nicht alle die selben Inhalte zur Verfügung stellen. KTCPVS wird in dieser Arbeit nicht verwendet.

2.2.1.5 IPVSADM

Bei `ipvsadm` handelt es sich um das Kommandozeilen-Werkzeug, mit welchem LVS konfiguriert und überwacht wird. Diese Arbeit verwendet `ipvsadm` in der Version 1.24. Abbildung 2.5 zeigt die Informationen, die bei einem Aufruf des Programms gezeigt werden. Zu sehen ist eine Konfiguration mit zwei *Real Servern*, IB7 und IB8, wie sie später in Abschnitt 6.2.1 (Seite 79) Verwendung finden wird. Zeile 1 zeigt den parameterlosen Aufruf von `ipvsadm`, gefolgt von allgemeinen Informationen in Zeile 2. Die Zeilen 3 und 4 stellen beispielhaft dar, wie der Rest der Ausgabe zu lesen ist. In Zeile 5 ist demzufolge der konfigurierte Dienst angegeben. Der Dienst verwendet TCP als Protokoll und ist auf das Interface mit dem DNS-Namen `ib1.leibniz.cluster` auf Port 80 (`HTTP`) gebunden. Das verwendete Scheduling-Verfahren ist Least Connection (`lc`). Die Zeilen 6 und 7 geben Aufschluss über die beiden verwendeten *Real Server*. Für das Routing müssen ihre Adressen, der lokal verwendete Port sowie das Routing-Verfahren (`Masq` steht für *masquerading*, also VS/NAT) angegeben werden. Dahinter befinden sich noch die Informationen über die eingetragenen Gewichte (`Weight`) sowie die aktiven (`ActiveConn`) und inaktiven Verbindungen (`InActConn`) des entsprechenden *Real Servers*. Eine Gewichtung von 0 bedeutet, dass der *Real Server* keine Anfragen erhält und somit vom LVS abgemeldet ist.

2.2.2 Apache-Webserver

Der *Apache-Webserver* [57] ist das Produkt des Webserver Projekts der *Apache Software Foundation*. Die Foundation sagt über das Projekt:

> *"The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows. The goal of this project is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards."*
>
> [57] entnommen

Laut den Internet-Analyse-Seiten `w3techs.com` [58] und `Netcraft` [59] ist Apache der weltweit am meisten genutzte Webserver. Dabei untersucht `w3techs.com` prinzipiell alle Internetseiten und unterteilt dann weiter unter die Top $\{1.000.000, 100.000, 10.000, 1.000\}$ Seiten. `Netcraft` verwendet eine spezielle Metrik, bei der sie nur aktive Webseiten untersuchen. Der so entstehende

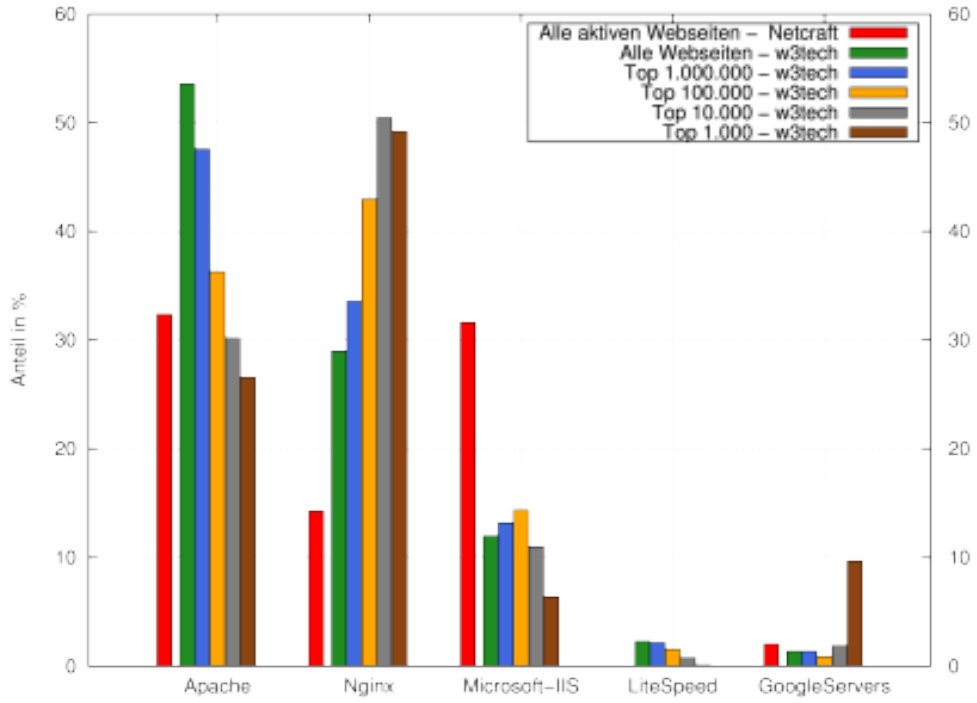


Abbildung 2.6: Weltweiter Marktanteil verschiedener Webserver. Daten von [58, 59].

Marktanteil ist in Abbildung 2.6 dargestellt und belegt die Nutzung von Apache als Marktführer. Es zeigt aber auch, dass Nginx unter den Top-Webseiten eine höhere Nutzung als Apache aufweist. In der LVS Terminologie läuft Apache als Dienst auf den *Real Servern*.

2.2.2.1 Multi-Processing Module

Wie Apache mit Anfragen umgeht, wird wesentlich durch das konfigurierte MPM bestimmt. Die aktuelle Version 2.4.23 (Stand 10.08.2016) unter Linux bietet drei verschiedene MPM, *MPM worker*, *MPM prefork* und *MPM event*. Es gibt noch eine ganze Reihe von Modulen, die verschiedenste Funktionen bereitstellen (z.B. erweitertes Protokollieren, Authentifikation, Caching etc.) aber diese sind optional und es sollte zwischen Funktionalität und Performance abgewogen werden, da jedes Modul zwar Funktionalität bereitstellt, aber Ressourcen kostet.

MPM prefork Das *prefork* Modul implementiert einen Webserver, der keine Threads verwendet. Dies ist vor allem nützlich, wenn Bibliotheken verwendet werden sollen, die nicht *thread-safe* sind. In diesem Modul werden durch vorherige `fork()` Befehle (daher der Name *pre-fork*) neue Arbeiter durch Prozesse erzeugt (mehr zur Prozesserzeugung in Kapitel 7.1, Seite 96). Da jeder Arbeiter in einem separaten Prozess läuft, sind diese weniger anfällig für Seiteneffekte, bei denen es zum Absturz eines Prozesses kommt. Würden mehrere Arbeiter innerhalb eines Prozesses laufen, hätte ein Absturz ebenfalls Folgen für die anderen Arbeiter innerhalb des Prozesses. [60]

MPM worker Das *worker* Modul implementiert einen Webserver, der, wie das *prefork* Modul, ebenfalls mehrere Prozesse verwendet, aber, im Gegensatz zum *prefork* Modul, pro Prozess wiederum mehrere Threads besitzt, die jeweils einen Arbeiter umsetzen. Durch die Verwendung von Threads werden die Systemressourcen besser ausgenutzt (Threads benötigen weniger Speicher als Prozesse), was dieses Modul performanter macht. Es ist deshalb aber, wie bereits beim *prefork* Modul beschrieben, anfälliger für Seiteneffekte, wenn es zu Problemen kommt. [61]

MPM event Das *event* Modul basiert auf dem *worker* Modul und ergänzt dieses um einen dedizierten *listener thread*. Dieser soll Probleme mit *Keep-Alive* [5] verhindern. Sowohl das *prefork* Modul als auch das *worker* Modul würden bei aktivem *Keep-Alive* pro Verbindung jeweils einen Prozess bzw. Thread nach Bearbeitung einer Anfrage bestehen lassen um auf weitere Anfragen zu warten. Der dedizierte *listener thread* übernimmt nun diese Aufgabe wofür ihm der entsprechende Socket vom bearbeitenden Prozess/Thread übergeben wird, nachdem dieser seine Antwort zum Client gesendet hat. Kommen erneut Daten an dem Socket an, kann dieser vom *listener thread* wieder an einen freien Arbeiter delegiert werden. Auf diese Weise werden Arbeiter durch *Keep-Alive* Verbindungen nicht mehr blockiert. [62]

Accept Mutex Ohne zu tief in technische Details zu gehen, ist es für den Leser noch interessant zu wissen, wie Apache eingehende Verbindungen/Anfragen verarbeitet. Für die Verarbeitung wird ein einzelner Port [6] im Hauptprozess genutzt. Auf dessen Socket greifen alle Arbeiter gemeinsam zu. Um diese Zugriffe zu organisieren, wird der sogenannte *Accept Mutex* verwendet, welcher von dem Arbeiter, der eine neue Anfrage entgegennehmen will, in Anspruch genommen wird. Durch dieses Vorgehen werden die Zugriffe sequenziell abgearbeitet und es kommt zu keinen Interferenzen beim Zugriff auf den Socket. Für mehr technische Details ist die Bachelorarbeit von Sijing You zu empfehlen [63]. Diese beschreibt die Funktion von Apache 2.4.x. In der Arbeit werden Funktionsweisen und die beiden MPM Module *worker* und *prefork* beschrieben, welche keine Änderungen hinsichtlich Version 2.2.x erfahren haben und sind somit auch für die verwendete Version gültig.

2.2.2.2 Parameter

Die Apache-Parameter, die in dieser Arbeit nicht ihren Standardwert behalten, werden hier genannt. Wenn diese Parameter verändert werden, ist dies an entsprechender Stelle in der Arbeit vermerkt.

Parameterbezeichnung:	ListenBacklog
Standardwert:	511 (Kernel-Parameter können diesen beschränken)
Bedeutung:	Dieser Parameter gibt die Länge der TCP-Backlog Queue an (siehe Abschnitt 2.2.5, Seite 30).

[5] Keep-Alive ist eine Funktionalität von HTTP/1.1, welche persistente Verbindungen realisiert. Durch persistente Verbindungen können mehrere Anfragen mit Hilfe der selben Verbindung durchgeführt werden, anstelle für jede Anfrage eine separate, aufwendige TCP-Verbindung aufzubauen.

[6] Standard ist Port 80. Für nicht privilegierte Nutzer wird typischerweise Port 8080 verwendet, da diese nur Portnummern größer 1024 verwenden dürfen. Es kann aber auch jeder andere beliebige (nicht bereits verwendete) Port genutzt werden.

Parameterbezeichnung:	KeepAlive
Standardwert:	An
Verwendeter Wert:	Aus
Bedeutung:	Dieser Parameter erlaubt die Verwendung von *Keep-Alive*. Um die Szenarien dieser Arbeiten möglichst überschaubar zu halten und die verwendeten Werkzeuge zur Lasterzeugung (siehe Abschnitt 2.2.6, Seite 32) HTTP/1.1 nicht unterstützen, wurde die Funktion abgeschaltet.

Parameterbezeichnung:	ExtendedStatus
Standardwert:	Aus
Verwendeter Wert:	An
Bedeutung:	Mit Hilfe des Moduls *mod_status* stellt Apache eine Seite bereit (`http://your.server.name/server-status`), welche Aufschluss über den momentanen Zustand des Servers gibt. Die Status-Seite wird ab Abschnitt 7.3 (Seite 100) von *CHERUB* verwendet, um Lastinformationen innerhalb der Status-Funktion zu sammeln. Der Simulator *ClusterSim* imitiert die Seite vollständig (Abschnitt 8.4, Seite 113). Abbildung 8.5 (Seite 115) zeigt, wie diese Status-Seite in der maschinenlesbaren [7] Variante aufgebaut ist.

Die folgenden Parameter beziehen sich auf das verwendete *MPM prefork*. Sie definieren, wie viele Arbeiter-Prozesse von Apache gestartet und vorgehalten werden sollen.

Parameterbezeichnung:	StartServers
Standardwert:	5 (für *MPM prefork*)
Bedeutung:	Gibt an, wie viele Arbeiter-Prozesse initial gestartet werden. Da sich die Anzahl der Arbeiter schnell verändern kann, ist diese Konfiguration nicht besonders wichtig und wurde für spätere Messungen, wenn nicht anders angegeben, gleich der Anzahl der maximalen Arbeiter (`MaxClients`) gesetzt.

Parameterbezeichnung:	MinSpareServers / MaxSpareServers
Standardwert:	5 / 10
Bedeutung:	Gibt an, wie viele Arbeiter-Prozesse, die keine Anfragen bearbeiten, gleichzeitig mindestens / maximal existieren müssen / dürfen. Um zu verhindern, dass es zu einem Arbeitermangel kommt, wurden die Parameter, wenn nicht anders angegeben, gleich der Anzahl der maximalen Arbeiter (`MaxClients`) gesetzt.

Parameterbezeichnung:	MaxClients
Standardwert:	256
Bedeutung:	Gibt an, wie viele Arbeiter-Prozesse maximal erzeugt werden dürfen.

[7] Die maschinenlesbare Variante wird mit `http://your.server.name/server-status?auto` abgerufen.

Parameterbezeichnung:	MaxRequestsPerChild
Standardwert:	10.000
Bedeutung:	Gibt an, wie viele Anfragen von einem einzelnen Arbeiter-Prozess in seinem Lebenszyklus bearbeitet werden können. Ist die Anzahl erreicht, wird der Prozess zerstört und gegebenenfalls durch einen neuen ersetzt. Der Wert 0 deaktiviert diesen Mechanismus.

Optimale Anzahl Arbeiter-Prozesse Die optimale Anzahl von Arbeitern ist immer vom konkreten Szenario abhängig. Dabei gilt nicht automatisch *mehr ist besser*. Es sollte immer gelten:

$$Prozessgröße * Prozessanzahl < Arbeitspeicher \tag{2.4}$$

Ist Formel 2.4 nicht erfüllt, muss der Server Arbeitspeicher auf vorhandene Festplatten auslagern, (sogenanntes *swapping*) um Platz im Arbeitspeicher zu schaffen. Dieses Vorgehen benötigt verhältnismäßig viel Zeit und ist katastrophal für die Performance jedes Servers.
Ein weiterer Zusammenhang besteht zur Verweilzeit der Anfragen. Je mehr Prozesse aktiv sind, desto geringer ist die Rechenzeit, die jeder einzelne Prozess in der CPU erhält. Wenn ausschließlich Anfragen mit geringer Bearbeitungszeit (Mikrosekundenbereich) bearbeitet werden müssen, stellt dies kein Problem dar. Muss der Server allerdings längere Anfragen (Sekundenbereich) bearbeiten, kann die durchschnittliche Verweilzeit der Anfragen unerwünscht steigen. Beispiel 1 stellt diesen Zusammenhang anschaulich dar.

Beispiel 1 Für das Beispiel wird eine sehr einfache und faire Verteilung der Rechenzeit angenommen, bei der jede Millisekunde der nächsten Arbeiter die CPU zum Rechnen erhält. Es wird außerdem angenommen, dass alle Anfragen hauptsächlich die CPU als Ressource verwenden.
Fall 1: 1 Server (1 Rechenkern) mit 5 Arbeiter-Prozessen bekommt 5 Anfragen zu je 200 ms Bearbeitungszeit. Jeder Prozess bekommt eine Anfrage zugewiesen, diese werden dann parallel bearbeitet. Bei angenommenem Scheduling ist die Verweilzeit der Anfragen: 996 ms, 997 ms, 998 ms, 999 ms und 1000 ms (Durchschnitt: 998, Median: 998).
Fall 2: 1 Server (1 Rechenkern) mit 1 Arbeiter-Prozess bekommt 5 Anfragen zu je 200 ms Bearbeitungszeit. Der einzelne Prozess muss diese sequenziell bearbeiten. Bei angenommenem Scheduling ist die Verweilzeit der Anfragen: 200 ms, 400 ms, 600 ms, 800 ms, 1000 ms (Durchschnitt: 600, Median: 600). □

Obwohl Fall 2 in hohem Maße ungerecht ist (sehr unterschiedliche Verweilzeiten der einzelnen Anfragen), ist dieser im Durchschnitt sehr viel effizienter (Anfragen besitzen im Durchschnitt nur 60 % der Verweilzeit von Fall 1). Problematisch wird es, wenn die Bearbeitungszeit der Anfragen stark variiert. Ist dies der Fall, empfiehlt es sich, mehr Arbeiter zu verwenden, da ansonsten die wenigen Arbeiter von rechenintensiven Anfragen blockiert werden können.

2.2.3 CHERUB

Das in dieser Arbeit verwendete und weiterentwickelte ESS trägt den Namen *CHERUB*. Das *CHERUB*-Projekt wurde im Jahr 2010 im Rahmen einer Diplomarbeit [20, 21, 22] entwickelt und in

einer prototypischen Version veröffentlicht. Es zeichnet sich durch eine fest definierte API und eine modulare Bauweise aus. Diese ermöglicht, dass Adapter für beliebige RMS entwickelt werden können. Die Diplomarbeit fokussierte sich dabei auf den HPC-Bereich, wohingegen diese Arbeit die Funktionsweise in einer SLB-Umgebung untersuchen und optimieren soll. Das Konzept und die dafür notwendigen Änderungen an *CHERUB* werden in Abschnitt 4.3, Seite 60 vorgestellt. In der LVS Terminologie läuft *CHERUB* parallel mit LVS auf dem *Dispatcher*.

Zustände Das Vorgehen von *CHERUB* sieht dabei zunächst das Vorhalten einer internen Repräsentation des zu verwaltenden Clusters vor. In dieser Repräsentation besitzt jeder Knoten einen von fünf Zuständen. Die definierten Zustände sind:

- **UNKNOWN:** Dieser Zustand ist als Notfall-Zustand gedacht. Er wird nur dann verwendet, wenn keiner der anderen Zustände ermittelt werden konnte. Dies ist nur im Fehlerfall möglich (z.B. Netzwerkfehler). Kann im Folgenden wieder einer der anderen Zustände ermittelt werden, wird dieser entsprechend angenommen (gestrichelte Pfeile, Abbildung 2.8, Seite 28).
- **BUSY:** Dieser Zustand bedeutet, dass die Maschine arbeitet, am RMS angemeldet ist und auf keinen Fall abgeschaltet werden darf.
- **ONLINE:** Eine Maschine besitzt diesen Zustand, wenn sie so wenig Last besitzt, dass sie abgeschaltet werden kann. Die Maschine ist außerdem am RMS angemeldet, so dass sie jederzeit neue Last erhalten kann.
- **OFFLINE:** In diesem Zustand ist die Maschine vom RMS abgemeldet und kann somit keine neue Last annehmen. Noch nicht fertig bearbeitete Last kann weiterhin bearbeitet werden.
- **DOWN:** In diesem Zustand ist die Maschine abgeschaltet. Sie sollte außerdem vom RMS abgemeldet sein, was zwar keine notwendige Bedingung für den Zustand *DOWN* darstellt, aber durch das Verhalten von *CHERUB*, ohne Fremdeinwirkung, erzwungen wird. Da die Maschine aus ist, kann sie auch keine Last bearbeiten und verbraucht auch keinen Strom [8].

2.2.3.1 Hauptroutine

Um die Zustände der einzelnen Maschinen zu ermitteln, wird in regelmäßigen Abständen die Hauptroutine von *CHERUB* ausgeführt. Das Intervall wird im Laufe der Arbeit auch als *Runde* bezeichnet (siehe hierfür Beispiel 2).

Beispiel 2 *Maschine X war für 5 Runden BUSY.* Dies bedeutet, dass Maschine X in den letzten fünf durchgeführten Hauptroutinen jeweils den Zustand BUSY hatte. □

[8] Technisch wird die Maschine weiterhin mit einer Grundspannung versorgt und verbraucht damit eine geringe Menge an Energie, die im weiteren Verlauf der Arbeit vernachlässigt wird. Mit dieser Grundspannung arbeitet unter anderem der erwähnte BMC, der IPMI (Abschnitt 2.1.2, Seite 11) umsetzt, bzw. die Netzwerkkarte bei der Verwendung von WOL (Abschnitt 2.1.3, Seite 11). Auch für STR (Abschnitt 2.1.1, Seite 9) ist die Grundspannung erforderlich, damit der RAM weiterhin seine Informationen halten kann.

```
1  def Hauptroutine(Cluster):
2          while(true):
3                  StatusFunktion(Cluster)
4                  LastFunktion(Cluster)
5                  for Rechner in Cluster:
6                          Zustandmaschine(Rechner)
7                  Sleep(Intervall - BenötigteRechenzeit)
8
9          ...
10
11 def Zustandsmaschine(Rechner):
12         if Rechner.Zustand == BUSY:
13                 Verhindere Abschalten
14         if Rechner.Zustand == ONLINE:
15                 Überprüfe Bedingungen zum Abmelden
16         if Rechner.Zustand == OFFLINE:
17                 Überprüfe Bedingung zum Anmelden
18                 Überprüfe Bedingung zum Abschalten
19         if Rechner.Zustand == DOWN:
20                 Überprüfe Bedingung zum Anschalten
21         if Rechner.Zustand == UNKNOWN:
22                 Protokolliere Warnung
```

Abbildung 2.7: Vereinfachter Pseudocode von *CHERUBs* Hauptroutine.

Die Länge des Intervalls ist entscheidend für die Reaktionsfähigkeit von *CHERUB*. Die optimale Länge des Intervalls wird in dieser Arbeit nicht weiter untersucht.
Der Pseudocode in Abbildung 2.7 stellt den Ablauf, inklusive der Zustandsmaschine, dar. Ein `Überprüfe Bedingung` impliziert die Ausführung der jeweiligen Aktion, falls die Bedingung erfüllt ist.

2.2.3.2 Informationssammelnde Funktionen

Innerhalb einer Runde werden zwei essentielle Informationen durch die sogenannten *informationssammelnden Funktionen* ermittelt. Erstens, der aktuelle Zustand jeder Maschine durch die Status-Funktion [9] (Zeile 3, Abbildung 2.7) und zweitens, die aktuell vorherrschende Last durch die Last-Funktion (Zeile 4, Abbildung 2.7).
Bei der Ermittlung der Last gibt es zwei grundlegende Vorgehensweisen. Entweder man überprüft die Last jeder einzelnen Maschine oder man abstrahiert und überprüft die Last des gesamten Systems. Beides hat seine Vor- und Nachteile und im Verlauf der Arbeit werden beide Varianten besprochen und verwendet (Abschnitt 7.4, Seite 102). Unabhängig von der gewählten Methode setzt die Last-Funktion, wenn sie Überlast ermittelt hat, ein knotenspezifisches Flag (engl. Markierung)

[9] Wird auch als Zustands-Funktion bezeichnet.

welches signalisiert, dass die Maschine gewählt wurde, um die anstehende Last abzufangen. Dieses Flag wird daher auch als Last-Markierung bezeichnet. Wann Überlast besteht, muss von Setup zu Setup ermittelt werden.
Nachdem beide informationssammelnden Funktionen (Last- und Status-Funktion) durchgeführt wurden, hat *CHERUB* ein aktuelles Bild des Clusters und kann auf Basis dieser Informationen Entscheidungen über zustandsverändernde Aktionen treffen.

2.2.3.3 Zustandsverändernde Aktionen

CHERUB beherrscht vier zustandsverändernde Aktionen, die jeweils pro Maschine ausgeführt werden können:

- **Boot (engl. hochfahren/starten):** Die Maschine erhält einen Befehl zum Hochfahren, welcher entweder via IPMI oder via WOL gesendet wird (Abschnitt 2.1, Seite 9).
  Bedingung: Die Maschine muss sich im Zustand *DOWN* befinden und ihre Last-Markierung muss gesetzt sein.
- **Shutdown (engl. herunterfahren/abschalten):** Die Maschine erhält einen Befehl zum Herunterfahren, welcher entweder via IPMI oder als Kommandozeilenbefehl via SSH gesendet wird.
  Bedingung: Die Maschine muss sich im Zustand *OFFLINE* befinden und ihre Last-Markierung darf nicht gesetzt sein.
- **Register (engl. registrieren/anmelden):** Die Maschine wird am lokalen RMS angemeldet, so dass ihr danach Last vom RMS zugewiesen werden kann.
  Bedingung: Die Maschine muss sich im Zustand *OFFLINE* befinden und ihre Last-Markierung muss gesetzt sein.
- **Sign Off (engl. abmelden):** Die Maschine wird vom lokalen RMS abgemeldet, so dass ihr danach keine Last mehr vom RMS zugewiesen werden kann.
  Bedingung: Die Maschine muss sich im Zustand *ONLINE* befinden und ihre Last-Markierung darf nicht gesetzt sein.

2.2.3.4 Verzögerungsparameter

Für alle vier Aktionen gibt es außerdem einen konfigurierbaren Verzögerungsparameter, welcher dafür sorgt, dass die jeweilige Aktion nicht unmittelbar ausgeführt wird. Ist ein Zeitfenster konfiguriert, muss in jedem Intervall innerhalb des Zeitfensters die Bedingung für die entsprechende Aktion erfüllt sein. Ist dies der Fall, wird die Aktion am Ende des Zeitfensters ausgeführt (siehe Beispiel 3). Da es insgesamt nicht viele Bedingungen für die zustandsverändernden Aktionen gibt, wird empfohlen für das Abmelden und/oder Herunterfahren einer Maschine, deren Verzögerungsparameter größer null zu setzten. Andernfalls führt dies dazu, dass schnell jegliche Reserve an Maschinen abgeschaltet ist bzw. bei schwankender Last eine unerwünschte Oszillation einsetzt (siehe dazu auch Abschnitt 4.2, Seite 59)

Beispiel 3 Ist der Verzögerungsparameter für die *Sign Off*-Aktion auf 1 Minute gestellt und das Intervall auf 15 Sekunden, muss eine Maschine vier Runden in Folge die Bedingung für die *Sign Off*-Aktion erfüllen, damit die Aktion auch ausgeführt wird. □

2.2.3.5 Zustandsdiagramm

Abbildung 2.8 zeigt noch einmal vereinfacht die fünf Zustände und ihre Zusammenhänge. Da der Zustand einer Maschine unabhängig von seinem vorherigen Zustand ist, existieren theoretisch Übergänge von jedem Zustand zu jedem anderen Zustand. Diese Übergänge können aber nur durch Fremdeinwirkungen erreicht werden und sind der Übersicht wegen nicht eingezeichnet (siehe Beispiel 4).

Beispiel 4 Ist eine Maschine im Zustand *BUSY* und jemand trennt die Maschine vom Strom, wird sie in der nächsten Runde den Zustand *DOWN* besitzen. □

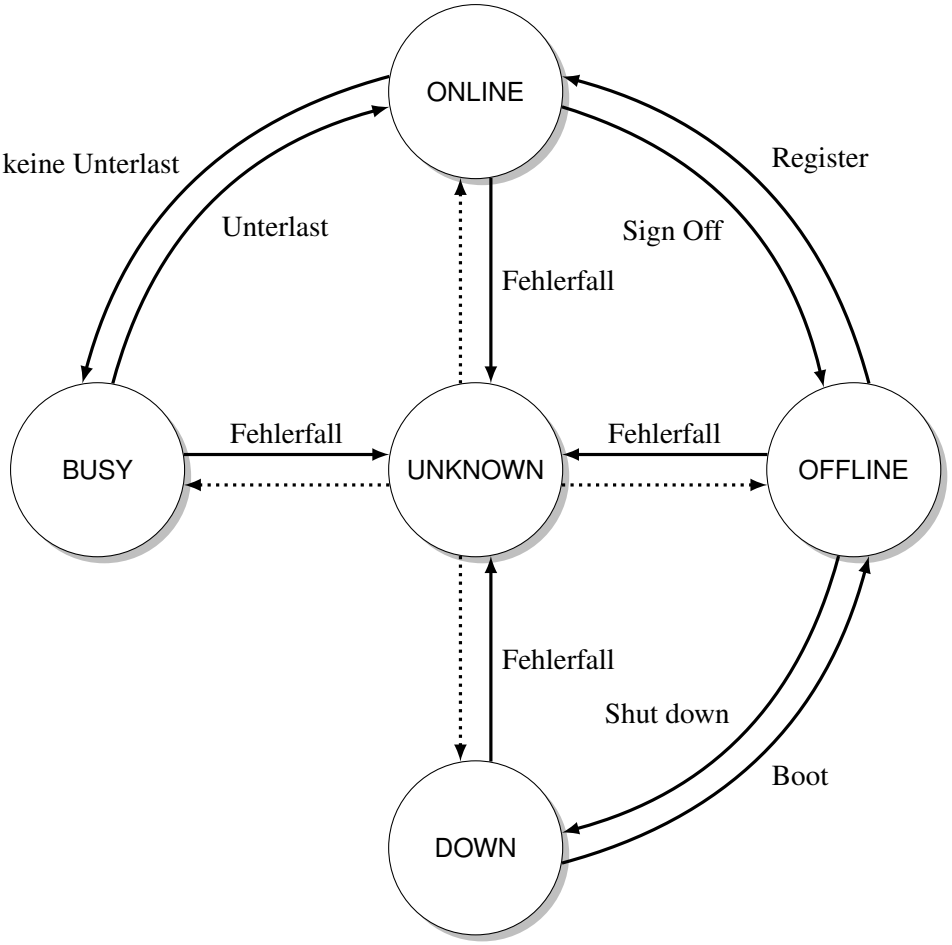


Abbildung 2.8: Vereinfachtes *CHERUB* Zustandsdiagramm ohne Fremdeinwirkung.

2.2.3.6 Parameter

Einige der nun folgenden konfigurierbaren *CHERUB* Parameter wurden bereits erwähnt und erklärt. Sie werden der Vollständigkeit wegen mit aufgeführt. Alle folgenden Parameter werden im Verlauf der Arbeit vielfach verwendet.

Parameterbezeichnung:	Hauptroutine, Intervall, Runde
Quellcodebezeichner:	`update_everything`
Standardwert:	60 Sekunden
Bedeutung:	Gibt an, wie häufig die Hauptschleife von *CHERUB* ausgeführt wird und mit ihr Informationen abgefragt und Aktionen ausgeführt werden (Abschnitt 2.2.3.1, Seite 25).

Parameterbezeichnung:	Anmeldeverzögerer
Quellcodebezeichner:	`wait_for_register`
Standardwert:	0 Minuten
Bedeutung:	Gibt an, wie lange gewartet wird, bis ein Rechner im Zustand *OFFLINE* und mit Last-Markierung am RMS angemeldet wird.

Parameterbezeichnung:	Abmeldeverzögerer
Quellcodebezeichner:	`wait_for_sign_off`
Standardwert:	1 Minute
Bedeutung:	Gibt an, wie lange gewartet wird, bis ein Rechner im Zustand *ONLINE* vom RMS abgemeldet wird.

Parameterbezeichnung:	Anschaltverzögerer
Quellcodebezeichner:	`wait_for_boot`
Standardwert:	1 Minute
Bedeutung:	Gibt an, wie lange gewartet wird, bis ein Rechner im Zustand *DOWN* und mit Last-Markierung gestartet wird.

Parameterbezeichnung:	Abschaltverzögerer
Quellcodebezeichner:	`wait_for_shutdown`
Standardwert:	1 Minute
Bedeutung:	Gibt an, wie lange gewartet wird, bis ein Rechner im Zustand *OFFLINE* und ohne Last-Markierung heruntergefahren wird.

Parameterbezeichnung:	Sequenzielles Abschalten
Quellcodebezeichner:	`sequential_shutdown`
Standardwert:	1
Bedeutung:	Gibt an, wie viele Rechner zum gleichen Zeitpunkt abgemeldet werden können (und damit auch heruntergefahren werden können). Durch diesen Parameter ist ein schrittweises Abschalten der Ressourcen möglich. Der Wert 0 aktiviert die automatische Erkennung, bei der *CHERUB* selbst entscheidet, wie viele Ressourcen abgeschaltet werden können. Der Wert 0 wird im späteren Verlauf auch als aggressives Abschalten bezeichnet, da die automatische Erkennung versucht, so viele Ressourcen wie möglich abzuschalten.

Es gibt noch weitere Service-Parameter (z.B. E-Mail Adresse für Benachrichtigung im Fehlerfall, Log-Level etc.), welche aber für das Verständnis der Arbeit nicht relevant sind und deshalb nicht weiter behandelt werden.

2.2.4 TCPDump

`tcpdump` [64, 65] ist ein mächtiges Kommandozeilen-Werkzeug zum Erfassen und Analysieren von Netzwerkverkehr. Es ist in der Lage, verschiedenste Filter anzuwenden und ermöglicht daher sehr gezieltes Überwachen, sowohl von eingehendem, als auch von ausgehendem Netzwerkverkehr. Es ist auch in der Lage, Dateien zu schreiben und zu lesen, welche kompatibel mit dem häufig verwendeten GUI-Netzwerlanalyse-Werkzeug Wireshark [66] ist. `tcpdump` wird in Version 4.3.0 verwendet, die aktuelle Version ist 4.7.4.

libpcap Damit `tcpdump` (und auch Wireshark) seine Funktionalitäten bereitstellen kann, macht es sich die C/C++ Bibliothek *libpcap* [64, 67] zu Nutze. Die *libpcap* implementiert für Unix-Systeme [10] verschiedenste Funktionen zum Erfassen und Filtern von Netzwerkverkehr. Auch das Versenden von Paketen ist mit der Bibliothek möglich. Dabei ist es ebenfalls möglich, Verkehr zu erfassen, der nicht an den eigenen Rechner adressiert ist. Voraussetzung ist, dass das Netzwerkpaket am eigenen Rechner eingeht. Vor allem bei WLAN ist dies ein sicherheitskritisches Thema. Im WLAN kann jeder Rechner den gesamten Verkehr aller Rechner im selben WLAN erfassen (die Luft ist ein geteiltes Medium). Im LAN kann dies möglich sein, wenn das Netzwerk nicht *geswitched* wird und deshalb Netzwerkpakete an alle Rechner im Netzwerk versendet werden.

Berkeley Packet Filter Um den Netzwerkverkehr zu erfassen und zu filtern, macht sich die *libpcap* den *Berkeley Packet Filter* (BPF) [69] [11] zu Nutze.
Abbildung 2.9 zeigt die Funktionsweise des BPF, wie sie in [69] vorgestellt wurde. Normalerweise sendet der Treiber des Netzwerk-Interfaces die empfangenen Pakete an den entsprechenden Protokoll-Stack des Kernels. Wird der BPF verwendet, sorgt dieser dafür, dass am Netzwerk-Interface eingehender Verkehr zuerst zum BPF geschickt wird. Dort prüfen konfigurierbare Filter, ob das Paket den Filterregeln entspricht. Dabei kann nach allem gefiltert werden, von Verwendung eines speziellen Protokolls bis hin zu Host-Adressen, Ports etc. Trifft der Filter auf das Paket zu, wird das entsprechende Paket bzw. der gewünschte Teil des Pakets aus dem Treiber kopiert und an den entsprechenden Prozess (z.B. `tcpdump`) zur Verarbeitung weitergegeben.

Eine detaillierte Beschreibung der Kernel-Strukturen und die damit einhergehende Umsetzung des BPF kann in Wright und Stevens *TCP/IP Illustrated, Volume 2, The Implementation* [71] ab Seite 1027 (Kapitel 31) gefunden werden. Das Zusammenspiel zwischen BPF und `tcpdump` findet sich in Stevens *TCP/IP Illustrated, Volume 1, The Protocols* [72] ab Seite 491 (Anhang A).

2.2.5 TCP-Backlog-Queue (BLQ) / Kernel-Parameter

Wenn in dieser Arbeit von der Backlog-Queue (BLQ) gesprochen wird, ist die TCP-Backlog-Queue gemeint. Die BLQ ist eine Datenstruktur im Kernel und besteht genau genommen aus zwei Queues. Die *receive* und *accept* Queue. Die *receive* Queue beinhaltet dabei TCP-Verbindungen, die noch nicht fertig aufgebaut sind. Nachdem der Drei-Wege-Handschlag einer Verbindung abgeschlossen ist, wird die Verbindung in die *accept* Queue des entsprechenden Sockets verschoben.

[10] Für Windows-Systeme steht die WinPcap [68] Bibliothek zur Verfügung.

[11] In Linux-Systemen wird der *Linux Socket Filter* (LSF) verwendet, welches den selben Filtermechanismus verwendet wie der BPF. [70]

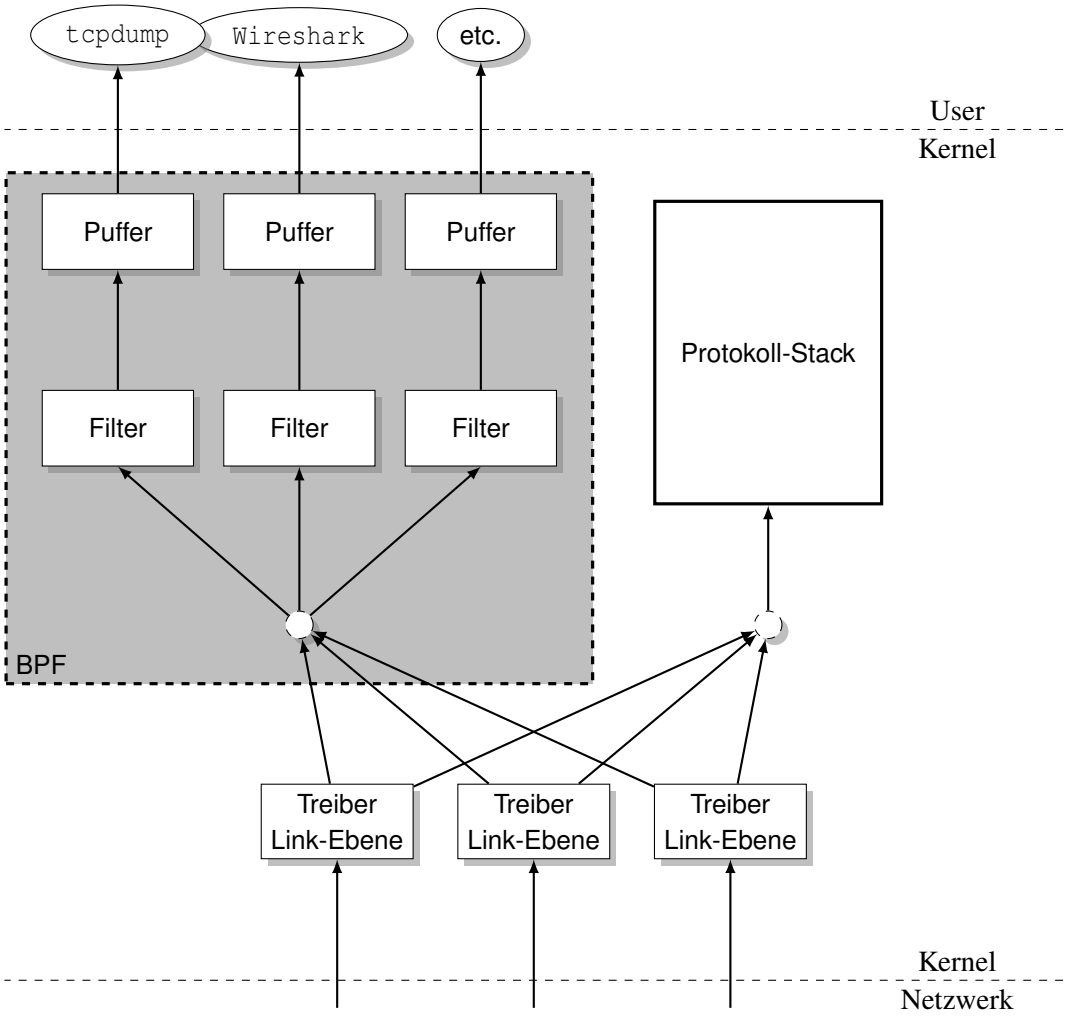


Abbildung 2.9: Übersicht der BPF Funktionsweise, nach [69].

Hier kann eine Verbindung mit dem namensgebenden Befehl `accept()` von einer Applikation entgegengenommen werden, woraufhin sie aus der *accept* Queue entfernt wird. Parameter, die sich auf die Länge der Queue beziehen (beispielsweise der Apache *ListenBacklog* Parameter, Abschnitt 2.2.2.2, Seite 22), umfassen typischerweise die Summe beider Queues. Hier muss beachtet werden, dass die maximale Länge von dem Kernel-Parameter `net.core.somaxconn` limitiert wird. Dieser müsste, wenn der Apache-Parameter größer wäre, ebenfalls angehoben werden, um den gewünschten Effekt einer längeren Queue zu erzielen.

Eine eingehende Verbindung, die nicht mehr in die Queue passt, wird standardmäßig verworfen, ohne dass es eine Rückmeldung gibt (*silently dropped*). Dieses Verhalten lässt sich durch den Kernel-Parameter `sys.net.ipv4.tcp_abort_on_overflow` ändern. Ist dieser ungleich Null, wird stattdessen ein TCP-RST (Reset) gesendet. Das RST zeigt dem Client, dass keine Verbindung aufgebaut werden kann und beendet den Verbindungsaufbau. Dieses Verhalten wird für Produktivsysteme nicht empfohlen, da ein Überlaufen der BLQ im Normalfall nur eintritt, wenn die Maschine

überlastet ist. Ist dies der Fall, würde die Maschine weitere Ressourcen verbrauchen, um RST-Pakete zu versenden, und die Situation der Maschine würde weiter verschlimmert. Weitere Details, inklusive Kernel-Strukturen etc. können in *UNIX Network Programming* von Stevens, Fenner und Rudoff [73], Kapitel 4 (Seite 95) und insbesondere in Abschnitt 4.5 (Seite 104) gefunden werden.

Optimale Länge der BLQ Ähnlich wie bei der optimalen Anzahl der Arbeiter-Prozesse eines Webservers verhält es sich mit der optimalen Länge der BLQ, denn auch hier gilt nicht immer *mehr ist besser*. Es ist notwendig zu wissen, welche Art von Dienst der verwendete Server normalerweise ausliefert. Handelt es sich um rechenintensive Aufgaben, empfiehlt es sich, die BLQ kürzer zu gestalten. Der Grund ist, dass die Anfragen der Reihe nach aus der BLQ entnommen werden. Füllt sich die BLQ, weil der Server nicht mit der Abarbeitung hinterherkommt, müssen alle Anfragen in der BLQ darauf warten, dass ihre Vorgänger fertig bearbeitet werden. Diese Wartezeit erhöht sich kaskadierend mit jeder Anfrage in der BLQ. Ist die BLQ zu lang, kann es passieren, dass alle Anfragen in der BLQ auf Nutzerseite eine Zeitüberschreitung erfahren. Ist die BLQ kürzer, werden einige Anfragen verworfen, da sie nicht mehr in die BLQ passen. Anfragen, die es in die BLQ schaffen, haben aber dafür eine Wartezeit + Bearbeitungszeit, die, richtig konfiguriert, zu keiner Zeitüberschreitung führt. Beispiel 5 illustriert dies noch einmal.

Beispiel 5 Für dieses Beispiel wird ein Server mit nur einem Arbeiter-Prozess angenommen. Zusätzlich werden Anfragen angenommen, die im Schnitt 50 ms Bearbeitungszeit besitzen. Ein Client registriert eine Zeitüberschreitung, wenn seine Anfrage nicht binnen 5 Sekunden bearbeitet ist. Die BLQ wird als voll angenommen und eine noch nicht bearbeitete Anfrage befindet sich bereits im Arbeiter-Prozess.
Fall 1: BLQ Länge 200: Die letzte Anfrage in der Queue besitzt hier eine Anfragedauer von 10,05 Sekunden (200 * 50 ms Wartezeit + 50 ms Bearbeitungszeit). Der Client erfährt eine Zeitüberschreitung.
Fall 2: BLQ Länge 50: Die letzte Anfrage in der Queue besitzt hier eine Anfragedauer von 2,55 Sekunden (50 * 50 ms Wartezeit + 50 ms Bearbeitungszeit). Der Client erfährt keine Zeitüberschreitung. □

Obwohl im 2. Fall weniger Anfragen in der Queue aufgenommen werden, haben die, die es in die Queue schaffen, die Möglichkeit innerhalb der Zeitbeschränkung zu bleiben. Im 1. Fall werden hingegen mehr Anfragen angenommen, aber da sie sich gegenseitig behindern, wird hier keine Anfrage innerhalb der Zeitbeschränkungen fertig. Dieses Verhalten zeigt sich auch in den Experimente in Abschnitt 8.8 (Seite 126).

2.2.6 Werkzeuge zur Lasterzeugung

Es gibt eine Reihe von Werkzeugen die ein Webserver Setup unter Last setzen können. Diese können grob in Lastgeneratoren und Trace-Replay-Tools unterschieden werden. Dabei werden Lastgeneratoren typischerweise dazu genutzt möglichst viel Last zu erzeugen, um so z.B. die Grenzen eines Dienstes zu erkunden. Trace-Replay-Tools hingegen werden genutzt, um einen bestimmten Verlauf wiederholt einzuspielen und so bestimmte Szenarien unter verschiedenen Bedingungen testen zu können (z.B. verschiedenen Scheduling-Algorithmen). Dadurch können diese verglichen oder optimieren werden. Verwendet werden in dieser Arbeit drei verschiedene solche Werkzeuge, `http_load`, `servload` und `SPECTRE`.

Lastgenerator: http_load Das einfachste der verwendeten Werkzeuge ist `http_load`. Dieses bekommt eine Datei mit URLs als Eingabe und kann dann mit beliebiger Intensität eine konfigurierte Adresse mit Anfragen aus der Eingabedatei unter Last setzten. Die Intensität lässt sich dabei entweder in Anfragen die pro Sekunde verschickt werden oder in Anfragen die parallel bearbeitet werden sollen angeben. Zusätzlich gibt man an, ob eine gewisse Gesamtanzahl an Anfragen bearbeitet oder ob die Last für eine gewisse Zeit aufrecht gehalten werden soll.
Ähnliche Lastgeneratoren sind z.B. `httperf` [74] oder `ab` [75].

Trace-Replay-Tool: servload `servload` [76] ist ein am Lehrstuhl entwickeltes Trace-Replay-Tool welches dazu in der Lage ist einen Trace im *Common Logfile Format* [77, 78] wieder einzuspielen und wurde im Rahmen des *salbnet*-Projektes [79] von Jörg Jung (ehemals Zinke) entwickelt. Für die Verwendung muss nur die Datei mit dem wieder einzuspielenden Trace und die Zieladresse des zu testende Dienstes angegeben werden (z.B. den *Dispatcher* eines LVS). Am Ende ausgegeben werden eine Reihe von Metriken u.a. Durchsatz, Verbindungszeiten und aufgetretene Zeitüberschreitungen.

Trace-Replay-Tool: SPECTRE Der *Simple Pyhton basEd Common Trace REplayer (SPECTRE)* ist eine Eigenentwicklung bei dem es sich, ebenso wie bei `servload`, um ein Trace-Replay-Tool handelt. Die Eigenentwicklung wurde im Verlaufe der Arbeit durchgeführt, da bestehende Werkzeuge wie `http _load` [80], `httperf` [74], `ab` [75] oder `servload` [81] nicht über benötigte Protokollierungsfunktionen verfügen. Neben den meisten Metriken die auch `servload` am Ende ausgibt, verfügt `SPECTRE` über die Eigenschaft nicht nur zu erfahren, wie viele Timeouts es gab, sondern er Protokolliert detailliert, wann ein Timeout auftrat und welche Anfrage ihn verursachte. Zusätzlich kann für jede Anfrage nachträglich die Bearbeitungsdauer ermittelt werden. Diese umfangreichen analytischen Eigenschaften kosten allerdings Ressourcen, so dass `SPECTRE` kein hochperformantes Werkzeug darstellt. Die Korrektheit von `SPECTRE` wurde mit Plausibilitätstests und dem exemplarischen Vergleich mit Ergebnissen von `servload` überprüft.

2.3 Eigenschaften und Klassifikation von Algorithmen

Im nachfolgenden Kapitel werden, vor allem im Zusammenhang mit theoretischen Arbeiten, oft verschiedene Eigenschaften von Algorithmen genannt, die auch zu deren Klassifikation genutzt werden können. Für ein besseres Verständnis bei der Erläuterung der Arbeiten und einer Einordnung der vorliegenden Arbeit werden die auftretenden Begrifflichkeiten nun vorgestellt.

2.3.1 Online- und Offline-Algorithmen

Die Menge aller Algorithmen lässt sich in Online- oder Offline-Algorithmen einteilen. Jeder Algorithmus ist damit entweder ein Online- oder ein Offline-Algorithmus.

Online-Algorithmen Als Online-Algorithmus werden Algorithmen bezeichnet, die keine vollständigen Informationen über das von ihnen zu lösende Problem besitzen müssen und ihre Strategie zur Laufzeit an neue Gegebenheiten anpassen können. Diese Art von Algorithmen sind daher

besonders gut für Aufgaben in einem laufenden Betrieb geeignet. Hier ist häufig die Reihenfolge und die Art der Eingaben von Beginn an unklar.

Da Entscheidungen auf der Basis von unvollständigem Wissen getroffen werden, kann in ungünstigen Situationen eine sehr mangelhafte Lösung das Resultat eines Online-Algorithmus sein.

Ein typisches Beispiel für einen Online-Algorithmus ist der Scheduler eines Betriebssystems. Dieser bekommt im Laufe seiner Ausführung, nach und nach, neue Arbeit zugewiesen. Die verfügbaren Ressourcen muss er dann, möglichst optimal, dem aktuellen Bedarf nach, verteilen.

Ein berühmter Vertreter von Problemen, die nur online zu lösen sind, ist das *Ski-Rental* Problem. Dieses Problem befasst sich mit der Frage, ob, und wenn ja, wann, man bei einem Ski-Urlaub Skier kaufen oder nur ausleihen sollte. Dabei sollen zwar die Kosten minimiert werden, allerdings sind im Vorfeld viele Faktoren unbekannt. Unter anderem, wie viele Tage Ski-gefahren wird, ob der Person der Sport gefällt, ob solch ein Urlaub wiederholt wird etc. Wären diese Faktoren bekannt, könnte sehr einfach vorher entschieden werden, ob kaufen oder ausleihen günstiger ist. So muss allerdings durch einen Online-Algorithmus entschieden werden, ob ausgeliehen oder gekauft wird.

Eine umfangreiche Sammlung von Algorithmen, Problemen und deren Analysen (einschließlich dem *Ski-Rental* Problem) kann in der *Encyclopedia of Algorithms* [82] gefunden werden.

Das in dieser Arbeit verwendete ESS *CHERUB* ist für den Einsatz in einem Produktivsystem konzipiert, in dem nicht von vornherein alle Informationen bekannt sind (z.B. Last der einzelnen Maschinen). *CHERUB* gehört daher in die Klasse der Online-Algorithmen.

Offline-Algorithmen Im Gegensatz zu Online-Algorithmen, müssen bei Offline-Algorithmen bereits alle Informationen über das zu lösende Problem bekannt sein. Der Vorteil von Offline-Algorithmen liegt darin, dass Probleme von ihnen optimal gelöst werden können. Das vollständige Wissen über das Problem macht die Berechnung einer optimalen Lösung möglich, da es nicht zu unvorhergesehenen Ereignissen kommen kann. Dies bedeutet nicht, dass jeder Offline-Algorithmus automatisch auch optimale Lösungen erzeugt.

Ein Anwendungsfall, der häufig in theoretischen Arbeiten untersucht wird, sind Scheduler in geschlossenen Systemen. Hier sind bereits alle Aufgaben, die jemals erfüllt werden müssen, bekannt. Daher kann eine optimale Belegung der Ressourcen berechnet werden, denn alle hierfür benötigten Informationen sind bereits vorhanden.

2.3.2 Optimale, Approximative und Kompetitive Algorithmen

Ein Ziel dieser Arbeit ist, den Energieverbrauch in einem Cluster zu minimieren. Es handelt sich somit um ein Minimierungsproblem bzw. ein Optimierungsproblem. Es gibt genauso Maximierungsprobleme bzw. häufig kann ein Problem so umformuliert werden, dass aus einem Minimierungsproblem ein Maximierungsproblem, und umgekehrt, wird. Das Ziel dieser Arbeit könnte zum Beispiel auch als Maximierungsproblem beschrieben werden, in dem der erzielte Gewinn des Clusterbetreibers maximiert werden soll. Bei einem Maximierungsproblem gelten die in diesem Abschnitt vorgestellten Formeln normalerweise umgekehrt, so dass aus einer Min-Funktion eine Max-Funktion wird, oder aus einem $\leq$ ein $>$ etc. Die folgenden Erklärungen und Definitionen sind, unter anderem, an das Buch *Online Computation and Competitive Analysis* [83] angelehnt in welchem sich viele weitere Online-Probleme finden, die vorgestellt und analysiert werden.

Optimale Algorithmen Ein Algorithmus *ALG* berechnet für ein Optimierungsproblem $\mathcal{P}$ und eine beliebige legale Eingabe I eine zulässige Lösung $ALG(I) = \mathcal{O} \in \mathcal{O}_{\mathcal{I}}$. Dabei sei $\mathcal{O}_{\mathcal{I}}$ die Menge aller validen Ausgaben, bei Eingabe von I. Die Kosten [12] von $\mathcal{O}$ können mit Hilfe der Kostenfunktion $ALG_C(I) = C(I,O)$ berechnet werden. Ein optimaler Algorithmus OPT muss für alle Eingaben I Formel 2.5 erfüllen.

$$OPT_C(I) = \min_{O \in \mathcal{O}_{\mathcal{I}}} (C(I,O)) \tag{2.5}$$

Umgangssprachlich muss der Algorithmus für alle Eingaben stets die valide Ausgabe mit den geringsten Kosten liefern.

Approximative Algorithmen Ein Algorithmus *ALG* für ein Optimierungsproblem $\mathcal{P}$ ist ein *asymptotisch c-approximativer* Algorithmus, wenn es für alle validen Eingaben eine Konstante $\alpha \geq 0$ gibt, so dass Formel 2.6 erfüllt ist.

$$ALG(I) - c * OPT(I) \leq \alpha \tag{2.6}$$

Bei $\alpha = 0$ ist die Rede von einem *c-approximativen* Algorithmus. Für $\alpha = 0$ würde ein 2-approximativer Algorithmus immer mindestens eine Lösung finden, die halb so gut ist wie die Lösung eines optimalen Algorithmus, der dasselbe Problem löst. Ein approximativer Algorithmus löst somit ein Problem näherungsweise. Diese Art von Algorithmen finden häufig dann Verwendung, wenn eine exakte oder optimale Lösung nicht effizient berechnet werden kann. Man tauscht dann Laufzeit gegen Optimum der Lösung. Ein approximativer Algorithmus bietet sich auch an, wenn es wichtiger ist, überhaupt eine Lösung in akzeptabler Zeit zu berechnen. Einen Algorithmus für eine optimale Lösung mag es dann zwar geben, dieser würde aber womöglich für die Berechnung zu lange brauchen.

Kompetitive Algorithmen Die Klasse der *c-kompetitiven* Algorithmen muss genau die gleiche Voraussetzung erfüllen wie die *(asymptotisch) c-approximativen* Algorithmen, mit der Einschränkung, dass der Algorithmus online berechnet werden muss. Das erreichte c kann dann als Wettbewerbs-Verhältnis angegeben werden. Anstatt möglichst nah an eine optimale Lösung, so wie bei den *asymptotisch c-approximativen* Algorithmen, zu kommen, wird hier versucht, möglichst nah an eine potenzielle Offline-Lösung eines Problems zu kommen. Da Offline-Algorithmen, wie bereits erklärt, alle Informationen im Voraus kennen, sind deren Lösungen immer mindestens gleich gut zu einem *c-kompetitiven* Algorithmus. Daraus resultiert ebenfalls, dass niemals $c < 1$ gelten kann. Dies gilt auch für *asymptotisch c-approximative* Algorithmen.

2.4 Fazit

Von den vorgestellten Themen dieses Kapitels, werden für die weitere Arbeit Teile aus IPMI und ACPI, sowie die gesamte Software verwendet. IPMI wird genutzt, um Rechner zu Starten und

[12] Die Kosten sind ein Maß, wie gut die Lösung eines Algorithmus ist. Dabei müssen die Kosten aber nicht in einem Zusammenhang mit der Laufzeit des Algorithmus stehen. Die Lösung eines Algorithmus kann somit minimale Kosten aufweisen, aber trotzdem eine schlechte Laufzeit besitzen.

Herunterzufahren und aus ACPI wird der Suspend-to-RAM Zustand näher untersucht. Für die Klassifikation anderer Arbeite, wird die Unterscheidung zwischen Online und Offline Algorithmen vorgenommen.

3 Verwandte Arbeiten

Die verwandten Arbeiten wurden in theoretische und praktische Arbeiten aufgeteilt. Bei den theoretischen Arbeiten handelt es sich um all jene, die sich in erster Linie mit Modellen und deren Analysen oder grundlegenden Konzepten beschäftigt haben. Die praktischen Arbeiten umfassen vor allem jene Arbeiten, die Messungen an realen Maschinen durchgeführt haben. Innerhalb der beiden Kategorien wurde versucht, die Arbeiten möglichst zeitlich, thematisch oder nach Arbeitsgruppe zu sortieren.

3.1 Theoretische Arbeiten

Titel: *Scheduling for Reduced CPU Energy*
Kennwörter: *Simulation, slow-and-steady, Scheduling, Vorhersage, gemixte Arbeitslast*
Eine der ersten und häufig zitierten Arbeiten, die das Energiesparen thematisiert, ist von Weiser et al. [12] und bereits aus dem Jahre 1994. Die Arbeit versucht, mit geschicktem Scheduling Energie zu sparen. Sie untersucht drei verschiedene Scheduling-Verfahren, *OPT*, *FUTURE* und *PAST*, welche sich in ihrem Wissen über die zukünftige Last unterscheiden. *OPT* besitzt vollständiges Wissen über die Last, *FUTURE* kann für ein gewisses Zeitfenster in die Zukunft sehen und *PAST*, welches für ein Zeitfenster in die Vergangenheit sieht, nimmt an, dass sich zukünftige Zeitfenster ebenso verhalten. Da *PAST* als einziges Verfahren kein Offline-Wissen verwendet und es daher als einziges Verfahren in der Realität Anwendung finden kann, konzentrieren sich die Autoren auf dieses Verfahren. Ihre Verfahren versuchen alle Zeiten, in der die CPU inaktiv ist, zu verhindern, indem mit DVFS Spannung und Frequenz so angepasst werden, dass anstehende Aufgaben gestreckt werden und so die inaktiven Perioden von den nun länger dauernden Aufgaben gefüllt werden. Als Arbeitslast für ihre Experimente dienen 32 Datensätze mit verschiedenen Desktop-Aktivitäten, welche von den Autoren aufgezeichnet wurden und normale Aktivitäten eines Arbeitsrechners umfassen (E-Mails abrufen, Dokumente verfassen oder Software entwickeln). Konstruiert wurde ein Simulator, der diese aufgezeichneten Aktivitäten verwendet und die Scheduling-Verfahren auf sie anwendet. Da *PAST* dafür sorgen kann, dass Aufgaben zu stark verlangsamt werden, muss in solchen Fällen mit erhöhter Spannung und Frequenz verlorene Zeit aufgeholt werden. Dieser Fall führt dazu, dass die zum Aufholen benötigte zusätzliche Energie die zuerst eingesparte Energie verbraucht oder sogar zu einem Mehrverbrauch führt. Ihre erzielten Ersparnisse mit *PAST* reichen von 5 % bis 75 %.
Kritik: Die größte Kritik an der Arbeit liegt im Umfang ihres Modells. Das Modell betrachtet die CPU als einzigen Energie-Verbraucher und nimmt an, dass keine Energie verbraucht wird, wenn diese inaktiv ist. Es gibt zwar Systeme, zumeist eingebettete Systeme, bei denen die CPU den Energieverbrauch dominiert, aber die meisten Systeme besitzen viele andere Komponenten (Speicher, Festplatte, Grafikkarte, Netzwerk, Netzteil), die einen ähnlich hohen Verbrauch aufweisen.
Auch die verwendete Arbeitslast schränkt die Aussagekraft der Arbeit ein. Die Aufgaben, die z.B.

ein Server-System zu behandeln hat, unterscheiden sich signifikant von Aufgaben eines Bürorechners.

Titel: *Comparing Algorithm for Dynamic Speed-setting of a Low-power CPU*
Schlüsselwörter: *Simulation, slow-and-steady, Scheduling, Vorhersage, gemixte Arbeitslast*
Govil et al. [13] setzen direkt auf die Arbeit von Weiser et al. [12] auf und verwenden denselben, leicht angepassten Simulator und dieselben Datensätze. Sie vergleichen das *PAST*-Verfahren aus [12] mit sechs eigenen Implementierungen, die sich durch Vorhersage- und Glättungs-Verfahren bezüglich der angestrebten Spannung und Frequenz unterscheiden. Die Autoren kommen dabei zu dem Schluss, dass nur eines ihrer Verfahren ein zufriedenstellendes Ergebnis liefert. Dieses nutzt lediglich zwei beobachtete Werte, um auf die zukünftige Last zu schließen. Es lohnt sich, laut Autoren, nicht besonders aufwendige Verfahren zu verwenden.
Kritik: Da weder an der Arbeitslast, noch an dem Modell von Weiser et al. Änderungen vorgenommen wurden, gelten für diese Arbeit dieselben Kritikpunkte wie für die von Weiser et al.

Titel: *A Scheduling Model for Reduced CPU Energy*
Schlüsselwörter: *Analytisch, slow-and-steady, Scheduling,* 2^2*-kompetativ (Energie)*
Yao et al. haben mit [14] eine sehr analytische Arbeit veröffentlicht. Sie beziehen sich auf die beiden Arbeiten von Weiser et al. [12] und Govil et al. [13] und verwenden ebenfalls den Ansatz, Energie durch ein möglichst optimales Scheduling-Verfahren einzusparen. Das von den Autoren verwendete Modell benutzt ein festes Zeitintervall, in dem eine bestimmte Anzahl an Jobs eintreffen. Diese Jobs werden durch drei Attribute spezifiziert, ihre Ankunftszeit, ihre Fristen und ihre benötigten CPU-Zyklen zur Fertigstellung. Aus diesen Parametern wird ein Ablaufplan errechnet, bei dem die CPU immer bei minimal nötiger Frequenz arbeitet und trotzdem alle Fristen eingehalten werden. Das Einhalten der Fristen ist immer möglich, da angenommen wird, dass unendliche Rechenleistung zur Verfügung steht. Es handelt sich zwar um einen Offline-Algorithmus, dieser kann allerdings, laut Autoren, durch verschiedene Heuristiken zu einem Online-Algorithmus erweitert werden. Die Online-Variante des Algorithmus ist, laut Autoren, 2^2-kompetitiv in Bezug auf Energie.
Kritik: Bei dieser Arbeit ist vor allem die Vorbedingung der unendlichen Rechenleistung eine zu starke Vereinfachung, um praktischen Nutzen aus der Arbeit zu ziehen. Zusätzlich gilt für die Arbeit, genau wie auch für [12, 13], dass sie nur Gültigkeit in Systemen hat, in der die CPU den Energieverbrauch klar dominiert.

Titel: *Dynamic Speed Scaling to Manage Energy and Temperature*
Schlüsselwörter: *Analytisch, slow-and-steady, Scheduling,* $\mathcal{O}(1)$*-approximativ (Temperatur),* $\mathcal{O}(1)$*-kompetativ (Energie)*
Eine auf [14] aufbauende Arbeit ist von Bansal et al. [84]. Sie untersucht, ob der *YDS*-Algorithmus (nach seinen Autoren Yao, Demers, Shenker) aus [14] nicht nur hinsichtlich Energie, sondern auch hinsichtlich Temperatur optimal ist. Die Autoren stellen fest, dass dies der Fall ist. Die von Yao et al. vorgeschlagene Online-Variante ist, laut Autoren, allerdings nicht optimal hinsichtlich Temperatur. Es wird ein eigener Algorithmus vorgestellt, basierend auf den Vorgaben des Modells von Yao et al. Ihr entwickelter *BKP*-Algorithmus (nach den Autoren Bansal, Kimbrel, Pruhs) ist, laut eigener Aussage, $\mathcal{O}(1)$-approximativ hinsichtlich Temperatur und $\mathcal{O}(1)$-kompetitiv hinsichtlich optimalem Energieverbrauch.

Kritik: Da auch Bansal et al. das *slow-and-steady*-Prinzip verwenden, ist ihre Strategie nicht im vorliegenden Einsatzgebiet zu gebrauchen.

Titel: *Optimal Power-Down Strategies*
Schlüsselwörter: *Analytisch, slow-and-steady, Scheduling,* $3+2\sqrt{2}$*-kompetitiv*
In [85] wird von Augustine et al., nach eigenen Angaben zum ersten Mal, ein Online-Powersaving-Algorithmus beschrieben, der für Prozessoren mit mehr als zwei Zuständen konzipiert ist und nicht von additiven Transitionskosten ausgeht. Additive Transitionskosten gelten, wenn $d_{i,j} + d_{j,k} = d_{i,k}$ erfüllt ist, wobei $d_{i,j}$ der Energieverbrauch für den Zustandsübergang des Prozessors von Zustand i nach Zustand j ist. Laut Autor gilt die Additivität in den meisten Fällen nicht. Der Algorithmus orientiert sich dabei am in Abschnitt 2.3.1 (Seite 33) erwähnten *Ski-Rental* Problem und senkt die Frequenz der CPU für ein kommendes Intervall immer dann, wenn in dem vergangenen Intervall eine zu hohe Frequenz verwendet wurde. Die Autoren zeigen, dass es eine optimale Strategie als Offline-Variante gibt und beweisen, dass es auch eine $3+2\sqrt{2}$-kompetitiv Online-Strategie geben muss. Daraus resultierend wird ein auf binärer Suche basierender, annähernd optimaler, deterministischer Algorithmus angegeben.
Kritik: Wie in den bisherigen vorgestellten Arbeiten auch, wird hier nur die CPU als Energie-Verbraucher betrachtet.

Titel: *Nonclairvoyant Speed Scaling for Flow and Energy*
Schlüsselwörter: *Analytisch, slow-and-steady, Scheduling,* $\mathcal{O}(1)$*-competitive*
Eine weitere Arbeit, die einen Online-Algorithmus für Prozess-Scheduling vorstellt, ist von Chan et al. [86]. Der Algorithmus *Last Arrival Processing Time* (LAPS) wird vorgestellt und es wird gezeigt, dass er $\mathcal{O}(1)$-competitive ist. LAPS geht so vor, dass er bei jedem Eintreffen und Beenden eines Jobs den Prozessor auf eine neue Frequenz einstellt. Dabei werden die Annahmen getroffen, dass ein Prozessor beliebig getaktet werden kann und auch eine unendliche Frequenz zur Verfügung steht. Es wird ebenfalls davon ausgegangen, dass Jobs ohne weitere Kosten pausiert werden können und andere Jobs an deren Stelle berechnet werden können (diese Eigenschaft wird als Präemptives-Multitasking bezeichnet).
Kritik: Auch in dieser Arbeit werden Annahmen getroffen, die zu starke Abstraktionen von der Realität sind (unendlicher Prozessorleistung, CPU einziger Energie-Verbraucher). Auch eine beliebig einstellbare Frequenz ist in CPUs nicht möglich. Wie in Abschnitt 2.1.1 (Seite 9) bereits erwähnt, besagt der ACPI Standard sogar, dass die Anzahl der P-States (Anzahl der verfügbaren Frequenzen) 16 nicht überschreiten soll.

Titel: *Online strategies for dynamic power management in systems with multiple power-saving states*
Schlüsselwörter: *Simulation, slow-and-steady, Schwellwerte & Vorhersage, 2-kompetitiv,* $e/(e-1)$*-kompetitiv, Festplatte*
Eine Arbeit, in der eine Evaluation eines Stromspar-Algorithmus vorgestellt wird, ist von Irani et al. [87]. In dieser Arbeit werden zwei Algorithmen vorgestellt, die für Systeme mit mehreren Energie-Zuständen konzipiert sind. Diese sind der deterministische *lower envelope algorithm (LEA)* und der *probabilistic lower envelope algorithm (PLEA)*.
LEA ist ein 2-kompetitiver Algorithmus, welcher Geräte Schritt für Schritt in einen immer sparsameren Zustand schaltet. *PLEA* erweitert *LEA* mit einem probabilistischen Aspekt. Es wird die

Wahrscheinlichkeit für die Länge von inaktiven Perioden berücksichtigt. Durch dieses Wissen kann schneller in den effektivsten Zustand gewechselt werden (bei langen Perioden wird schneller in tiefere Zustände geschaltet und umgekehrt). *PLEA* wird als $e/(e-1)$-kompetitiv (~1,582) bewiesen.Da das Wissen der nötigen Wahrscheinlichkeiten in der Praxis nicht vorhanden ist, wird von Irani et al. ein auf *PLEA* basierender Online-Algorithmus implementiert. Der *Online probability based algorithm (OPBA)* nutzt die letzten n Werte, um in festen Abständen auf die aktuellen, inaktiven Perioden zu schließen und darauf basierend, den besten Zustand zu wählen. In einem Experiment werden die Algorithmen verglichen. Verwendet wird ein Trace mit 400.000 verschiedenen Festplatten-Zugriffen. Als ideales Fenster ergibt sich in den Messungen $n = 50$. Kleinere Fenster führen dazu, dass zu wenig Daten für eine gute Verteilung zur Verfügung stehen und bei zu großen Größen wird die aktuelle Last-Situation nicht genügend berücksichtigt. Im Vergleich zu anderen Algorithmen zeigt sich *OPBA* als bester Kompromiss aus gutem kompetitiven Verhältnis und geringer Latenz (der Metrik ihres Experiments).
Kritik: Die entwickelten Algorithmen scheinen für einzelne Komponenten zu funktionieren. Im Gegensatz zu einer Festplatte (wie im Experiment verwendet), hat ein Rechner, der aus vielen Komponenten besteht, andere Anforderungen und Merkmale (bspw. viel höhere Dauer für die Rückkehr aus einem Ruhezustand). Es kann nicht davon ausgegangen werden, dass die verwendete Strategie ebenfalls für ganze Maschinen/Cluster anwendbar ist.

Titel: *Towards Energy-aware Scheduling in Data Centers Using Machine Learning*
Schlüsselwörter: *Simulation, sprint-to-halt, Scheduling, Schwellwertverfahren, gemixte Arbeitslast, Cluster Computing*
Berral et al. [15] beschäftigen sich mit Algorithmen zum An- und Abschalten von Maschinen. Ihre Studie untersucht einfache, schwellwertbasierte Verfahren. Genauer haben sie sich mit der Frage der optimalen Schwellwerte auseinandergesetzt. Ihre Studie kommt zu dem Ergebnis, dass ein Unterlast-Schwellwert, welcher zum Abschalten von Maschinen verwendet wird, bei 30 % Systemauslastung liegen sollte. Ein Überlast-Schwellwert, welcher zum Anschalten von neuen Maschinen verwendet wird, sollte hingegen bei 60 % Systemauslastung liegen.
Kritik: Obwohl auch andere Arbeiten, wie zum Beispiel [17], sehr ähnliche Grenzen als optimal betrachten, fehlt der Bezug zur Bootzeit der verwendeten Hardware. Ein optimaler Schwellwert hängt vor allem davon ab, wie schnell neue Ressourcen verfügbar gemacht werden können. Bei einem Setup mit SSD-Festplatten, in Kombination mit STD/STR-Techniken, kann ein Bootvorgang erheblich beschleunigt werden und ein Überlast-Schwellwert könnte großzügiger ausfallen.

Titel: *Dynamic Energy-aware Capacity Provisioning for Cloud Computing Environments*
Schlüsselwörter: *Simulation, sprint-to-halt, feste Arbeitslast, Cloud Computing*
Neuere Arbeiten versuchen die Verfahren zum An- und Abschalten auch auf Cloud Computing zu übertragen. In [16] wird von Zhang et al. ein Verfahren vorgestellt, in dem durch Zeitreihenanalyse die optimale Anzahl an aktiven Rechnern für eine gegebene Last gesucht wird. Dabei werden auch die aktuellen Strompreise berücksichtigt. Ihr Verfahren geht davon aus, dass alle Maschinen homogen sind. Zur Überprüfung wird eine Simulation durchgeführt. Als Arbeitslast dient ein realer Trace (engl. Verlaufsprotokoll) von Google [88], der von einem ca. 12.000 Maschinen-Cluster stammt und etwa den Zeitraum von einem Monat umfasst. Abhängig von der erlaubten Verzögerung der Anfragen, wird zwischen 50 % (auf Kosten von signifikanten Verzögerungen) und 18,5 % (Verzögerung von ca. 10 Sekunden) Energie gespart.

Kritik: Die verursachten Verzögerungen sind für einen praktischen Gebrauch zu hoch und müssen deutlich verringert werden.

Titel: *Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis*
Schlüsselwörter: *Analyse, feste Arbeitslast*
Eine weitere Arbeit, die den bereits erwähnten Google Trace [88] verwendet, ist von Reiss et al. [89]. Die Arbeit stellt eine sehr detaillierte Analyse des Traces dar. Es fallen zwei Aspekte auf. Zum einen ist das zur Aufzeichnung verwendete Cluster sehr heterogen (6 größere homogene Mengen von Maschinen) und zum anderen unterscheidet sich die reale Auslastung der Maschinen grundlegend von der Reservierung der Ressourcen. Die Arbeit beschreibt, dass bei einer Reservierung von etwa 80 % des Speichers und teilweise mehr als 100 % der CPU, fast durchgängig nur etwa 50 % der angeforderten Ressourcen auch tatsächlich genutzt werden.

Titel: *Scalability and Performance Management of Internet Applications in the Cloud*
Schlüsselwörter: *Messung & Simulation, sprint-to-halt, Cloud Computing, Schwellwerte & Vorhersage, feste Arbeitslast*
In der Dissertation von Wesam Dawoud [17] geht es ebenfalls um die Skalierung und Performance von Diensten in der Cloud. Es werden Verfahren zur vertikalen Skalierung [1] und zur Vermeidung von gegenseitiger Beeinflussung in öffentlichen *Infrastructure as a Service* (IaaS) Umgebungen vorgestellt. Es werden aber auch für diese Arbeit interessante Verfahren zur horizontalen Skalierung vorgestellt und verglichen. Dabei wird über reale Messungen am Amazon EC2 Dienst [90] ein Energie-Modell bestimmt. Dieses Modell wird zusammen mit dem Simulator *CloudSim* [2] [91] verwendet. Für *CloudSim* wurde zusätzlich ein Modul, *ScaleSim*, implementiert. Dieses Modul sorgt dafür, dass der Simulator Skalierungen durchführen kann, wozu er vorher nicht in der Lage war. Als Arbeitslast wird der oft verwendete und öffentlich zugängliche Trace der Fussballweltmeisterschaft aus dem Jahre 1998 in Frankreich [92] modifiziert und verwendet. Getestet werden verschiedene Verfahren, darunter ein einfaches schwellwertbasiertes Verfahren, ein verbessertes schwellwertbasiertes Verfahren und fünf weitere, mehr oder weniger intelligente, Vorhersageverfahren (darunter ihr eigenes). Ihr eigenes Verfahren schneidet in ihren Metriken *Service Level Objectiv*-Verletzungen am besten ab, aber bei der *Erhöhung der Kosten* liegt es nur auf dem vorletzten Platz.
Kritik: Leider wird die Modifikation der Arbeitslast unzureichend beschrieben und es macht den Eindruck, als seien ein paar Referenzen durcheinander geraten, was die Verständlichkeit zusätzlich erschwert. Das größere Problem ist allerdings die Dauer der Vorhersage aller Verfahren. Es wird ausschließlich eine Periode, welche 1 Minute umfasst, in die Zukunft prognostiziert. Um tatsächlich Maschinen ab- oder anzuschalten, muss in ihrem System der Schwellwert fünf Mal in Folge verletzt werden. Das System besitzt somit eine Reaktionszeit von mindestens 5 Minuten. Obwohl *CHERUB* ebenfalls einen Verzögerungsmechanismus implementiert, wird dieser in der vorliegenden Arbeit so gesetzt, dass das System eine Reaktionszeit von höchstens einer Minute besitzt. Die Reaktionszeit kann sich zusätzlich verlängern, da nach einem An- oder Abschalten einer Maschine

[1] Unter vertikalem Skalieren wird die Leistungssteigerung eines Systems bezeichnet, wenn dafür bestehende Komponenten verbessert werden (mehr Speicher, bessere CPU, weitere Beschleunigerkarten etc.). Im Gegensatz dazu wird bei der horizontalem Skalieren die Leistungssteigerung erzielt, indem weitere Rechner einem Setup hinzugefügt werden.

[2] *CloudSim* wird in Abschnitt 8 (ab Seite 109) vorgestellt, aber aus verschiedenen Gründen nicht genutzt.

weitere 5 Minuten vergehen müssen, bevor wieder eine Aktion durchgeführt werden kann. Dies soll mögliches Zustands-Flattern (*Flapping*, siehe hierfür Abschnitt 4.2) verringern, sorgt aber bei starkem Lastanstieg für ungewollte Verzögerungen. Ein System mit so hoher Reaktionszeit ist für den Einsatz im SLB-Bereich nicht geeignet. Ein weiteres Problem ist, dass das System nicht feststellen kann, wie viele Maschinen tatsächlich in der Zukunft benötigt werden. Es kann lediglich eine feste Anzahl an Maschinen konfiguriert werden, welche im Falle einer Überlast angeschaltet werden. Der Autor führt Messungen mit den Werten von 1-4 Maschinen durch, doch diese fallen nicht zufriedenstellend aus. Beispielsweise werden bei 4 Maschinen sehr häufig zu viele Maschinen angeschaltet, die dann wieder abgeschaltet werden müssen.

Titel: *Power-Efficient Load Distribution in Heterogeneous Computing Environments*
Schlüsselwörter: *Simulation, sprint-to-halt, Scheduling, feste Arbeitslast, Cluster Computing*
Die Arbeit von Lenhardt et al. [18] beschäftigt sich mit der intelligenten Verteilung von Last. Ressourcen, die nach der Verteilung ungenutzt sind, sollen abgeschaltet werden. Sie vergleichen vier Scheduling-Verfahren miteinander, von denen eines ihr eigenes ist. Bei den Verfahren, die verglichen werden, handelt es sich um *Absolute load balancing* (de facto Round-Robin), *Relative load balancing* (de facto Weighted Round-Robin), *Best performance to power ratio first (bpprf)* (Knoten mit der höchsten Effizienz werden zuerst belegt) und ihrem eigenen *Adaptive load distribution (ald)*. Bei *ald* wird für alle Rechner zusätzlich ihre Effizienz bei 10 %, 20 % ... 100 % Auslastung mit in Betracht gezogen. Die Evaluation des Verfahrens wird anhand von Simulationen durchgeführt, wobei vier verschiedene Clusterkonfigurationen verwendet werden. Ihre Arbeitslast wird pro Konfiguration mit jeweils allen vier Verfahren getestet. Bei drei von vier Konfigurationen gewinnt *ald* knapp vor *bpprf* und die beiden einfachen Verfahren liegen, bis auf eine Ausnahme, jeweils hinten.
Kritik: Leider wird über die Arbeitslast, die verwendet wird, keine weitere Auskunft gegeben. Dadurch ist nicht klar, ob diese nur für gewisse Situationen repräsentativ ist oder allgemeingültig ist bzw. ein breites Spektrum an Situationen abdeckt. Es ist auch nicht klar, ob durch ihren Aufbau gewisse Verfahren bevorzugt werden. Der vorgeschlagene *ald* Algorithmus besitzt zudem eine signifikant schlechtere Performance. Die Autoren schlagen vor, alle möglichen Lastsituationen im Voraus zu berechnen, um diesen Nachteil auszugleichen. Vorteilhaft an *ald* ist, dass das Verfahren auch in heterogenen Umgebungen funktioniert. Dafür muss die Effizienz der einzelnen Maschinen aber hinterlegt sein. Diese Informationen zu beschaffen, ist häufig mit hohem Aufwand verbunden und kann in der Praxis dazu führen, dass ein Administrator lieber ein weniger aufwendigeres Verfahren verwendet.

Titel: *Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services*
Schlüsselwörter: *Simulation, sprint-to-halt, Scheduling, Vorhersage, gemixte Arbeitslast, Webserver*
In der Arbeit von Chen et al. [93] geht es um verbindungsorientierte Dienste, wie den *Windows Live Messenger*. Bei diesen Diensten besteht das Problem, dass das Abschalten einer Maschine zu vielen Neuverbindungen mit den noch bestehenden Maschinen führt. Dies wiederum führt zu einer extremen Belastung für die noch aktiven Maschinen, da eine Neuverbindung aufwendig ist. Die Arbeitsgruppe verwendet ein spezielles Scheduling-Verfahren, um dem entgegenzuwirken. Bei ihrem Verfahren werden alle aktiven Maschinen, bis auf eine, immer möglichst voll ausgelastet.

Die letzte Maschine hat im Falle eines Abschaltens nur sehr wenige Verbindungen und verursacht demzufolge weniger Last, wenn sie abgeschaltet wird. Verbindungsorientierte Dienste liegen außerhalb des Fokus dieser Arbeit und werden nicht näher in Betracht gezogen.

Titel: *PowerNap: Eliminating Server Idle Power*
Schlüsselwörter: *Analytisch, sprint-to-halt, STR, Cluster Computing*
In [10] stellen Meisner et al. zwei neue Konzepte, *PowerNap* und *RAILS*, vor. Bei *PowerNap* geht es um die Idee, dass jegliche Hardware nur zwei Zustände kennen sollte, aktiv und inaktiv. Der inaktive Zustand wird als extrem energiesparend angenommen. Zusätzlich wird vorausgesetzt, dass die Transition zwischen beiden Zuständen sehr schnell (Mikrosekundenbereich) möglich ist. Laut Autoren gibt es bereits für die wichtigen Komponenten eines Rechners Technologien, die die entsprechenden Voraussetzungen erfüllen. Ein großes Problem liegt laut Autoren bei aktueller Software. Aktuelle Systeme führen in regelmäßigen Abständen (Millisekundenbereich) Operationen aus, welche dafür sorgen würden, dass der aktive Zustand regelmäßig angenommen werden müsste und der Vorteil von *PowerNap* dadurch verringert würde. Neben der CPU wird als zweite Komponente mit großem Energiesparpotential das Netzteil ausgemacht. Aktuelle Netzteile besitzen eine schlechte Effizienz bei geringer Auslastung. Selbst 80 PLUS [94] zertifizierte Netzteile müssen nur eine gewisse Effizienz bei einer Auslastung von 20 %, 50 % und 100 % nachweisen [3]. Um dieses Problem zu umgehen, schlagen die Autoren ein *Redundant Array for Inexpensive Load Sharing (RAILS)* vor. Das Konzept sieht vor, anstelle weniger, teurer und leistungsstarker Netzteile eine Reihe von billigen Netzteilen (wie sie in Desktop PCs verwendet werden) in Clustern zu verwenden. Durch die Verwendung vieler schwacher Netzteile kann feingranular entschieden werden, wie viele Netzteile zur gegebenen Zeit benötigt werden. Durch dieses Vorgehen kann die schlechte Effizienz der Netzteile bei geringer Auslastung umgangen werden, indem immer so viele Netzteile Energie liefern, wie es nötig ist, damit alle bei hoher Effizienz arbeiten. Die Autoren erzielen Einsparungen von 80 % und 70 % in zwei verschiedenen Szenarien.
Kritik: Leider sind die Experimente, unter welchen die 80 % und 70 % Einsparung erreicht wurden, völlig unzureichend präsentiert. Es ist unklar, wie die verwendete Arbeitslast aussieht und es wird nicht klar gesagt, ob es sich um eine Simulation oder eine pure Analyse handelt. Die Autoren behaupten, vor allem in drei Bereichen große Einsparungen erzielen zu können. Die drei Bereiche betreffen die CPU, den Speicher und das Netzteil. Es wird nicht plausibel erklärt, weshalb das Konzept der Autoren notwendig ist, um die in diesem Bereich angegebenen Einsparungen zu ermöglichen. Vielmehr entsteht der Eindruck, dass diese Technologien bereits vorhanden sind und auch ohne *PowerNap* große Einsparungen in bestehenden Systemen verursachen würden. Auch *RAILS* ist nur eine feingranularere Umsetzung eines bestehenden Systems und das von den Autoren angegeben Einsparpotential scheint zu optimistisch zu sein.

Titel: *Power Consumption Estimation Models for Processors, Virtual Machines, and Servers*
Schlüsselwörter: *Analytisch, Literaturübersicht*
In [95] präsentieren Möbius, Dargie und Schill eine Übersicht über verschiedenste Arbeiten im Bereich der Energie-Modelle. Sie beginnen damit zu zeigen, auf welche Art und Weise in der Literatur Energieverbrauch in Rechnern gemessen oder abgeschätzt werden kann. Beim Messen

[3] Für das 80 PLUS Titanium Siegel muss zusätzlich eine sehr hohe Effizienz von 90 % bei einer Auslastung von 10 % nachgewiesen werden.

kann man beispielsweise Spezialhardware einsetzten um, wie z.B. in [96], möglichst exakt den Verbrauch einzelner Komponenten zu ermitteln. Beim Abschätzen ist es hingegen wichtig, welche Eingangs-Parameter das entsprechende Modell verwendet. Möglich ist z.B. die Verwendung bestimmter Zähler, die entweder direkt von der Hardware kommen (Anzahl CPU-Zyklen etc.) oder vom Betriebssystem bereitgestellt werden. Eine weitere Alternative ist das Trainieren solcher Modelle mit Hilfe von Benchmarks. Im Rest der Arbeit werden Arbeiten vorgestellt, die Modelle für den Energieverbrauch von einzelnen CPUs, virtuellen Maschinen und ganzen Servern modellieren. Dabei schlussfolgern die Autoren, dass es bei der Ermittlung der Abweichung der einzelnen Modelle auf drei Dinge ankommt. Darunter fallen die verwendeten Modell-Parameter, die Wahl, wie das Modell trainiert wurde (z.B. lineares vs. nicht-lineares Modell) und die verwendete Referenz. Bei der verwendeten Referenz ist beispielsweise zu beachten, dass für CPU- und VM-Modelle die Verlustleistung des Netzteils, die sich laut Autoren auf bis zu 35 % belaufen kann, nicht in die Referenz einfließen sollte.

3.2 Praktische Arbeiten

Die ersten praktischen Arbeiten zum Thema Energiesparen im SLB-Bereich wurden bereits 2002 publiziert. Vorher war Energiesparen ein Thema für mobile Geräte. In diesem Bereich sollte eine längere Betriebsbereitschaft erzielt werden, indem die Batterien weniger stark beansprucht werden.

Titel: *The Case for Power Management in Web Servers*
Schlüsselwörter: *Messung & Simulation, slow-and-steady, gemixte Arbeitslast, Webserver*
Viele Arbeiten kommen von einer Gruppe um Charles Lefurgy, einem Mitarbeiter der *Power-Aware Systems Group* von IBM. Diese haben in Bohrer et al. [97] damit begonnen, die Charakteristiken eines Webservers in Bezug auf dessen Energieverbrauch zu studieren. Dafür wurden drei verschiedene Arbeitslasten in Form von Traces verwendet, welche alle real aufgezeichnet wurden. Darunter der Mitschnitt der Last der Webserver, die die Webseite der Olympischen Winterspiele 1998 in Nagano auslieferten, eine Last aus dem Finanzbereich sowie eine Last von Squid-Proxyservern. Alle Traces wurden mit einem gepatchten `httperf` [74] wieder eingespielt. Mit Hilfe eines vierten, synthetischen Verlaufs, bei dem die Last graduell erhöht wurde, wurde der Verbrauch der einzelnen Komponenten des Testsystems gemessen. Dafür wurde an fünf verschiedenen Stellen im System der Stromverbrauch gemessen und beobachtet, dass der Hauptanteil des Verbrauchs durch die CPU verursacht wird. Bei dem Wiedereinspielen des Olympia- und Finanz-Traces wurden ausschließlich statische Abfragen verwendet, weshalb bei ihnen ebenfalls die CPU den höchsten Verbraucher darstellt. Bei dem Squid-Proxyserver-Trace hingegen ist auch der Verbrauch der Festplatten signifikant. Auf Basis dieser Daten und inspiriert von der Arbeit von Weiser et al. ([12]) wurde ein Simulator entwickelt, welcher den Stromverbrauch einer Webserver-CPU nachahmt. In den Simulator wurde die Möglichkeit der Nutzung von DVFS implementiert und für alle drei Traces getestet, wie viel Energie durch die Verwendung eingespart werden kann. Die Messungen erzielen dabei Einsparungen von bis zu 36 % für den Olympia-Trace und 22,7 % für den Squid-Proxyserver-Trace.
Kritik: Obwohl dynamische Seiten nur einen geringen Anteil am Olympia- bzw. Finanz-Trace ausgemacht haben, können diese durchaus einen wichtigen Unterschied darstellen. Außerdem

kann nicht davon ausgegangen werden, dass ein so geringer Anteil an dynamischen Seiten für heutige Webserver repräsentativ ist.

Titel: *On Evaluating Request-Distribution Schemes for Saving Energy in Server Clusters*
Schlüsselwörter: *Messung, sprint-to-halt, Schwellwerte & Vorhersage, gemixte Arbeitslast, Cluster Computing*
In einer darauf folgenden Arbeit der IBM-Gruppe, erstellt von Rajamani et al. [98], werden größere Cluster betrachtet. Diese Arbeit hat diverse Schnittpunkte mit der vorliegenden Arbeit. Sie nutzen ebenfalls einen Daemon in Zusammenhang mit LVS, um verschiedene Strategien zum An- und Abschalten von Ressourcen zu testen. Unter den Strategien befindet sich ein einfaches, schwellwertbasiertes Verfahren, aber auch Verfahren, die versuchen, aus den Charakteristiken der Arbeitslast Wissen über die zukünftige Last abzuleiten. Dabei werden drei verschiedene Ebenen von Wissen über die Last angenommen und getestet. Die Ebenen sind Wissen über das Maximum, Wissen über alle lokalen Maxima und vollständiges Wissen. In zwei verschiedenen Szenarien erreichen sie mit ihrem Verfahren 45 % und 26 % Einsparungen.
Kritik: Obwohl mit einer Charakterisierung der Arbeitslast ein grobes Lastprofil (z.B. Tag-Nacht-Zyklen) erkannt werden kann, so sind die drei Ebenen von Wissen über zukünftige Last kritisch zu betrachten. Vor allem das volle Wissen über die zukünftige Last ist in realen Umgebungen nicht vorhanden. Auch das Wissen über lokale Maxima oder das globale Maximum können für ein Lastprofil nur grob vorhergesagt werden. Im Falle eines Webservers, wie er in der vorliegenden Arbeit betrachtet wird, hilft grobes Wissen über die Last nicht aus, um auch kurzfristig auf Lastschwankungen reagieren zu können.

Titel: *Energy Management for Commercial Servers*
Schlüsselwörter: *Analytisch, technische Übersicht*
Weitere Arbeiten der Gruppe um Charles Lefurgy sind z.B. Lefurgy et al. [99], in der verschiedenste Techniken mit einem gewissen Maß an Energiesparpotential erläutert werden (Platzierung von Daten, Cache-Kohärenz, DVFS etc.).

Titel: *Autonomic Multi-agent Management of Power and Performance in Data Centers*
Schlüsselwörter: *Messung, sprint-to-halt, ESS, feste Arbeitslast, Webserver*
In Das et al. [100] stellt die IBM-Gruppe einen Multiagenten ESS vor. Es besteht aus verschiedenen Komponenten, die den Zustand des Clusters überwachen und versuchen, nur so viele Rechner angeschaltet zu haben wie nötig. Um die Entscheidungen über die Anzahl der momentan nötigen Rechner treffen zu können, wurden im Vorfeld Offline-Messungen durchgeführt. Bei diesen Messungen wurde für jede mögliche Anzahl aktiver Server jede mögliche Last, in Form von aktiven Clients, getestet. Mit diesen Daten wurde ein Modell gelernt, welches die optimale Anzahl von Rechnern für eine vorher konfigurierte Ziel-Performance kennt. Das so gelernte Modell weist allerdings in Randfällen ein unintuitives Verhalten auf (der Randfall tritt in ihrem Experiment nicht auf). In ihrem Experiment verwenden sie drei Server, auf denen jeweils Apache als Webserver läuft. Als Arbeitslast wird ein 96 Stunden langer Trace der NASA-Webseiten verwendet, welcher auf 9,6 Stunden gestaucht wird. Dabei erzielen sie durch das Abschalten von überflüssigen Rechnern an Werktagen Einsparungen von 20,1 % und am Wochenende 41,8 %.
Kritik: Die Autoren gestehen in der Arbeit ein, dass der vorherige Aufwand für die Ermittlung des Modells für größere Setups ungeeignet ist. Da nur drei Server verwendetet werden, lässt sich

auch nicht schlussfolgern, ob dieser Ansatz skaliert. Seltsam ist vor allem, dass das Modell schon für drei Server im Überlastbereich unintuitives Verhalten aufweist (es möchte Rechner abschalten). Die Autoren analysieren außerdem nicht, wie viel Energieersparnis in einem optimalen Fall möglich gewesen wäre, dadurch ist nicht klar, wie gut das System wirklich funktioniert.

Titel: *SHIP: Scalable Hierarchical Power Control for Large-Scale Data Centers / SHIP: A Scalable Hierarchical Power Control Architecture for Large-Scale Data Centers*
Schlüsselwörter: *Simulation, slow-and-steady, ESS, feste Arbeitslast, Cluster Computing*
Die neueren Arbeiten von Wang, Chen, Lefurgy und Keller [101, 102] stellen ein ESS namens *SHIP* vor. *SHIP* ist ein dezentrales ESS und für sehr große Datenzentren gedacht. Es verwendet Kontrolltheorie, um auf drei verschiedenen Ebenen (Rechenzentrum-Level, Rack-Level, Server-Level) den Energieverbrauch zu steuern. Anders als *CHERUB* ist es nicht das Ziel von *SHIP*, möglichst viel Strom zu sparen, sondern ein vorgegebenes Energie-Budget durchzusetzen. Der Administrator kann festlegen, wie viel Strom zur Verfügung steht und die Controller setzen die Vorgabe um. Eine detaillierte Analyse und Beschreibung des verwendeten Controllers und Vergleiche zu anderen Energie-Budget-Controllern können in Lefurgy et al. [103] gefunden werden. Auf unterster Ebene (Server-Level) wird DVFS verwendet, um den Energieverbrauch zu regulieren. Die oberen Ebenen verteilen nur ein gewisses Budget an das darunter liegende Level. Ihr System wird an einem Cluster mit 9 Rechnern (aufgeteilt in 3 Racks) getestet und die Vorgaben werden dabei präzise umgesetzt. *SHIP* wird außerdem mit dem zentralistischen ESS *Per-Rack* verglichen. Da *Per-Rack* versucht Energie-Budgets gleichmäßig zu verteilen, schneidet *SHIP* in dem gewählten Szenario, in dem 1/3 der Server nicht aktiv sind, wesentlich besser ab. *SHIP* ist eigentlich für sehr große Cluster gedacht, weshalb weiterhin eine Simulation mit 5415 Servern durchgeführt wird. Die Autoren kommen zu dem Schluss, dass *SHIP* auch in diesem Setup das gegebene Energie-Budget erfolgreich umsetzt.
Kritik: Über den verwendeten Simulator wird nur verraten, dass es sich um ein C++ Programm handelt, welches echte Daten für die CPU-Auslastung verwendet (welche aber nur Mittelwerte über 15 Minuten Perioden sind). Eine Validierung findet nicht statt.
Um den Vorteil des dezentralen Charakters ihres System argumentativ zu unterstützen, werden Laufzeiten ihres Controllers vorgestellt, bei denen die Menge der verwalteten Rechner variiert wird. Die Messungen zeigen, dass ihr Controller schlecht skaliert. Dies zeigt aber nicht, dass ein zentralistisches System diese Aufgabe nicht lösen kann. Als weiteren Beleg für den Vorteil eines dezentralen Systems wird das bessere Abschneiden ihres ESS in den Messungen angeführt. Die Ursache ist allerdings der zu einfache Algorithmus von *Per-Rack* in dem untersuchten Szenario und nicht *Per-Racks* zentralistischer Ansatz. *Per-Rack* hätte in ihrer 5415 Server umfassenden Simulation getestet werden müssen, um eine valide Aussage über dessen Skalierung zu erhalten.
Als Grund für die Verwendung von DVFS, anstelle eines An/Abschalt-Algorithmus, wird die mögliche Servicereduktion angegeben. Ihr Algorithmus würde allerdings bei einem zu geringen Budget ebenfalls die Performance des Service beeinträchtigen. Wie sehr der Service von ihrem System beeinflusst werden kann, wird nicht weiter betrachtet.

Titel: *Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems*
Schlüsselwörter: *Messung, sprint-to-halt, gemixte Arbeitslast, Cluster Computing*
Pinheiro, Bianchini, Carrera und Heath entwickelten in [104] eine der ersten Strategien, welche Maschinen, abhängig von der vorherrschenden Last-Situation, an- bzw. abschalten. In [105] ver-

feinern sie ihre Strategie auf unterschiedliche Art und Weise. Ihre verbesserte Strategie verwendet Regel-Technik (einen *Proportinal-Integral-Differenzial (PID) feedback Controller*) zur Entscheidungsfindung. Dieser Controller berücksichtigt den aktuellen Bedarf an Ressourcen (nicht deren Auslastung), den vergangenen Bedarf und die Änderung des Bedarfs. Die Anteile werden mit 0,7/0,15/0,15 gewichtet. Für die Strategie kann außerdem angegeben werden, wie stark der entsprechende Dienst beeinträchtigt werden darf. Sie implementieren ihre Strategie in einem selbst entwickelten Webserver (*PRESS* [106, 107]) und einem selbst entwickelten Betriebssystem für Cluster (*Nomad* [108, 109]). Ihr System erreicht, abhängig von gewähltem Szenario und Parametern, eine Einsparung von bis zu 45%.
Kritik: Diese Arbeit erreicht zwar beeindruckende Ersparnisse, sie analysiert aber nicht, wieviel eine optimale Strategie für ihre Szenarien eingespart hätte. Da dieser Vergleich fehlt, ist es schwierig zu sagen, wie gut ihre Strategie wirklich funktioniert. Eine hohe Ersparnis kann in einem Experiment leicht erreicht werden, indem man viele Maschinen, aber wenig Last wählt. Daher sollte immer ein Vergleich zum möglichen Optimum durchgeführt werden. Anhand der präsentierten Ergebnisse ist eine solche Abschätzung für den Leser schwierig.
Es fehlt ebenfalls eine Betrachtung, wie stark der jeweilige Dienst eingeschränkt wurde. Die Strategie berücksichtigt zwar, wie erwähnt, einen Parameter, der die erlaubte Beeinträchtigung angibt, es wird aber nicht überprüft, inwiefern dieser auch eingehalten wird und wie stark die wirkliche Beeinträchtigung war.
Auf eine Betrachtung der Skalierbarkeit ihrer Strategie wird ebenfalls verzichtet. Getestet wird mit maximal 8 Rechnern. Reale Setups können wesentlich größer sein und es wäre interessant zu sehen, ob die verwendete Strategie in größeren Umgebungen an ihre Grenzen stößt.

Ihre Arbeit hat die Gruppe um Bianchini[4] (Rutgers University, New Jersey) damit begonnen, Festplattenzugriffe zu optimieren, um Energie in Laptops einzusparen [110]. Aktuell befassen sie sich mit der Entwicklung eines vollständig aus erneuerbarer Energie angetriebenen kleinen Rechenzentrums namens *PARASOL* [111, 112].

Titel: *Power management by load forecasting in web server clusters*
Schlüsselwörter: *Messung, sprint-to-halt, Vorhersage, feste Arbeitslast, Webserver*
Santana, Leite und Mossé setzen in [113] einen zu dieser Arbeit sehr ähnlichen Ansatz um. Sie vergleichen in einem aus fünf heterogenen Back-End-Servern bestehenden Apache-Cluster drei verschiedene Verfahren mit ihrem eigenen. Für die Lasterzeugung wird `httperf` verwendet und für die Lastverteilung wird Apache mit dem Modul `mod_proxy_balancer` eingesetzt. Die Verfahren, die verglichen werden, sind: Alle Rechner arbeiten mit maximaler Frequenz, alle Rechner verwenden Linux-interne Methoden, um DVFS zu nutzen und eine Methode, die DVFS verwendet und zusätzlich Maschinen nach Bedarf an- und abschaltet. Ihr eigenes Verfahren erweitert das letzte Verfahren um eine Lastvorhersagetechnik. Beide Verfahren, die Rechner an- und abschalten, verwenden eine offline erzeugte Liste, in der zu jeder möglichen Auslastung (in 0,1 % Auflösung) die Konfiguration des Clusters vermerkt ist, welche in dieser Situation angenommen werden sollte. Die Konfigurationen umfassen, welche Rechner mit welcher Frequenz laufen sollen und welche Rechner abgeschaltet sein sollen. Das verwendete Verfahren baut auf Vorarbeiten der Gruppe auf, die in [114, 115, 116, 117] vorgestellt werden. Als Arbeitslast ihrer Experimente wird ein 12,5

[4] Bianchini arbeitet jetzt für Microsoft Research und optimiert deren Rechenzentren und online Services.

Stunden langer Teil des Traces der Fussballweltmeisterschaft 1998 [92] verwendet. In den Experimenten erreichen sie Einsparungen von bis zu 59 %, wobei der Unterschied zu den ersten beiden genannten Verfahren sehr hoch ist (bis zu 74,7 %), aber zu dem Verfahren, welches auf Lastvorhersage verzichtet, sehr gering ist (bis zu 1,6 %). Die Lastvorhersage macht sich allerdings in den gleichzeitig erreichten Werten für die *Qualität des Dienstes* bemerkbar. Hier schneidet ihr eigenes Verfahren mit Lastvorhersage gegenüber dem Verfahren ohne Lastvorhersage deutlich besser ab.
Kritik: Der für die Messungen verwendete Trace ist bereits sehr alt. Internetverkehr hat sich seitdem nicht nur um ein Vielfaches verstärkt, sondern ist durch das Web 2.0 auch vielfältiger geworden. Die Autoren betrachten allerdings nur einen Typ von Anfrage. Für eine Untersuchung moderner Webserver ist dies nicht ausreichend.
Um ihr System verwenden zu können, muss vorher durch intensives Benchmarking des vorhandenen Systems die erwähnte Konfigurationstabelle erzeugt werden. Ein so aufwendiges Verfahren schreckt potentielle Nutzer von einer Verwendung ab. Die präsentierten Daten lassen außerdem darauf schließen, dass die erzeugten Konfigurationen anfällig für Zustands-Flattern sind. Wenn die Last zwischen der Grenze von zwei verschiedenen Konfigurationen schwankt, kann ein häufiger Zustandswechsel der Rechner beobachtet werden. Die Autoren rechtfertigen das Verhalten damit, dass die Methode ohne Lastvorhersage noch mehr Zustandswechsel verursacht.
Obwohl hohe Einsparungen erzielt werden und die Qualität des Dienstes bewertet wurde, wird nicht darauf eingegangen, was die optimale Einsparung für das verwendete Szenario gewesen wäre. Ohne dieses Optimum kann, wie bei den anderen Arbeiten auch, nicht eingeschätzt werden, wie gut das vorliegende System wirklich funktioniert.

Titel: *Balancing Power Consumption in Multiprocessor Systems*
Schlüsselwörter: *Messung, Scheduling, gemixte Arbeitslast, Cluster Computing*
In einer Arbeit von Merkel und Bellosa [118] wird ein energiebewusster Scheduling-Algorithmus für Multikern-Systeme vorgestellt. Dieser klassifiziert Tasks, abhängig von *event monitoring countern* im Betriebssystem, in *heiße* Tasks (verbrauchen viel Energie bei der Ausführung) und *kalte* Tasks (verbrauchen wenig Energie bei der Ausführung). Die gelernte Eigenschaft kann sich über die Zeit noch verändern, wodurch initiale Fehleinschätzungen korrigiert werde können. Anhand der Klassifikation wird versucht, Tasks so zu verteilen oder zu migrieren, dass eine stabile Temperatur auf allen Kernen herrscht. So werden kalte und heiße Tasks auf der selben CPU kombiniert, um sich auszugleichen, oder heiße Tasks von erhitzten Kernen auf kühlere Kerne migriert. So soll *Thermal Throttling*, also das Heruntertaktet bei Überhitzung, verhindert werden. Der Algorithmus wurde in einen Linux 2.6.10 Kernel integriert und praktischen Tests in einem 8-Kern-System unterzogen. Die präsentierten Messungen zeigen für einen Mix aus sechs Programmen (z.B. `openssl`, `bzip2`) einen Performancegewinn von 4,9 %. Es wird zusätzlich untersucht, ob eine heterogene Arbeitslast von Vorteil für diese Strategie ist. Dafür werden 18 Tasks, bestehend aus drei verschiedenen Programmen (heiße, mittlere und kalte Tasks) in verschiedenen Verhältnissen eingespielt. Die Arbeitslast mit 8 heißen, 2 mittleren und 8 kalten Tasks erreicht dabei den höchsten Performancegewinn mit 12,3 %, wohingegen eine Arbeitslast mit 18 mittleren Tasks nur eine Steigerung von etwa 1,9 % erreicht. Ein weiterer Vorteil ergibt sich, wenn weniger heiße Tasks laufen, als es Prozessoren gibt. In diesem Fall können die Tasks auf die nicht laufenden kalten Kerne migriert werden, um dort weiterhin mit voller Leistung zu rechnen. In ihrem Experiment erreichen sie so einen Performancegewinn von 76 % bei einem Task auf ihrem 8-Kern-System.
Kritik: Obwohl die Arbeit einen sehr interessanten Ansatz verfolgt, wird nicht weiter quantifi-

ziert, wie viel Energie eingespart wurde. Solch eine Technik könnte andernfalls, wie auch DVFS, zusätzlich zum An- und Abschalten von Maschinen genutzt werden.

Titel: *Phase-Based Application-Driven Hierarchical Power Management on the Single-chip Cloud Computer*
Schlüsselwörter: *Messung, slow-and-steady, gemixte Arbeitslast, Cluster Computing*
In der Arbeit von Ioannou et al. [119] wird ein Werkzeug vorgestellt, welches mit Hilfe von DVFS, Energie in MPI-Programmen sparen soll. Das Werkzeug ist für den experimentellen Single-Chip Cloud Computer (*SCC*) von Intel konzipiert, welcher 48 Kerne besitzt [120]. Das Werkzeug versucht anhand von wiederkehrenden MPI-Calls, Programmteile, die sich wiederholen, zu entdecken. Initial wird durch sukzessives Absenken der Frequenz und Spannung die ideale Einstellung für jede erkannte Phase des Programms ermittelt. Dabei wird so lange die Leistung gedrosselt, bis ein vorher gewählter Grenzwert unterschritten wird (z.B. Performanceverlust von 10 %). Ob Frequenz und Spannung angepasst werden, wird allerdings nicht pro Kern entschieden [5], sondern von einem Domain-Manager. Jeder Kern besitzt einen lokalen Manager, der seinen Wunsch bezüglich Frequenz und Spannung an den Domain-Manager sendet. Abhängigkeit von einer gewählten Strategie (z.B. höchster Wunsch oder Durchschnitt der Wünsche einer Domain) entscheidet der Domain-Manager dann über Frequenz und Spannung der gesamten Domain. Zur Evaluierung wurde eine Reihe von Experimenten durchgeführt, welche Tests aus dem NAS Parallel Benchmark [121] und dem SPEC MPI 2007 Benchmark [122] verwenden. Die verwendete Haupt-Metrik ist ein normalisiertes *Energy Delay Product (EDP)*, welches die verbrauchte Energie und die erreichte Performance ins Verhältnis setzt. Ihre Ergebnisse liegen im Durchschnitt, bei einer Verbesserung von 11,4 % (*EDP*) mit einem durchschnittlichen Performanceverlust von 7,7 %. Im besten Falle erreichen sie eine Verbesserung von 27,2 % (*EDP*). Ihr selbst entwickelter Phasen-Prädiktor wird ebenfalls mit einer Alternative verglichen, welche sich als weniger genau herausstellt.
Kritik: Auch dieser Ansatz könnte zusätzlich zu einer An- und Abschalt-Strategie verwendet werden. Da das Verfahren auf MPI-Programme beschränkt ist, ist ihr Anwendungsfeld sehr begrenzt und nützlicher für den HPC-Bereich.

Titel: *Designing Power-Aware Collective Communication Algorithms for InfiniBand Clusters*
Schlüsselwörter: *Messung, sprint-to-halt, gemixte Arbeitslast, Cluster Computing*
Kandalla et al. untersuchen in [123] ebenfalls MPI-Programme hinsichtlich ihres Energiesparpotenzials. Im Fokus steht die Verwendung von kollektiver Kommunikation in InfiniBand Clustern mit MVAPICH2 [124] auf einer Intel *Nehalem* Architektur. Sie unterteilen kollektive Kommunikation in drei Phasen. Die erste Phase kümmert sich um intranode Kommunikation. Alle Kerne senden ihre Nachrichten an einen einzelnen Kern, den *Communication Leader* (CL). Im zweiten Schritt tauschen sich alle CL über die einzelnen Rechner hinweg aus (internode). Der letzte Schritt umfasst die Verteilung der Nachrichten vom jeweiligen CL einer Maschine zurück an die restlichen Kerne der jeweiligen Maschine. Bei diesem Modell der Kommunikation kann während der zweiten Phase Energie gespart werden, indem die Kerne, die nicht CL sind, auf ein Minimum getaktet werden. Ein zweiter Vorschlag betrifft Kommunikation, bei der alle Kerne involviert sind (sogenanntes *All-to-All*). Ihr Vorschlag teilt diese Art der Kommunikation in vier Schritte ein. Der

[5] Es können auf dem *SCC* nur vier sogenannte *Tiles* gleichzeitig ihre Spannung ändern und die Frequenz kann nur pro *Tile* verändert werden. Auf jedem *Tile* befinden sich 2 CPUs.

erste Schritt behandelt wieder intranode Kommunikation. Beim zweiten Schritt kommunizieren erst 50 % der Kerne jedes Knotens mit je 50 % der Kerne anderer Knoten. In dieser Zeit kann die verbleibende Hälfte auf ein Minimum getaktet werden. Schritt drei entspricht Schritt zwei, mit dem Unterschied, dass die in Schritt zwei inaktiven Kerne nun die Rollen mit den aktiven Kernen tauschen. Der letzte Schritt umfasst Paare, die bis jetzt noch nicht kommuniziert haben. Auch hier können die nicht kommunizierenden Kerne heruntergetaktet werden. Die Kommunikation wird aufgeteilt, damit das verwendete Netz entlastet wird. Neben einer Modell-Beschreibung werden auch Experimente vorgestellt, welche als Arbeitslast u.a. den NAS Parallel Benchmark [121] verwenden. In den Experimenten ergibt sich ein Performanceverlust von bis zu 5 % bei einer Energieersparnis von bis zu 8 %.
Kritik: Diese Arbeit hat dieselben Probleme, wie auch die von Ioannou et al. [119]. Sie ist in ihrem Anwendungsbereich sogar weiter eingeschränkt, da sie nur in InfiniBand Clustern arbeiten kann und nur dann Vorteile bringt, wenn die verwendete Applikation sehr kommunikationsintensiv ist. Andernfalls gibt es zu wenig Situationen, in denen ihre Strategie Energie einsparen kann. Die Ergebnisse der Arbeit sind mit Einsparungen von 8 % bei gleichzeitigem Performanceverlust von 5 % eher ernüchternd.

Titel: *Protection System against Overload and Distributed Denial of Service Attacks*
Schlüsselwörter: *Messung, Überlast, Cluster Computing*
Tóth et al. schlagen in [125] eine Strategie für Überlast-Situationen bzw. Denial-of-Service (DoS) Angriffe vor. In ihrem System wird vorgeschlagen, dass Anfragen nicht beim Eintreffen an der Queue verworfen werden (es wird eine unendlich lange Queue angenommen), sondern beim Verlassen der Queue und unmittelbar vor der potenziellen Verarbeitung. Die Anfragen werden dafür beim Eintreffen mit Metadaten versehen. Anhand dieser Metadaten wird ermittelt, wie lange die Anfrage schon gewartet hat, bevor sie zur Verarbeitung weiter gereicht werden würde. Überschreitet die gewartete Zeit ein gewisses Maß, wird davon ausgegangen, dass der Client das Warten auf die Antwort aufgegeben hat und die Anfrage wird, ohne sie zu bearbeiten, verworfen. Vor die Queue wird zusätzlich ein Filter (Black/White Liste) geschaltet, der den Verkehr weiter reduzieren soll. Das Konzept sieht zwar eine unendlich lange Queue und unendlich viele Arbeiter vor, es gibt jedoch jeweils ein Monitoring Modul, welches einschreitet, falls die Queue zu lang wird bzw. es zu viele Arbeiter gibt. In beiden Fällen wird ein Angriff angenommen. Kommt es dazu, wird nach bösartigen Anfragen in der Queue gesucht und diese in die Blacklist aufgenommen. Werden zu viele Arbeiter erzeugt, werden Arbeiter, die überdurchschnittlich viele Ressourcen verbrauchen, entfernt. Durch dieses System wollen die Autoren Blocking-Attacks [6] verhindern.
Kritik: Die Idee, Request zu verwerfen, bevor sie bearbeitet werden, ist sehr interessant. Die Autoren geben allerdings zu, dass ein versierter Angreifer auch in ihrem System Probleme bereiten kann. Insbesondere gegen DDoS-Angriffe [7] mit gespooften [8] IP-Adressen kann dieses System nicht schützen. Die Annahmen einer unendlich langen Queue und einer unendlichen Anzahl an Arbeitern ist ebenfalls nicht realistisch. Da die Länge und Anzahl aber durch das Monitoring Mo-

[6] Eine Form von DoS-Angriff, bei dem beschränkte Ressourcen (z.B. Anzahl von Arbeitern) möglichst lange vom Angreifer blockiert werden.
[7] Distributed Denial of Service. Bei diesen Angriffen werden ganze Farmen von Rechnern, sogenannte Botnetze, für den Angriff verwendet.
[8] Die IP-Adresse des Absenders wird in diesem Fall gefälscht, so dass eine Rückverfolgung des Angreifers praktisch unmöglich wird.

dul überwacht werden, wird nicht ganz klar, inwiefern diese Anforderung überhaupt notwendig ist oder zu praktischen Problemen führen kann.

Titel: *RTP: Robust Tenant Placement for Elastic In-Memory Database Clusters*
Schlüsselwörter: *Messung, sprint-to-halt, Scheduling, feste Arbeitslast, Cloud Computing*
Eine Arbeit, die sich mit der Platzierung von In-Memory Datenbanken beschäftigt, wurde von Schaffner et al. [126] publiziert. Sie versuchen mit verschiedenen Scheduling-Verfahren eine besonders günstige Verteilung von Datenbanken zu erreichen. Die Verteilung soll erreichen, dass Server eingespart werden können. Jeder Dienst hat dabei mindestens ein Replikat, mit dem es sich die Last teilen kann. Das Replikat dient im Notfall als Backup und muss daher immer genügend freie Kapazität besitzen, um unerwartete Hardwareausfälle abfangen zu können. Die Platzierungsalgorithmen benötigen dabei Laufzeiten in der Größenordnung von 5 Minuten, bei einem Intervall von 10 Minuten. Eine interessante Erkenntnis ist, dass sich die Kosten in ihrem Fall insgesamt senken lassen, wenn mehr Replikaten vorgehalten werden, als notwendig wäre. Es wird vermutet, dass die Ursache dafür die geringere Übersteuerung beim Anschalten von neuen Servern ist, wenn insgesamt mehr Replikate verwendet werden. Die seltenen Update-Zyklen wirken sich hier negativ auf ein Übersteuern aus.
Kritik: Die vorgeschlagene Lösung ist für Webserver Systeme aufgrund der hohen Latenz nicht geeignet.

Titel: *ElasticTree: saving energy in data center networks*
Schlüsselwörter: *Messung & Simulation, sprint-to-halt, Routing, gemixte Arbeitslast, Cluster & Cloud Computing*
Heller et al. beschreiben in [127] einen Netzwerk-Power-Manager, der im Stande ist, das gesamte Netzwerk zu überwachen und ungenutzte Netzwerkkomponenten (Router, Switches) abzuschalten. Ihr System besteht aus drei Elementen. Der *Optimizer* berechnet anhand der Topologie, dem aktuellen Netzwerkverkehr, einem Energie-Modell der Hardware und einer gewünschten Fehlertoleranz das minimale Netzwerk, welches für die Anforderung benötigt wird. Die Information, welche Komponenten aktiv bleiben und welche abgeschaltet werden sollen, wird vom *Optimizer* an den *Power-Controller* und das *Routing-Modul* weitergegeben. Der *Power-Controller* kümmert sich um das An- und Abschalten der Komponenten im Netzwerk und das *Routing-Modul* berechnet anhand der neu entstandenen Topologie alle Routen und aktiviert diese in den Netzwerkkomponenten [9]. Die Arbeit wird basierend auf Mahadevan et al. [129] motiviert. Mahadevan et al. haben festgestellt, dass das Netzwerk in Rechenzentren 10 - 20 % des gesamten Energieverbrauchs ausmacht. Es wurde zusätzlich ermittelt, dass aktuelle Switche bei 0 % Auslastung immernoch 70 - 80 % der Energie benötigen, die sie bei 100 % Auslastung verbrauchen. Durch dieses schlechte Energieverbrauchsprofil wird das Abschalten einer solchen Ressource besonders attraktiv. Heller et al. untersuchen im weiteren Verlauf ihrer Arbeit vor allem die *Fat-Tree*-Topologie, in der es besonders viele redundante Netzwerkgeräte gibt. Mit Messungen in einem echten Testbett überprüfen sie ihre Methode. Zum Einsatz kommen verschiedene Lastprofile, die darüber entscheiden, welche Maschinen innerhalb der Topologie miteinander kommunizieren. Darunter befinden sich unter anderem zufällige Paare, Paare, die nah beieinander liegen (z.B. am selben Switch ange-

[9] Hierfür wird unter anderem OpenFlow [128] verwendet. OpenFlow ist ein Werkzeug, um *Software Defined Networks* zu verwalten.

schlossen sind), Paare, die weit entfernt sind und damit den gesamten Baum durchlaufen müssen und ein Trace aus einem realen Rechenzentrum. Die erreichten Einsparungen hängen sehr stark von dem Kommunikationsmuster und der Netzauslastung ab. Bei Paaren, die nah beieinander liegen, kann bis zu 60 % eingespart werden. Generell kann bei mehr Auslastung und weiter entfernten Paaren weniger gespart werden. Für den realen Trace (welcher für die Größe des Testbetts angepasst wurde) verzeichnen sie Einsparungen von 25 % - 62 %. Da sie für ihre Methoden volles Wissen über zukünftigen Verkehr voraussetzen und dieser in der Realität nicht vorhanden ist, experimentieren sie ebenfalls mit Lastvorhersage, um das fehlende Wissen auszugleichen. Verwendet wird ein einfaches *AR(1)* Modell. Ergebnisse werden aber nicht präsentiert.
Kritik: Wie die Autoren feststellen, ist das verwendete Wissen in der Realität nicht vorhanden. Es wäre sehr interessant gewesen, die erzielten Ergebnisse unter Verwendung des AR(1) Modells zu sehen.

Titel: *Analysis of the Power and Hardware Resource Consumption of Servers under Different Load Balancing Policies*
Schlüsselwörter: *Messung, Scheduling, feste Arbeitslast, Webserver*
In einer Arbeit von Dargie und Schill [96] werden verschiedene Scheduling-Verfahren in einem Multimedia Cluster auf ihre Effizienz hin untersucht. Das aus zwei Back-End-Servern bestehende Cluster stellt Videos in der Größe zwischen 3 - 100 MB bereit und liefert diese über einen Apache-Server aus. Das Netzwerk ihres Setups wird durch Spezialhardware [130] emuliert. Die beiden Back-Ends werden durch spezielle Hardware vermessen. Mit dieser kann die Stromaufnahme der verschiedenen Adern (3,3V, 5V, 12V1 etc.) genau festgestellt werden und dadurch auf den Stromverbrauch der einzelnen Komponenten geschlossen werden. Untersucht werden die zustandslosen Scheduling-Verfahren Round-Robin und Random sowie die drei zustandsbehafteten Verfahren Least-Session, Least-CPU und Least-Bandwidth. Als Arbeitslast für ihr Experiment wird eine Poisson-Verteilung angenommen, welche laut Autoren dem Verhalten bei IPTV-Anwendern ähnelt und eine Dauer von 12 Stunden umfasst. Ihre Messungen ergeben keine signifikanten Unterschiede zwischen den Verfahren. Da die CPU nie mehr als 50 % ausgelastet ist, wiederholen sie das Experiment und verwenden diesmal ein 10 Gbit/s Netzwerk anstelle eines 1 Gbit/s Netzwerks. Durch die Verzehnfachung des Netzwerks wird dieses als Flaschenhals entfernt. Der Durchsatz der Server verdoppelt sich, bei gleichzeitiger Erhöhung des Energieverbrauchs von nur 1,25 %. Ihr letztes Experiment untersucht den zusätzlichen Effekt, den eine Verwendung von DVFS mit sich bringt. Da die Arbeitslast nicht CPU-gebunden ist, kann die Energieeffizienz um 12 % gesteigert werden, ohne dass der Dienst beeinträchtigt wird. Diese Steigerung wird von den Autoren nicht als signifikant bewertet. Sie schlagen stattdessen vor, komplementäre Dienste zu kombinieren (in diesem Fall also zusätzlich CPU-gebundene Anfragen), um alle Ressourcen möglichst auszulasten.
Kritik: Die Autoren verwenden für die Überprüfung der Scheduling-Verfahren nur zwei Back-End-Rechner und diese sind die meiste Zeit ihres Tests nur mäßig belastet. Scheduling-Verfahren zeigen ihre Stärken häufig erst in extremeren Situationen, wie längere Überlastphasen oder große Cluster. Für eine höhere Aussagekraft hätten die Experimente daher in einem größeren Umfang getestet werden müssen.

Titel: *Analysis of the Power Consumption of a Multimedia Server under Different DVFS Policies*
Schlüsselwörter: *Messung, Scheduling, feste Arbeitslast, Webserver*
Dargie überprüft in [131], einer scheinbaren Vorarbeit zu [96], sehr ähnliche Aspekte. Beide Arbeiten stammen aus demselben, vom DFG geförderten Projekt. Hier wird allerdings nur eine einstündige Arbeitslast verwendet, dafür werden hier bereits zwei komplementäre Dienste kombiniert und unter den selben DVFS-Bedingungen wie in [96] getestet. In [131] wird ihre als Arbeitslast verwendete Verteilung motiviert und vorgestellt, welche auch in [96] genutzt wird. Ansonsten beschränken sich die hier erzielten Ergebnisse im Grunde genommen auf *„Each of the policies have their merits and demerits."* [10].

Titel: *Dynamic Voltage and Frequency Scaling in Multimedia Servers*
Schlüsselwörter: *Messung, Scheduling, gemixte Arbeitslast, Webserver*
Brihi und Dargie überprüfen in [132] im Wesentlichen die Resultate aus [131]. In [132] wird, im Gegensatz zu [131], auf einen heterogenen, aus zwei Back-End-Servern bestehenden Apache-Webserver zurückgegriffen. Die Ergebnisse sind ähnlich aufschlussreich wie die aus [131].

Titel: *Investigation into the energy cost of live migration of virtual machines*
Schlüsselwörter: *Messungen, gemixte Arbeitslast, Cloud Computing*
In [133] untersuchen Rybina, Dargie, Strunk und Schill den Einfluss von Live-Migration von VMs auf den Energieverbrauch der betreffenden Maschinen. Virtualisierung stellt eine wichtige Strategie in der Reduktion von Energiekosten dar und durch die Konsolidierung von VMs in Cloud-Clustern können zusätzlich Maschinen abgeschaltet werden. In der Arbeit werden vier verschiedene Arbeitslasten (in Form von vier verschiedenen Verteilungsfunktionen) untersucht, welche allerdings alle CPU-gebundene Last darstellen. Weiterhin variiert wird die Größe der VM (800 MB, 900 MB, 1200 MB) und die Bandbreite des Netzwerks (50 MB/s, 90 MB/s, 100 MB/s). Die Autoren stellen fest, dass während der Migration nur der Verbrauch des Quell-Rechners einen schwankenden Energieverbrauch besitzt, da dieser die Arbeit während der Migration weiter ausführt. Außerdem wird festgestellt, dass die Bandbreite einen entscheidenden Unterschied in ihrem Experiment ausmacht. Je kleiner die VM oder je größer die Bandbreite, desto schneller kann eine Migration erfolgen und umso weniger Energie wird benötigt.
Alle vier Arbeiten ([131, 96, 132, 133]) stammen aus demselben DFG Projekt (SFB 912/1 2011), innerhalb dessen noch weitere, hier nicht vorgestellte, Arbeiten publiziert wurden.

Titel: *The saving of energy in Web server clusters by utilizing dynamic sever management*
Schlüsselwörter: *Messung, Vorhersage, feste Arbeitslast, ESS, Webserver*
Lien et al. stellen in [134] ein ESS vor, welches auf Basis des *Delay-Power Products* errechnet, wie die optimale Konfiguration des Clusters zu einem gegebenen Zeitpunkt sein sollte. Sie stellen ein Modell für den Verbrauch eines Clusters und die Verzögerungszeit von Anfragen in einem Cluster vor. Das Produkt beider Modelle ergibt das erwähnte *Delay-Power Product*, welches die Anzahl der aktiven Rechner und die Wartezeiten von Anfragen möglichst gering halten soll. Da die Rate der eintreffenden Anfragen eine große Rolle für ihr System spielt, verwenden sie historische Daten, um eine Vorhersage für die aktuelle Last zu treffen. Mit Daten aus zehn Läufen erreichen sie eine Vorhersagegenauigkeit von 78,2 %. Sie prüfen ihr System mit einem Experiment, welches ein

[10] *„Jede Strategie hat ihre Vor- und Nachteile."*

19 Maschinen Cluster nutzt (zwei Sets aus jeweils homogenen Maschinen). Als Arbeitslast wird ein einwöchiger Trace verwendet. Dieser weist starke Tag-Nacht-Zyklen auf. Mit ihrem System erreichen sie Einsparungen von 16,9 % Energie und halten dabei die Verzögerung der Anfragen konstant.
Kritik: Die Arbeit fällt sehr kurz aus, weshalb wichtige Fragen offenbleiben. Unter anderem ist nicht klar, wie viel Energie mit einem optimalen System hätte gespart werden können und wie gut ihr System somit wirklich arbeitet. Ihr Modell wird nicht auf Korrektheit überprüft. Es wird nicht klar, welche Daten zum Lernen für die Vorhersage in ihrem Experiment verwendet wurden. Die Auflösung ihres Experiments ist für einen Webserver zu grob (Stunden bzw. Tage). Es kann dadurch nicht erkannt werden, ob und wie ihr System auf kleine lokale Schwankungen (Sekunden bzw. Minuten) reagiert.

Titel: *Towards Greener Data Centers with Storage Class Memory: Minimizing Idle Power Waste Through Coarse-grain Management in Fine-grain Scale*
Schlüsselwörter: *Messungen, sprint-to-halt, STR, Webserver*
In einer Arbeit von Doh et al. [135] wird vorgeschlagen, sogenannten *Storage Class Memory (SCM)* zu verwenden, um die Dauer der Bootzeit von Rechnern auf ein Minimum zu reduzieren. Bei SCM handelt es sich um eine Art von Speicher, welcher, im Gegensatz zu DRAM, seine Informationen auch ohne Stromzufuhr hält (sogenannter nichtflüchtiger Speicher). Die Arbeit verwendet dieselbe Technik, die auch bei STR angewendet wird, und speichert alle Systeminformationen im Speicher, um ein erneutes Starten des Rechners zu beschleunigen. Da SCM aber keine Stromversorgung benötigt, kann in ihrem Ansatz der Rechner komplett vom Strom getrennt werden. Die Autoren führen Experimente in einem kleinen vier-Knoten-Cluster durch. Verwendet wird ein Apache-Webserver und 32 MB FeRAM (eine Art von SCM). In ihrem Setup benötigen die Rechner 1,22 Sekunden, um nach einem STR wieder einsatzbereit zu sein. Nachdem ein einzelner Rechner auf sein Verhalten hin untersucht wurde, unternehmen die Autoren ein Experiment mit dem gesamten Cluster. Als Arbeitslast dient ein Trace von IBM, welcher typische Tag-Nacht-Zyklen eines Rechenzentrums beinhaltet. Dieser einwöchige Trace wird auf 20 Minuten gestaucht und die Last wird so reduziert, dass sie noch von ihrem Setup bewältigt werden kann. Obwohl keine konkreten Zahlen zur Einsparung und zu Antwortzeiten genannt werden, sehen die präsentierten Diagramme überzeugend aus. Die Autoren sind sich bewusst, dass zwischen ihrem Setup und einer echten Nutzung noch große Unterschiede bestehen (u.a. sind 32 MB RAM viel zu wenig für aktuelle Setups) und diskutieren mögliche Schritte und Alternativen für ihren Ansatz. Beispielsweise argumentieren sie, dass es immer mehr kleine Systeme gibt (Sensoren etc.) und der vorgestellte Ansatz besonders in diesen Bereichen berücksichtigt werden sollte.
Kritik: Die Autoren verwenden einen sehr einfachen Scheduler, der die Rechner nacheinander mit Last füllt (Rechner 1 bekommt Last bis eine Kapazität erreicht ist, dann Rechner 2 usw.). Einer der aktiven Rechner schwankt somit immer in seiner Last, während die restlichen aktiven Rechner voll ausgelastet werden. Anhand des schwankenden Rechners kann ihr System erkennen, ob eine neue Maschine angeschaltet werden muss oder ob die aktuelle Maschine abgeschaltet werden kann. Dieser Ansatz ist in größeren Setups mit Last, die stärker schwankt, nicht geeignet. Sobald die Last stärker schwankt, als die Kapazität einer Maschinen ist, werden Maschinen nie rechtzeitig an sein (auch wenn sie nur 1,22 Sekunden zum Hochfahren benötigen) und Maschinen werden sehr viel häufiger an- und abgeschaltet, was ihre Lebensdauer einschränken kann.
Obwohl sich die Autoren, wie bereits erwähnt, bewusst über die praktischen Schwächen ihres

Prototypen sind, sieht die von ihnen aufgestellte Prognose bezüglich der Entwicklung von SCM nicht gut aus. Die Arbeit, die im Jahr 2010 entstanden ist, zitiert eine Schätzung, dass im Jahr 2012 die Kosten pro Megabyte für *Phase-Change Memory* (ebenfalls eine Form von SCM) gleichauf mit denen von normalen Festplatten sein werden. Im Jahre 2016 ist SCM allerdings noch nicht auf dem Stand, der für 2012 prognostiziert wurde.
Das Kernproblem ist, wie die Autoren richtig feststellen, die Dauer des Bootvorgangs und dieser ist in normalen Systemen, auch mit STR, höher als in ihren, auf eingebetteten Systemen basierenden, Prototypen.

Titel: *NapSAC: Design and Implementation of a Power-proportional Web Cluster*
Schlüsselwörter: *Messung & Simulation, sprint-to-halt, Lastvorhersage, Webserver*
Krioukov et al. schlagen in [136] vor, stark heterogene Cluster zu verwenden, um die Stärken der einzelnen Klassen von Maschinen zu kombinieren. Die Autoren verwenden deshalb in ihrer Arbeit leistungsstarke 4-Kern Intel Nehalems zusammen mit Intel Atom (etwa 10 Mal schwächer als ein Nehalem) ausgestatteten Rechnern und *BeagleBoards* [137] (etwa 7 Mal schwächer als ein Atom). In ihrem Setup gilt, je schwächer das System, je schneller ist ihr Bootvorgang, aber desto weniger energieeffizient ist das System. Ihre hierarchische Architektur sieht vor, die schwachen Maschinen zu verwenden, um kurze Schwankungen in der Last abzufangen und die Zeit zu überbrücken, bis die leistungsstarken Rechner betriebsbereit sind. Um Startentscheidungen zu treffen, wird auch in dieser Arbeit auf verschiedene Weise Lastvorhersage betrieben. Am effektivsten schneidet ein Verfahren ab, welches ein gleitendes Fenster über die letzten n Sekunden der eingetroffenen Last legt und diese Werte mittelt. Für die Evaluation ihrer Architektur simulieren sie ein Wikipedia-Cluster und erzielen bis zu 90 % der optimal möglichen Einsparung. Sie vergleichen ebenfalls, wie ein homogenes Cluster aus Nehalem- oder Atom-Rechnern gegenüber ihrem heterogenen Ansatz abschneiden würde. Das aus Atom-Rechnern bestehende Cluster schneidet dabei ähnlich dem heterogenen Ansatz ab. Die Autoren argumentieren, dass die Atom-Rechner ein zu schwaches Leistung pro Volumen Verhältnis besitzen und deshalb ungeeignet für einen größeren Einsatz in Rechenzentren sind.
Kritik: Die Autoren geben keine Informationen über den genutzten Simulator. Außerdem wird nur eine synthetische Arbeitslast verwendet, was in Kombination mit einem Wikipedia Back-End nicht notwendig ist. Wikipedia stellt ebenfalls Traces ihrer Rechenzentren zur Verfügung, die eine realistischere Analyse ermöglicht hätten.

Titel: *Power Provisioning for a Warehouse-sized Computer*
Schlüsselwörter: *Messung & Simulation, technische Übersicht, Cluster Computing*
Fan, Weber und Barroso [11] analysieren und beschreiben in [138] den Energieverbrauch und die hierarchische Distribution der Energie in Rechenzentren, die bis zu 15.000 Server umfassen. Sie stellen dabei unter anderem fest, dass durch die konservativen Herstellerangaben zum Energieverbrauch von Hardware ein viel zu hoher Gesamtverbrauch veranschlagt wird. Wo es auf Rack-Ebene (unterste Ebene der Hierarchie) durchaus zu einer fast vollständigen Auslastung kommt, wird diese auf Clusterebene nie erreicht. Es könnten somit zusätzliche Racks angeschafft werden, um das Cluster besser auszulasten. Sie untersuchen ebenfalls, wie stark die Nutzung von DVFS

[11] Ebenfalls aus der Google-Gruppe, die in [9] über *Energy-Proportional Computing* geschrieben hat, siehe Abschnitt 1.1, Seite 2

die Effizienz weiter steigern könnte und erreichen Energieeinsparungen von bis zu 23 %. Für diese Überprüfung wurde vorher ein entsprechendes Energie-Modell aus bestehenden Daten abgeleitet und dessen Abweichung beträgt in ihrer Überprüfung nur 1 %.

Titel: *EnerJ: Approximate Data Types for Safe and General Low-Power Computation*
Schlüsselwörter: *Messung, approximatives Rechnen, gemixte Arbeitslast*
In Sampson et al. [139] schlagen die Autoren *approximate data types*, also *näherungsweise Datentypen*, zur Nutzung vor. Diese werden als Java-Erweiterung zur Verfügung gestellt. Die Idee ist, den Verbrauch von Speicher und die Komplexität von Operationen zu verringern, indem Datentypen verwendet werden, bei denen auf eine 100 prozentige Exaktheit verzichtet wird. Ein Beispiel für eine Anwendung solcher Datentypen sind Computerspiele. Hier ist es irrelevant, wenn bei der Berechnung der Grafik ein einzelnes Pixel falsch berechnet wird, da bei typischerweise 60-120 Bildern pro Sekunde einzelne Pixelfehler nicht auffallen. Monte-Carlo-Algorithmen, bei denen Zufall eine wichtige Rolle spielt, sind ebenfalls als Einsatzgebiet denkbar. Es wird auch vorgeschlagen, *näherungsweise Datenspeicherung* zu nutzen. Dabei kann z.B. die DRAM-Aktualisierungsrate oder die SRAM-Versorgungsspannung reduziert werden, da Fehler trotzdem noch unwahrscheinlich (abhängig vom Grad der Reduktion) oder unkritisch wären.

3.3 Fazit

Zusammenfassend kann gesagt werden, dass es bereits eine große Menge an Arbeiten im Bereich Energieeffizienz im Generellen und auch im Speziellen für den SLB-Bereich gibt.
Ein häufiges Problem der bestehenden Arbeiten sind Modelle, die unrealistische Annahmen treffen oder deren Umfang nicht groß genug ist. Sehr häufig wird nur die CPU modelliert, obwohl diese, wie bereits erwähnt, nicht allein den Energiebedarf dominiert. In dem später vorgestellten Simulator *ClusterSim* wird daher auf eine Datenbasis zurückgegriffen, die den Gesamtverbrauch eines Rechners berücksichtigt.
Des Weiteren fehlt in allen vorgestellten praktischen Arbeiten (bis auf eine) ein Vergleich mit einem möglichen erreichbaren Optimum. Eine aussagekräftige Bewertung erreichter Ergebnisse kann nur stattfinden, wenn das mögliche Optimum für den konkreten getesteten Fall bekannt ist. Es wird häufig eine erreichte Energieeffizienz oder erreichte Einsparungen angegeben, aber ohne die Angabe des möglichen Optimums stehen diese Werte in keinem Verhältnis und haben daher nur eine sehr bedingte Aussagekraft. Wo immer möglich, vergleicht diese Arbeit ihre Ergebnisse mit dem besten, theoretisch möglichen Ergebnis, um so aussagekräftig wie möglich zu sein.
Aus den vorgestellten Arbeiten geht hervor, dass eine reine Nutzung von DVFS nicht ausreicht, um in heutigen Clustern möglichst hohe Einsparungen zu erzielen. Es sollte in jedem Fall eine An-/Abschalt-Strategie verwendet werden, so wie es von *CHERUB* vorgesehen ist.
Die meisten Arbeiten, die An-/Abschalt-Strategien umsetzen, beschränken sich in ihrer Entscheidung über das An-/Abschalten auf einfache schwellwertbasierte Verfahren. Dieser Ansatz hat mehrere Schwachstellen. Es kommt zum Beispiel zu Problemen bei einem starken Lastanstieg und/oder bei der Verwendung von Hardware, die längere Bootzeiten hat. Es gibt zwar bereits Arbeiten, die sich im Zusammenhang mit SLB auch mit Lastvorhersage beschäftigen, doch diese besitzen noch zum Teil umfangreiche Offline-Anteile, die sie für die Praxisanwendungen umständlich machen, da sie einen erheblichen Mehraufwand für die Einrichtung des Systems bedeuten. Die vorlie-

gende Arbeit zeigt die Vorteile von Lastvorhersagen gegenüber klassischen, schwellwertbasierten Ansätzen und kommt ohne einen Offline-Anteil aus.
Viele der vorgestellten Systeme übernehmen außerdem RMS-spezifische Aufgaben, wie beispielsweise das Scheduling von Anfragen. Der Einsatz eines solchen Systems in einem bestehenden Cluster bedeutet, das bestehende RMS komplett zu ersetzen. Dies ist, auch abhängig von der Größe des Clusters, in den seltensten Fällen möglich. Der Vorteil von *CHERUB* ist, dass es vollständig unabhängig als separates Verwaltungswerkzeug fungiert und keine Eingriffe in bestehende Systeme vorgenommen werden müssen. Dadurch kann es praktisch in einem beliebigen RMS eingesetzt werden.

4 Anforderungen, Herausforderungen und Konzeption

Nachdem in den vorherigen Kapiteln erklärt wurde, warum die Thematik der energieeffizienten Clusterverwaltung wichtig ist und was bereits dafür getan wurde, ergeben sich für ein Energiesparsystem (ESS) im SLB-Bereich gewisse Anforderungen, die erfüllt werden müssen, und resultierende Herausforderungen, denen begegnet werden muss. Dafür wird ein Ansatz, der in den folgenden Kapiteln genauer untersucht wird, vorgestellt.

4.1 Anforderungen

Folgende Anforderungen müssen von einem ESS im SLB-Bereich bestmöglich erfüllt werden:

1) Energieeinsparung Ein ESS wird verwendet, um eine Gewinnmaximierung zu realisieren. Diese wird durch das Einsparen von sonst verschwendeter Energie erzielt (Abschalten von überschüssigen, inaktiven Maschinen).

2) Qualität des Dienstes Das Aufrechterhalten einer maximalen Qualität des Dienstes (engl. Quality of Service, *QoS*) muss gewährleistet werden. Nur wenn die *QoS* nicht signifikant leidet, kann ein ESS auch in einer Produktivumgebung eingesetzt werden.

Anforderung 1 und 2 stehen damit in direktem Gegensatz, da Anforderung 1 impliziert, möglichst wenige Ressourcen aktiv zu betreiben und Anforderung 2 möglichst viele.

3) Geringe Belastung der Hardware Das häufige An-/Abschalten der Rechner kann die Hardware auf Dauer belasten. Dies würde zusätzliche Kosten verursachen, weshalb eine übermäßige Beanspruchung vermieden werden sollte.

4) Skalierbarkeit Sowohl die Eigenschaften des ESS-Algorithmus als auch dessen Performance müssen skalierbar sein, um einen Einsatz in größeren Clustern zu ermöglichen.

5) Online-Algorithmus Um in einem Produktivsystem einsetzbar zu sein, muss ein Online-Algorithmus verwendet werden.

6) Reaktionsfähigkeit Das System muss auf spontane Änderungen der Eingabeparameter (Last und Zustand der Rechner) in kürzester Zeit reagieren können.

4.2 Herausforderungen

Aus den Anforderungen ergeben sich folgende Herausforderungen:

1) An-/Abschalt-Entscheidung Die Entscheidung, unter welchen Umständen Rechner zum Sparen abgeschaltet werden und wann sie für das Abfangen von Last wieder angeschaltet werden, ist essentiell für jedes ESS. Werden Maschinen zu spät angeschaltet, gehen mögliche Anfragen verloren und die *QoS* leidet. Bei schlecht konfigurierten Clustern können dadurch auch die aktiven Rechner in Mitleidenschaft gezogen werden. Werden Maschinen hingegen zu früh angeschaltet, wird wertvolle Energie verschwendet. Gleiches gilt umgekehrt für zu frühes oder zu spätes Abschalten von Rechnern. Im SLB-Bereich sind die Entscheidungen besonders schwierig zu treffen, da hier Auslastung der Rechner und Anzahl der eingehenden Jobs viel stärker in die Entscheidungen einfließen müssen als im HPC-Bereich.

2) Balance und Wahl der Parameter Die beiden wichtigsten Anforderungen, Energieeffizienz und *QoS*, sind grundsätzlich konträr zueinander. Es gilt somit, eine möglichst optimale Balance (Pareto-Optimum) zu finden, bei der alle Anforderungen bestmöglich erfüllt sind. Möglich ist auch, dass einzelne Anforderungen für bestimmte Rechenzentren wichtiger sind als andere. Ist dies der Fall, muss eine Gewichtung vorgenommen und die Parameter entsprechend angepasst werden.

3) State Flapping Sogenanntes *State Flapping* [1] stellt eine spezielle pathologische Form der An-/Abschalt-Entscheidung dar. Es handelt sich beim *Flapping* um einen unerwünschten, instabilen Zustand einer Ressource, bei der der physikalische Zustand häufig in kurzer Folge zwischen An und Aus wechselt. Mögliche Ursachen können z.B. schlecht gewählte Parameter sein.
Auch schlecht prozessierte Werte können eine Ursache darstellen. Wenn beispielsweise einzelne Werte zur Entscheidungsfindung genutzt werden und diese auf Grund von stark schwankender Last ein schlechtes Bild der aktuellen Lage widerspiegeln.

4) Performance Das System muss auch für größere Setups problemlos einsetzbar sein. Die Berechnungen und der Algorithmus müssen daher performant und skalierbar sein.

5) Unvermeidbare Verzögerungen Eine große Herausforderung bilden Verzögerungen, die nicht vermieden werden können und sich deshalb auf die Reaktionszeit des ESS auswirken. Unvermeidbare Verzögerungen sind zum einen die Intervalllänge des verwendeten ESS und zum anderen die Bootzeit der verwendeten Hardware.
Die Intervalllänge beeinflusst maßgeblich, ob das System bei einer signifikanten Änderung der Last schnell genug reagieren kann und stets genügend Ressourcen bereitstellen kann. Ein ESS sollte daher möglichst häufig überprüfen, ob An- oder Abschaltungen vorgenommen werden müssen. Die Berechnung für die Entscheidung muss möglichst schnell und effizient durchgeführt werden, da ein Intervall mindestens die Länge dieser Berechnungen haben muss.
Die Bootzeit der verwendeten Hardware ist ebenso entscheidend für die Reaktionszeit, aber lässt sich nur sehr eingeschränkt beeinflussen. Eine Möglichkeit ist, sofern es die Hardware zulässt,

[1] Engl. für Zustandsflattern, kurz *Flapping*, wird auch manchmal als *oscillation* (engl. für Oszillation) bezeichnet.

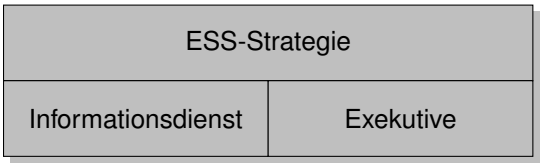


Abbildung 4.1: Notwendige Komponenten eines ESS.

STR zu verwenden.
Für die Reaktionszeit gilt insgesamt: Je kürzer sie ist, desto schneller und besser kann ein System reagieren und desto bessere Resultate sind zu erwarten. Je länger sie ist, desto schwieriger ist es, eine richtige Entscheidung zu treffen, da die Chance für eine Änderung der Last zwischen den Intervallen stark zunimmt.
Komplexe Setups mit Diagnose-Prozeduren können eine sehr lange Zeit für ihren Bootvorgang in Anspruch nehmen (z.B. 45 Minuten in [140]). Solche Systeme sind für den Gebrauch als Webserver nicht geeignet und könnten nur in Umgebungen eingesetzt werden, in denen Lastschwankungen sehr gering oder sehr selten sind.

4.3 Konzept und notwendige Untersuchungen

Abbildung 4.1 zeigt die grundlegenden Komponenten, über die ein ESS verfügen muss. Unter den Informationsdienst fallen Funktionen zur Ermittlung des aktuellen Zustands der Rechner und der aktuellen Last. Auf Basis dieser Informationen kann die Strategie entscheiden, welche Aktionen von der Exekutiven ausgeführt werden sollen.

Den identifizierten Herausforderungen soll mit der in Abschnitt 2.2.3, Abbildung 2.7 (Seite 26) vorgestellten Strategie begegnet werden. Vor Ausführung der Zustandsmaschine (Abschnitt 2.2.3) müssen noch die folgenden wichtigen Schritte durchgeführt werden:

1) Ermittle aktuelle Kapazität.

2) Ermittle aktuelle Last.

3) Berechne zukünftige Last (Lastvorhersage, ab Abschnitt 6.3.1).

4) Berechne Anzahl der nötigen Knoten, um eingestelltes Backup zu erfüllen (ab Kapitel 9).

5) Ermittle Differenz zwischen Kapazität und Last.

6) Ermittle aus der Differenz, vorhergesagter Last und berechnetem Backup die Anzahl der Maschinen, die gestartet oder beendet werden müssen.

7) Markiere die ermittelte Anzahl von Maschinen für die Zustandsmaschine.

Im Zusammenhang mit diesen Schritten ergeben sich folgende Fragen:

I: **Kapitel 6, Seite 78**
- Welche Methode zur Lastermittlung hat das beste Kosten-Nutzen-Verhältnis?

- Sind schwellwertbasierte Verfahren ausreichend oder müssen Vorhersagetechniken für eine Verbesserung angewendet werden?
- Bringen Vorhersagetechniken Vorteile?

II: **Kapitel 7, Seite 96**
- Ist der verwendete Informationsdienst performant genug für große Setups?
- Ist die verwendete Exekutive performant genug für große Setups?

III: **Kapitel 9, Seite 134**
- Ist die Strategie auch in großen Setups effektiv?
- Kann die Strategie Energieeffizienz und Qualität des Dienstes gleichermaßen gewährleisten?
- Wie stark wirkt sich das Backup auf die Ergebnisse aus?

Um die Strategie so durchführen zu können, benötigt es noch folgende Schritte:

1) Einführung eines Überlast- und Unterlast-Schwellwertes
War im HPC-Bereich die Entscheidung, ob Überlast oder Unterlast herrscht, noch binär [2], ist diese Identifikation im SLB-Bereich auf Grund der Kürze und Menge an Jobs schwieriger. Um Entscheidungen treffen zu können, müssen daher zwei Schwellwerte eingeführt werden.
Der *Überlast-Schwellwert* wird eingeführt, um zu markieren, wann sich eine Maschine in einer Überlastsituation befindet und wann nicht. Wird der Schwellwert überschritten, herrscht Überlast und neue Ressourcen müssen angeschaltet werden. Wird er unterschritten, herrscht keine Überlast und es müssen keine Aktionen durchgeführt werden.
Der *Unterlast-Schwellwert* ist für die Ermittlung einer Unterlastsituation gedacht. Wird dieser Schwellwert unterschritten, ist die Ressource unterfordert und kann abgeschaltet werden. Ist der Schwellwert überschritten, herrscht keine Unterlast und es muss keine Aktion durchgeführt werden.

Änderung an CHERUB
Diese beiden Schwellwerte werden als konfigurierbare Parameter in *CHERUB* eingeführt und werden in *Anfragen* (Anf) angegeben.

Parameterbezeichnung:	Überlast-Schwellwert
Quellcodebezeichner:	`more_than_ActivConn_is_overload`
Standardwert:	Verschieden, ab Kapitel 9 (Seite 134) 25 Anf
Bedeutung:	Entscheidet in der Last-Funktion mit darüber, ob eine Überlast-Situation vorliegt. Ist dies der Fall, wird die Last-Markierung von genügend Maschinen gesetzt, die zur Zeit nicht *BUSY* oder *ONLINE* sind, um die auftretende Last zu bewältigen. Dieser Parameter wird NICHT für die Unterscheidung von Zuständen einzelner Maschinen genutzt.

[2] Im HPC-Bereich gilt, Überlast = mehr Jobs als Maschinen verfügbar, Unterlast = Maschine hat keinen Job

Parameterbezeichnung:	Unterlast-Schwellwert
Quellcodebezeichner:	`less_than_ActivConn_is_idle`
Standardwert:	Verschieden, ab Kapitel 6.2.3 (Seite 83) automatisch durch *CHERUB* konfiguriert. Angabe in Anf.
Bedeutung:	Dieser Parameter entscheidet darüber, ob für eine bestimmte Maschine eine Unterlast-Situation eingetreten ist und demzufolge darüber, ob sie den Zustand *BUSY* oder *ONLINE* besitzt.

Da der Unterlast-Schwellwert für die Unterscheidung der Zustände *BUSY* und *ONLINE* benötigt wird, muss die Definition der Zustände (siehe Abschnitt 2.2.3, Seite 25) leicht angepasst werden:

- **BUSY:** Die Maschine ist am RMS angemeldet und die aktuelle Last liegt über dem Unterlast-Schwellwert. Die Maschine darf auf keinen Fall abgeschaltet werden.
- **ONLINE:** Die Maschine ist am RMS angemeldet und die aktuelle Last liegt unter dem Unterlast-Schwellwert. Die Maschine kann vom RMS abgemeldet werden und ihre Restlast muss von den anderen Maschinen abgefangen werden.

Mit dieser Anpassung ist *CHERUB* nun in der Lage, im SLB-Bereich die aktuelle Auslastung und den aktuellen Zustand der Knoten ermitteln zu können.

2) Anti-Flapping Maßnahmen
Um *Flapping* zu vermeiden, hilft in erster Linie eine gute Konfiguration. Wie bereits erwähnt, können schlechte Parameter dies begünstigen. Mit Einführung der Schwellwerte gibt es eine weitere Möglichkeit Fehler, bei der Konfiguration zu machen. Beispiel 6 soll dies verdeutlichen.

Beispiel 6 Bei der Verwendung von zwei Back-End-Maschinen und einem Unterlast-Schwellwert von 40 Anf und einem Überlast-Schwellwert von 50 Anf wird eine Gleichverteilung aller Anfragen angenommen. Bei einer stetigen Last von 60 Anfragen pro Sekunde (Anf/s) wird, wenn beide Maschinen an sind, jede Maschine 30 Anf/s erhalten. Da der Unterlast-Schwellwert nun unterschritten ist, wird eine Maschine abgeschaltet und die Last nun von der verbleibenden Maschine übernommen. Diese Maschine erhält die vollen 60 Anf/s und überschreitet somit den Überlast-Schwellwert. Wenn die zweite Maschine wieder angeschaltet wird, übernimmt diese erneut die Hälfte der Last und der Vorgang beginnt erneut. □

Inwiefern dies zu einem Problem wird, muss analytisch und anhand von Experimenten ermittelt werden. Als erste Maßnahme werden *CHERUBs* Verzögerungsparameter für das Abmelden, Herunterfahren und Hochfahren auf 1 Minute gestellt. Da der Parameter für das Hochfahren selbiges verzögert und damit verhindert, dass womöglich benötigte Ressourcen sofort gestartet werden, ist dessen Untersuchung besonders interessant. Wie sehr dieser ins Gewicht fällt, wird in Kapitel 9 (Seite 134) untersucht.
Zusätzlich wird ab Abschnitt 6.2.3 (Seite 83) ein dynamischer Unterlast-Schwellwert eingeführt, welcher abhängig von der Anzahl der aktiven Rechner und dem Überlast-Schwellwert ist und daher Situationen wie in Beispiel 6 verhindert.

3) Maßnahmen zur Vermeidung von Verzögerungen
Die beiden erwähnten unvermeidbaren Verzögerungen (Intervalllänge und Bootzeit) müssen so

stark wie möglich reduziert werden. Die Intervalllänge ist dabei problemlos, da sie als Parameter einfach konfiguriert werden kann. Hier muss nur beachtet werden, dass innerhalb des Intervalls genügend Zeit für die Berechnungen bleibt.
Die Bootzeit hingegen kann typischerweise nicht verändert werden. Die damit einhergehende Trägheit des Systems soll zum einen anhand von Experimenten untersucht werden und zum anderen soll ermittelt werden, ob eine Vorhersage der Last die vorhandenen Bootzeiten möglicherweise kaschieren kann.

Änderung an CHERUB

Für Techniken, die die Bootzeit der einzelnen Maschinen berücksichtigen, muss diese als Parameter angegeben werden. Als neuer *CHERUB*-Parameter wird diese deshalb nun eingeführt.

Parameterbezeichnung:	Bootzeit
Quellcodebezeichner:	`boot_duration`
Standardwert:	abhängig von genutzter Hardware
Bedeutung:	Gibt an, wie lange die verwendete Hardware benötigt, um zu starten. Die Dauer wird ab Abschnitt 6.3.1 (Seite 86) für die Lastvorhersage benötigt.

Wie erwähnt, ist das Intervall nur zu konfigurieren. Für den HPC-Fall wurde in der Vergangenheit 1 Minute als Intervall verwendet. Für einen Webserver ist dies aber zu träge, so dass die Zeit auf 15 Sekunden gesenkt wird. Es wird geschätzt, dass *CHERUB* so genügend Zeit hat, um alle benötigten Informationen zu sammeln oder zu berechnen. Ob die 15 Sekunden tatsächlich genügen, muss im Laufe der Arbeit beobachtet werden.

4) Wahl der Parameter auf Basis von Experimenten

Wie bereits in den Herausforderungen erwähnt, ist die richtige Wahl der Parameter eine sehr schwere Aufgabe. Obwohl es Parameter gibt, deren Konfiguration recht eindeutig ist, sind vor allem die Parameter schwierig zu wählen, die schlussendlich über das An-/Abschalten und damit zwischen Energieeffizienz und Qualität des Dienstes entscheiden.
Um diese konträren Parameter abhängig von einer gewählten Strategie möglichst optimal wählen zu können, wird eine Parameterstudie durchgeführt, deren gewichtete Ergebnisse Aufschluss über die Wahl der Parameter für die verschiedenen Strategien geben soll. Untersucht werden dabei die Parameter Dauer der Bootzeit, Größe des Backups, aggressives oder langsames Abschalten und ob ein Bootbefehl sofort oder erst nach mehrfach ermittelter Überlast gesendet werden soll.

5) Komponententests

Die Performance und die damit verbundene Skalierbarkeit kann mit Hilfe von Komponententests überprüft werden. Hier muss jede Funktion, deren Laufzeit von der Größe des zu verwaltenden Clusters abhängt, untersucht werden.

Das in diesem Kapitel vorgeschlagene Konzept wird nun im weiteren Verlauf der Arbeit Stück für Stück umgesetzt, überprüft und, wenn nötig, angepasst werden.
Wie das zur Überprüfung der aufgeworfenen Fragen verwendete Setup aussieht und wie die Parameter abhängig von diesem Setup gewählt werden müssen, wird im kommenden Kapitel gezeigt.

5 Beschreibung des Aufbaus der Messexperimente

Um das geeignetste Verfahren zur Lastermittlung zu bestimmen, werden Messexperimente durchgeführt. In diesem Kapitel werden der Aufbau, die verwendeten Komponenten und die relevanten Metriken für die in Kapitel 6 folgenden Messungen beschrieben. Beginnend mit der verwendeten Hardware wird danach die verwendete Arbeitslast vorgestellt. In diesem Zusammenhang wird auch das Hardware Setup auf seine Leistung hin untersucht. Zum Schluss werden die verwendeten Metriken vorgestellt und diskutiert.

5.1 Verwendete Hardware und Software

Die Experimente werden auf Teilen des *Infiniband*-Clusters (IB) und des *Leibniz*-Clusters des Instituts für Informatik an der Universität Potsdam durchgeführt. Tabelle 5.1 zeigt die Spezifikationen der einzelnen Maschinen. Die Zeit, die die Back-End-Maschinen zum Booten benötigen, beträgt 78 Sekunden [1]. Abbildung 5.1 stellt das Setup anschaulich dar.

Die Knoten sind untereinander über Gigabit Ethernet verbunden (das namensgebende Infiniband wird für das Experiment nicht verwendet). Vom *Leibniz*-Cluster ist der Front-Knoten involviert, welcher als Last-Generator dient. Als Werkzeug für die Lasterzeugung wird `servload/0.5` [76] (siehe Abschnitt 2.2.6, Seite 32) verwendet. Als Lastverteiler ist die IB1 im Einsatz. Hier läuft der Linux Virtual Server (*LVS/1.2.1* siehe Abschnitt 2.2.1, Seite 16) und verteilt eingehende Anfragen von Leibniz an die Back-Ends IB7 und IB8. Die IB7 und IB8 haben je eine Wikipedia-Instanz vom 03.01.2008 installiert, die etwas mehr als 6 Millionen englische Wikipedia-Artikel umfasst. Die Software, die benötigt wird, um diese Instanz verfügbar zu machen, kann aus Tabelle 5.2 entnommen werden. Verwendet wird Apache in der Version 2.2.3. Die aktuelle Version ist 2.4.23, die Unterschiede zwischen den 2.2.x und den 2.4.x Versionen liegen im Wesentlichen in der Einführung des *MPM event* (siehe Abschnitt 2.2.2.1, Seite 21), verbessertem Protokollieren und einer Reihe neuer Module [141]. Alle diese Funktionalitäten werden im Laufe der Arbeit nicht verwendet, weshalb die Verwendung einer älteren Version keinen Einfluss auf die Ergebnisse der Arbeit haben sollte.

Die verwendete Konfiguration ist in Tabelle 5.3 zu sehen. Nicht angegebene Parameter sind weniger wichtig und wurden nicht verändert. Eine Optimierung der Apache-Parameter ist nicht erforderlich, da es in den Experimenten nicht um die Optimierung der Performance des Setups geht.

[1] Die IB7/IB8 benötigen ~77,6 s (Mittelwert aus 5 Werten, Standardabweichung 0,548 s), bis sie betriebsbereit sind.

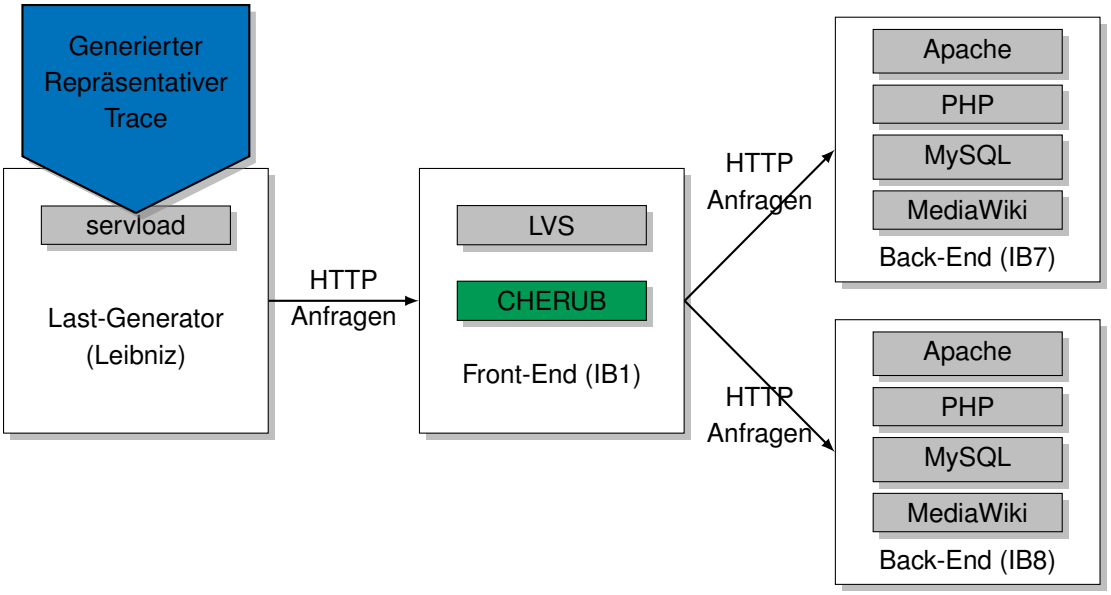


Abbildung 5.1: Messaufbau mit verwendeter Hardware und Software.

Name	Komponente	System	CPU
Leibniz	Last-Generator	Linux 2.6.26-2-amd64	AMD Opteron(tm) (2x 1,8GHz)
IB1	Front-End (LVS)	Linux 2.6.18-274.12.1.el5	AMD Opteron(tm) (2x 1,8GHz)
IB7	Back-End (Wiki)	Linux 2.6.18-274.12.1.el5	Intel(R) Xeon(R) (2x 1,86GHz)
IB8	Back-End (Wiki)	Linux 2.6.18-274.12.1.el5	Intel(R) Xeon(R) (2x 1,86GHz)

Tabelle 5.1: Verwendete Rechner, alle sind mit 4GB RAM ausgestattet.

Name	Version	Release
Apache (httpd)	2.2.3	53.el5.centos.3
PHP	5.1.6	27.el5_5.3
MySQL	14.12	4.el5_6.6
Mediawiki	1.16.5	-

Tabelle 5.2: Verwendete Software auf den Back-Ends.

Name	Wert	Name	Wert
StartServers	8	ServerLimit	96
MinSpareServers	5	MaxClients	96
MaxSpareServers	20	ListenBacklog	511
MaxRequestsPerChild	4000	KeepAlive	Aus
Multi-Processing Module	prefork		

Tabelle 5.3: Verwendete Apache-Parameter.

5.2 Leistung des Setups

Obwohl die Performance des Setups nicht optimiert werden soll, muss die Leistung dennoch ermittelt werden. Die Leistung des Setups muss bekannt sein, damit 1) *CHERUBs* Parameter entsprechend sinnvoll eingestellt werden können und 2) die Arbeitslast des Experimentes später so gewählt werden kann, dass das Setup voll ausgelastet wird.
Für diese Voruntersuchung wurde von Leibniz aus mit dem Tool `http_load` [80] (siehe Abschnitt 2.2.6, Seite 32) die IB7 und IB8 (baugleich) via der IB1 unter Last gesetzt. Dafür wählt `http_load` aus 10.000 Adressen des installierten Wikis zufällige aus, welche dann angefragt werden. Die verwendete Version von `http_load` ist vom 12.03.2006 und beinhaltet einen Patch, der es ermöglicht, einen festen *Seed* für die Zufallswerte zu setzen. Dadurch ist die Reihenfolge der Anfragen bei jedem Durchlauf gleich und damit besser vergleichbar.
Jede Anfrage, die länger als 1 Sekunde benötigt, wird als Zeitüberschreitung gewertet. Obwohl für die späteren Messungen eine Zeitüberschreitung von 5 Sekunden gilt, wird hier erst ermittelt, wie die Leistung der Rechner in *Anfragen pro Sekunde* sind. Daher gilt hier die erwähnte Zeitüberschreitung von 1 Sekunde

5.2.1 Performance der IB7

Als erstes wurde die Leistung eines einzelnen Rechners untersucht. Dafür wurde nur die IB7 unter Last gesetzt und die IB8 vom LVS abgemeldet. Das Experiment besteht aus 100 Wiederholungen, die jeweils 30 Sekunden dauern. In den 30 Sekunden werden *n* Anf/s verschickt. Dabei entspricht *n* der Nummer der Wiederholung. Eine Anfrage, die nach einer Sekunde nicht beantwortet ist, wird als Timeout gewertet. Wie man in Abbildung 5.2 (blaue Kurve) deutlich erkennen kann, bricht die Performance ab ~30 Anf/s erheblich ein. Mehr Anfragen kann die Maschine pro Sekunde nicht bearbeiten, ohne dass es zu Verlusten von Anfragen kommt.

5.2.2 Gemeinsame Performance der IB7 und IB8

Da in den Hauptmessungen die Situation eines Webserver-Clusters in Betrieb nachgeahmt wird, muss auch die kombinierte Leistung der Rechner getestet werden. Das Vorgehen entspricht demselben wie bei der Ermittlung der Performance der IB7. Der Unterschied hier ist, dass beide Rechner genutzt werden (beide sind am LVS angemeldet) und zwischen den beiden Verteilungsalgorithmen Round-Robin und Least-Connection unterschieden wird.

Round-Robin (RR) Beim RR-Verfahren werden die Anfragen immer abwechselnd an die IB7 und die IB8 gesendet. Dies kann bei hoher Last und verschiedenartigen Anfragen (rechenintensive und weniger rechenintensive Anfragen) zu einer ungleichmäßigen Belastung der Maschinen führen. Wie man in Abbildung 5.2 (rote Kurve) erkennen kann, bricht bei der Nutzung beider Maschinen mit RR die Performance erst ab ~45 Anf/s ein.

Least-Connection (LC) Beim LC-Verfahren wird dem Rechner eine neue Anfrage zugewiesen, der zur Zeit am wenigsten aktive Verbindungen hat. Dieses Vorgehen soll sich bei hoher Last mit verschiedenen Anfragen, im Gegensatz zu RR, positiv auswirken. Wie man in 5.2 (grüne Kurve)

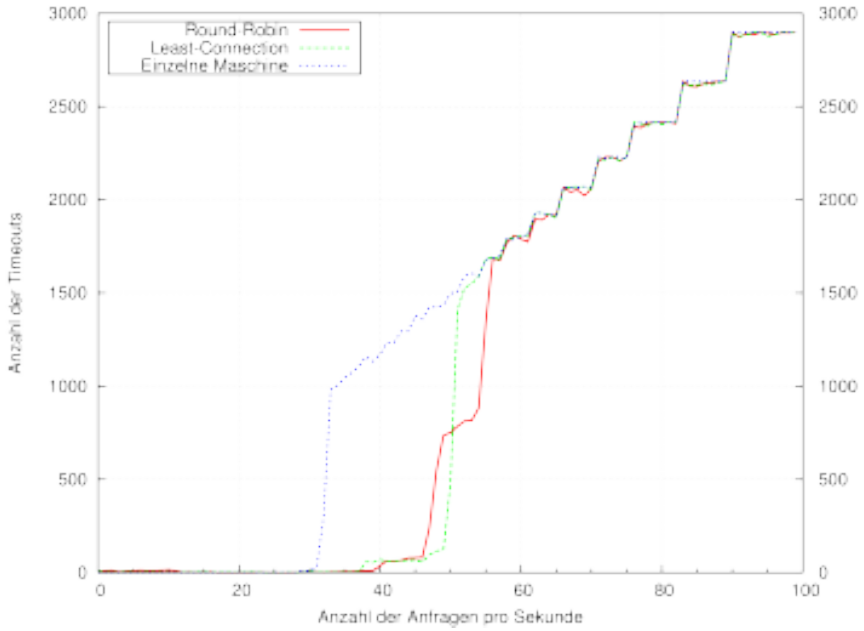


Abbildung 5.2: Leistung von IB7 und IB8 mit Least-Connection gegenüber Round-Robin und gegenüber einer einzelnen Maschine.

erkennen kann, bricht bei der Nutzung von LC die Performance in diesem Fall bei ~48 Anf/s ein. LC schneidet hier demnach minimal besser ab (etwa 6,66% mehr Anfragen) als RR.
Die Literatur ist sich bei dem Vergleich beider Algorithmen nicht einig, welcher der bessere ist [142, 143]. Beide Arbeiten präsentieren Simulationsergebnisse mit bis zu 8 ([142]) und 32 ([143]) Rechnern und verwenden für ihre Überprüfung jeweils mehrere verschiedene Arbeitslasten.

5.2.3 Zusammenfassung der Leistungsanalyse

Die Leistungsanalyse zeigt, dass eine Nutzung von LC bevorzugt werden sollte (~48 Anf/s vs. ~45 Anf/s). Bei hoher Last werden die Rechner bei RR schlechter ausgelastet. Dies kann am früheren Anstieg der Timeouts bei ~45 Anf/s und an dem darauf folgenden geringeren Anstieg der Timeouts um ~50 Anf/s erkannt werden. Hier ist eine der beiden Maschinen bereits überlastet, was zu den erhöhten Timeouts führt. Da RR nur minimale Zustandsinformationen verwendet, werden der überlasteten Maschine weiterhin Anfragen zugewiesen. Erst ab ~55 Anf/s ist auch die zweite Maschine überlastet und die Timeouts nehmen wieder stärker zu. Im Gegensatz dazu hält LC die Performance etwas länger und bricht dann ebenfalls ein.
Überraschend ist, dass obwohl eine einzelne Maschine erst ab ~30 Anf/s überlastet ist, zwei Maschinen schon deutlich vor 60 Anf/s anfangen, Timeouts zu verursachen. Da die Konfiguration des Webservers aber, wie bereits erwähnt, nicht optimiert wurde, könnten hier eventuell noch bessere Ergebnisse erzielt werden. Ausgehend von der gemeinsamen Performance von etwa 50 Anf/s, wird für die kommenden Messungen eine Leistung der einzelnen Maschinen von 25 Anf/s angenommen.

Parameter	Wert	zukünftig Referenziert als
update_everything	15 s	Intervall
wait_for_sign_off	1 min	-
wait_for_shutdown	1 min	-
wait_for_boot	1 min	-
wait_for_register	0 min	-
more_then_ActivConn_is_overload	15 Anf	Überlast-Schwellwert
less_then_ActivConn_is_idle	5 Anf	Unterlast-Schwellwert
sequential_shutdown	1	sequenzielles Abschalten
boot_duration	120	Bootzeit

Tabelle 5.4: CHERUBs Parameter.

5.3 Initiale Konfiguration von CHERUB

Durch die Leistungsanalyse können nun auch die *CHERUB*-Parameter gesetzt werden. Für die Messungen, bei denen *CHERUB* auf dem Front-End angeschaltet ist (siehe Abbildung 5.1), können die Einstellungen der Parameter Tabelle 5.4 entnommen werden.
Mit 15 Sekunden wurde ein recht kurzes Intervall gewählt, hierdurch soll eine schnelle Reaktion des Systems auf Last-Unterschiede sichergestellt werden.
Für die Verzögerungsparameter, die auf 1 Minute gestellt wurden, gilt durch das Intervall von 15 Sekunden, dass viermal in Folge der selbe Schwellwert bei der selben Maschine über- bzw. unterschritten werden muss, bis eine der korrespondierenden Aktionen auch ausgeführt wird. Für wait_for_register gilt allerdings, dass es sinnvoller ist, eine Maschine, die schon an ist, auch sofort wieder zu aktivieren, wenn sie benötigt wird. Daher ist dieser Parameter auf 0 Minuten gestellt.
Das Booten eines Rechners benötigt 78 Sekunden und der Verzögerungsparameter für das Booten zögert einen möglichen Start weiter hinaus. Um dem gerecht zu werden, wird die `boot_duration` auf 120 Sekunden gesetzt. Diese wird aber, wie bereits erwähnt, erst ab Abschnitt 6.3.1 (Seite 86) verwendet.
Obwohl die Rechner eine gemessene Kapazität von 25 Anf/s (siehe Abschnitt 5.2, Seite 66) aufweisen, wird der Überlast-Schwellwert für die Experimente mit 15 Anf sehr konservativ gewählt. Dies entspricht 60 % der ermittelten maximalen Leistung und entspricht damit der Größenordnung der Schwellwerte, welche zumeist in der Literatur vorzufinden sind (siehe Kapitel 3, Seite 37). Durch diese Wahl sollte *CHERUB* genügend Zeit haben, um Ressourcen bei Bedarf schnell genug zu aktivieren. Im Laufe der Experimente wird dieser Parameter durch Verbesserungen noch auf 20 Anf (80 % der maximalen Leistung) erhöht. Auch der Unterlast-Schwellwert ist mit 5 Anf (20 %) sehr konservativ gewählt und wird ebenfalls im weiteren Verlauf der Experimente angepasst (7,5 Anf (30 %) bzw. 10 Anf (40 %)). Die Veränderungen werden in den jeweiligen Messungen noch einmal explizit erwähnt und motiviert.
Zusätzlich zu den erwähnten Parametern wurde die IB7 so konfiguriert, dass sie nicht heruntergefahren werden kann. Dies ist in den ersten Messungen notwendig, damit nicht beide Rechner gleichzeitig abgeschaltet werden. Im Verlaufe der Experimente wird dafür eine elegantere Lösung vorgestellt (siehe Abschnitt 6.2.3, Seite 83). Als Last eines Back-Ends wird immer die Anzahl an Anfragen verwendet, die innerhalb von einer Sekunde an das Back-End weitergegeben wur-

den. Die Einheit in der die Last gemessen wird und in der die Schwellwerte vorliegen, ist somit *Anfragen* und nicht *Anfragen pro Sekunde*.

5.4 Arbeitslast

Die Arbeitslast (oder engl. workload), die in Form eines Traces vorliegt und durch das Tool `servload` in den Experimenten wieder eingespielt wird, muss drei Anforderungen erfüllen:

1. Der Trace muss so realistisch wie möglich sein.
2. Der Trace muss vom verwendeten Setup verarbeitet werden können.
3. Das Profil des Traces muss eine Lastspitzen-Situation widerspiegeln.

Während Anforderung 1 und 2 nahe liegen und keiner näheren Erläuterung bedürfen, ist Anforderung 3 ebenfalls von äußerster Wichtigkeit. Eine Lastspitzen-Situation stellt die schwierigste Situation dar, die in einem Webserver-Szenario auftreten kann. Andere Situationen, die auftreten können, sind:

A: Schwacher Last-Anstieg, hier hat das verwendete ESS viel Zeit zu reagieren.

B: Kein Anstieg der Last, hier muss das ESS nichts unternehmen.

C: Last sinkt, hier besteht ebenfalls keine Gefahr, dass Anfragen verloren gehen.

D: Stark schwankende Last, in diesem Fall ist die Wellenlänge entscheidend. Bei einer langen Welle entspricht die Situation der einer immer wiederkehrenden Lastspitzen-Situation (welche von den Betrachtungen abgedeckt wird). Bei einer kurzen Wellenlänge handelt es sich um ganz normale Schwankungen, die in jedem natürlichem Trace enthalten sind.

Somit bleibt als einzige kritische Situation, die untersucht werden muss, die in Anforderung 3 genannte Lastspitzen-Situation übrig.

5.4.1 Vorhandene Traces

Da es sich um ein Labor-Setup und nicht um ein Produktiv-Setup handelt, liegen auch keine Mitschnitte von realem Datenverkehr vor, der die drei genannten Anforderungen erfüllt. Um die Anforderungen trotzdem zu erfüllen, wurden zwei vorliegende Traces verwendet und kombiniert. Bei den Traces handelt es sich um:

1. einen Trace eines Clusters von Wikipedia (im Folgenden als Wikipediatrace bezeichnet),
2. einen Trace der Firma Iconmobile[2] [144] (im Folgenden als Iconmobiletrace bezeichnet).

Beide Traces liegen in einer anonymisierten Form, im sogenannten *Common Logfile Format* [77, 78] vor. Beispiel 7 zeigt, wie dieses Format aufgebaut ist.
Wichtig sind an dieser Stelle das Feld mit der Information über den Zeitpunkt der Anfrage ([date]) und das Feld, welches die angefragte Datei in Form des Pfades enthält ("request"). Andernfalls kann der Trace nicht wieder korrekt eingespielt werden.

[2] Iconmobile ist eine weltweit operierende Firma, die vor allem im Mobilfunkmarkt tätig ist.

Beispiel 7

```
1 | remotehost rfc931 authuser [date] "request" status bytes
2 | ...
3 | www.salbnet.org - 5396292430 [12/Nov/2007:19:31:30.224000000 +0100]
  | "GET /w/images/thumb/8/8b/AlterOrient.jpg/350px-AlterOrient.jpg HTTP/1.1" 200 -
4 | www.salbnet.org - 5396292438 [12/Nov/2007:19:31:30.227000000 +0100]
  | "GET /w/thumb/4/4a/Commons-logo.svg/20px-Commons-logo.png HTTP/1.1" 200 -
5 | ...
```

□

Wikipediatrace Dieser Trace erfüllt Anforderung 1 und 2. Er ist realistisch, da er 1) real erfasst und nicht generiert wurde und er 2) von unserem Setup verstanden werden kann, da dieses die dazu korrespondierende Wikipedia-Instanz bereit stellt. Anforderung 3 wird nicht erfüllt, da der Trace eine konstante Last von ~2500-2800 Anf/s umfasst. Der Trace umfasste ~9,6 Mio Anfragen. Eine nähere Analyse und weitere Informationen zum Wikipedia-Projekt findet sich in [145].

Iconmobiletrace Dieser Trace erfüllt Anforderung 1 und 3. Auch er ist realistisch, da er real erfasst und nicht generiert ist. Zusätzlich beinhaltet er mehrere mitgeschnittene Lastspitzen-Situationen. Die einzelnen Lastspitzen sind praktisch von identischem Profil. Dieser Trace erfüllt allerdings nicht Anforderung 2. Iconmobile bietet auf ihren Servern nicht Wikipedia an, weshalb die enthaltenen Anfragen nicht von unserem Setup verwendet werden können.

5.4.2 Kombinierter finaler Trace

Um einen Trace zu erzeugen, der alle 3 Anforderungen erfüllt, wurden beide Traces in mehreren Schritten kombiniert.

Schritt 1: Als erstes wurde ein Profil des Iconmobiletraces erzeugt, welches eine der Lastspitzen-Situationen umfasst. Beispiel 8 zeigt das Format des Profils. Durch das Profil ist bekannt, zu welchem Zeitpunkt (sekundengenau) wie viele Anfragen am System ankommen. Ein zusätzliches Feld (Second_ID) nummeriert die Zeit innerhalb des Profils und wird später, anstelle der eigentlichen Zeit verwendet. Die Spalte *Time* kann daher ignoriert werden. Das Profil umfasst 15 Minuten und seine Lastspitze liegt bei ~160 Anf. Dauer und Ausschlag sind daher noch ungeeignet. Bei einer Bootzeit von 2 Minuten würde diese bereits 13,3 % des gesamt Traces ausmachen und etwa 50 % der Last-Anstiegs-Phase (Phase 2). Der verbleibende Spielraum für *CHERUB* wäre damit zu gering und ein rechtzeitiges Anschalten von Maschinen praktisch nicht möglich. Der zu testende Zeitraum soll 30 Minuten umfassen, und das verwendete Setup kann, wie in Abschnitt 5.2.3 gezeigt, nur maximal ~50 Anf/s verarbeiten.

Schritt 2: Um die Dauer zu erhöhen, wurde in dem Profil jeder Eintrag dupliziert und dessen ID angepasst. Die führt zu einer Verdoppelung der Dauer und würde aus Beispiel 8, Beispiel 9 machen. Zusätzlich senkt dies die Stärke des Anstiegs, was die Situation für das ESS fairer gestaltet. Im Vergleich zu anderen Arbeiten in diesem Umfeld (siehe Kapitel 3), ist der verwendete Anstieg aber immer noch sehr stark. In anderen Arbeiten nimmt die Last über Stunden hinweg langsam

Beispiel 8

Time	Second_ID	Requests
16/Jul/2010:15:01:20	1	25
16/Jul/2010:15:01:21	2	15
16/Jul/2010:15:01:22	3	16

□

Beispiel 9

Time	Second_ID	Requests
16/Jul/2010:15:01:20	1	25
16/Jul/2010:15:01:20	2	25
16/Jul/2010:15:01:21	3	15
16/Jul/2010:15:01:21	4	15
16/Jul/2010:15:01:22	5	16
16/Jul/2010:15:01:22	6	16

□

Phase	Dauer	Anstieg	durchschnittliche Last	Standardabweichung
1	5 min	-	0,47 Anf/s	0,77 Anf/s
2	8,33 min	0,076 Anf/s	26,60 Anf/s	12,04 Anf/s
3	16,66 min	-0,027 Anf/s	18,38 Anf/s	8,95 Anf/s

Tabelle 5.5: Die drei Phasen des Traces.

zu. In dem verwendeten Trace wird das Maximum schon in weniger als 9 Minuten erreicht (siehe Tabelle 5.5, Phase 2).

Schritt 3: Um die maximale Last des Traces der Leistung des Setups anzupassen, wurde das Profil weiter manipuliert. Dabei wurde der gesamte Trace ausgedünnt, indem an jedem Zeitpunkt nur 31% der Anfragen verwendet wurden. Dadurch befindet sich das Maximum des Profils nur noch bei ~50 Anf. Das Profil, und somit die Lastspitzen-Situation, wird dabei beibehalten.

Schritt 4: Das so erzeugte Profil kann nun verwendet werden, um einen neuen Trace aus dem vorhandenen Wikipediatrace zu extrahieren. Dazu wurden pro Zeitpunkt vom Wikipediatrace so viele Anfragen verwendet, wie das in Schritt 3 und 4 überarbeitete Profil vom Iconmobiletrace vorgibt. Wie bereits erwähnt, beinhaltet der Wikipediatrace pro Sekunden ~2500-2800 Anfragen. Dadurch sind immer genügend Anfragen pro Sekunde des Profils verfügbar. Der so erzeugte neue Trace hat nun genau das selbe Profil wie das überarbeitete Profil, aber besteht nun aus Wikipedia-Anfragen, welche vom Setup verarbeitet werden können.

Abbildung 5.3 zeigt das Profil des neuen Traces, welcher nun zum Wiedereinspielen verwendet werden kann. Der Trace umfasst 31.806 Anfragen, von denen 13.170 disjunkt sind. Der Verlauf des Traces kann in drei Phasen unterteilt werden. Die erste Phase ist durch eine stete und sehr

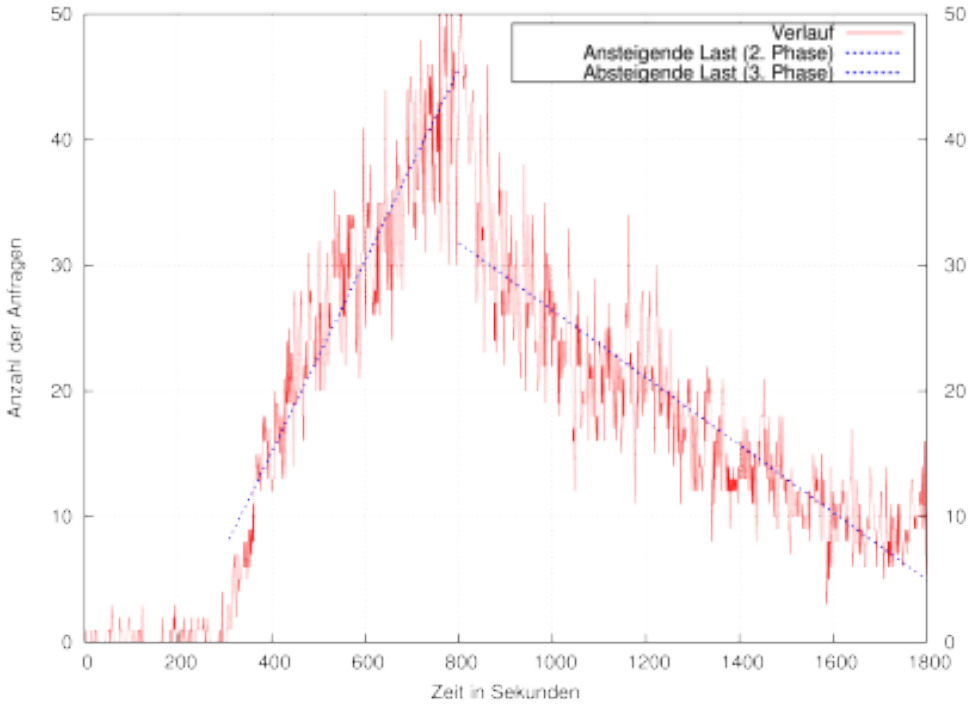


Abbildung 5.3: Das verwendete Lastspitzen-Profil.

geringe Last ausgezeichnet. Phase zwei stellt anschließend die kritische Situation dar, in der es einen starken Anstieg der Last gibt. Die letzte Phase umfasst die Entspannung, in der die Last wieder konstant abnimmt. Lediglich bei Sekunde 1200 ist noch einmal ein kleiner Anstieg zu verzeichnen. Tabelle 5.5 enthält die zu den Phasen gehörigen Werte.

5.4.3 Verteilung der Anfragen

Um eine bessere Übersicht über die Last der einzelnen Anfragen zu erhalten, wurden 10.000 Anfragen des Traces zufällig ausgewählt und deren Antwortzeiten überprüft. Abbildung 5.4 zeigt, dass ein Großteil der Anfragen (13,24 %) eine Antwortzeit von unter 0,01 Sekunden besitzt. 64,84 % der Anfragen haben eine Antwortzeit von unter 0,1 Sekunde und nur 2,12 % der Anfragen benötigen länger als eine Sekunde, bis die Antwort vorliegt. Der Median der Anfragen liegt bei 0,02 Sekunden, was noch einmal unterstreicht, dass ein Großteil der Anfragen in sehr kurzer Zeit beantwortet werden können. Die Messungen wurden in dem vorliegenden Setup mit Hilfe des Tools `wget` durchgeführt. Da nur eine einfache Übersicht erstellt werden sollte, wurden die Messungen nicht wiederholt.

5.5 Optimale Downtime / Optimale Energieeinsparung

Bei der optimalen Downtime bzw. der optimalen Energieeinsparung handelt es sich um die Zeit (bzw. die dazu korrespondierende Energie), die Maschinen in einem vorhandenen Setup maximal

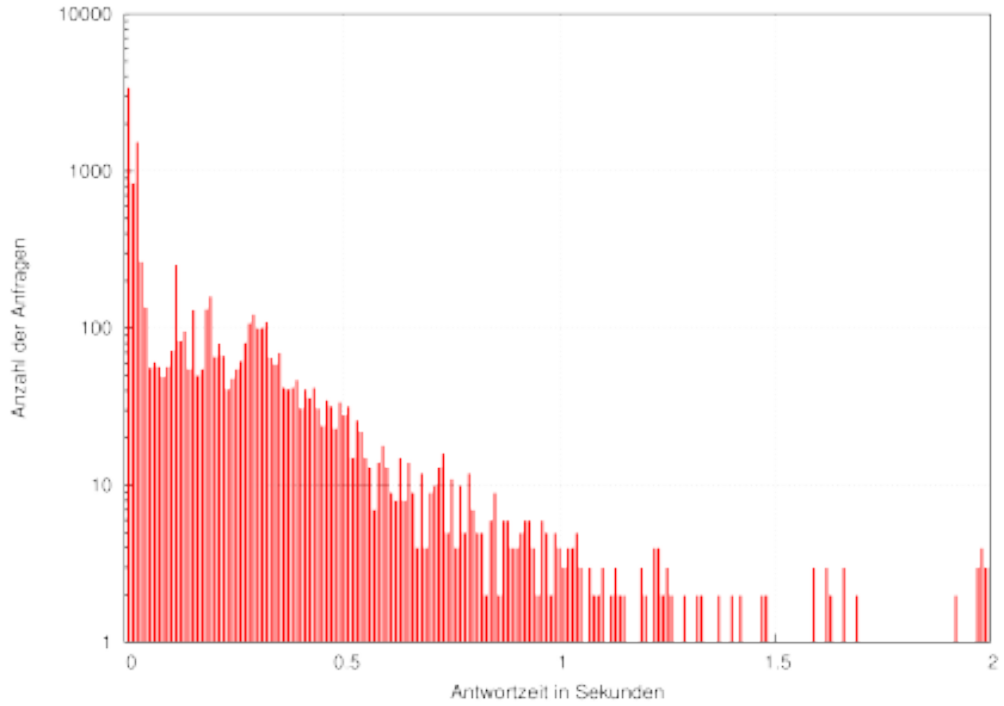


Abbildung 5.4: Verteilung der Antwortzeiten der Anfragen (Y-Achse ist logarithmisch).

abgeschaltet sein können, ohne dass der angebotene Dienst auf Grund des Abschaltens der Ressourcen leidet. Die Höhe hängt dabei von der Arbeitslast und der verwendeten Hardware ab. Je schneller eine Maschine an- bzw. ausgeschaltet werden kann, desto höher ist die optimale Downtime, da Schwankungen innerhalb der Arbeitslast genutzt werden können, um überflüssige Maschinen in dieser Zeit abzuschalten.

Wenn es Maschinen geben würde, die praktisch ohne Verzögerung einsatzbereit wären (z.B. weil ihr Bootvorgang im Millisekundenbereich liegt), könnte die optimale Downtime wie folgt berechnet werden:

$$T_{opt\_down} = \sum_{t=1}^{t=T_{Dauer}} f_{An}(t) \tag{5.1}$$

Dabei entspricht T_{Dauer} der Länge der Arbeitslast und $f_{An}(t)$ berechnet, wie viele Maschinen zum gegebenen Zeitpunkt t angeschaltet sein müssen, um die zu diesem Zeitpunkt anfallende Last erfolgreich zu bearbeiten. Für die verwendete Arbeitslast ergibt sich eine Downtime von 1318 Sekunden (21,97 Min).

Da das Hochfahren aktueller Hardware allerdings bis zu mehreren Minuten dauern kann, muss die Funktion f_{An} durch $f_{An\_Erweitert}$ ersetzt werden. $f_{An\_Erweitert}$ gibt ebenfalls die Anzahl der Maschinen zurück, welche aktuell für die Lastverarbeitung benötigt werden, berücksichtigt dabei aber die Startzeit der vorhandenen Maschinen. Daraus ergibt sich die der Formel 5.1 fast identische

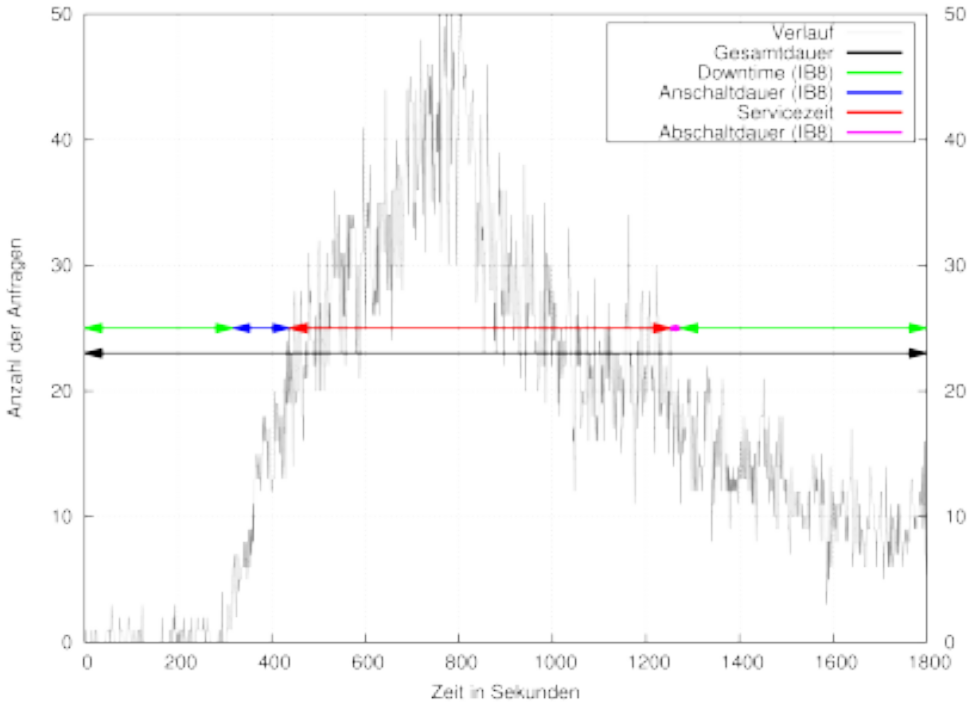


Abbildung 5.5: Darstellung der einzelnen Zeitabschnitte bezüglich der maximalen Downtime.

Formel:

$$T_{opt\_down} = \sum_{t=1}^{t=T_{Dauer}} f_{An\_Erweitert}(t) \tag{5.2}$$

Wenn die Last an einer Stelle im verwendeten Trace sinkt und damit Maschinen abgeschaltet werden könnten, darf dies nicht geschehen, wenn die Dauer der Last-Senke geringer ist als die Zeit, die die Maschinen wieder zum Hochfahren benötigen. Die hierfür verwendeten Informationen sind in einem Echtzeitsystem nicht vorhanden und können nur ermittelt werden, wenn der Lastverlauf bekannt ist. Ein Echtzeitsystem kann durch die Unwissenheit über die Zukunft daher praktisch nie die optimale Downtime/Energieeinsparung erreichen.

Abbildung 5.5 stellt die Zusammenhänge noch einmal anschaulich für das gegebene Szenario dar. Angenommen wird dabei, dass die Maschine IB7 über die gesamte Dauer des Traces angeschaltet und IB8 zu Beginn des Traces abgeschaltet ist und zum Ende des Traces wieder abgeschaltet wird. Der schwarze Pfeil umfasst die gesamte Dauer des Traces (30 Min). Davon abgezogen wird die Zeit, in der beide Maschinen angeschaltet sein müssen (roter Pfeil). Ebenso abgezogen werden müssen die Dauer für das Starten (blauer Pfeil) und das Herunterfahren (lila Pfeil) der IB8. Übrig bleibt, als grüner Pfeil dargestellt, die optimale Downtime für das Szenario. Man kann erkennen, dass die Zeit für das Starten einer Maschine noch Potenzial für Einsparungen besitzt. Hardware-Hersteller sollten daher ihre Serversysteme mit Suspend-to-RAM (Abschnitt 2.1.1, Seite 9) ausstatten, um dieses Potenzial ausschöpfen zu können. Bis heute ist es in Standard-Serversystemen

nicht möglich, das sonst in Desktop-PCs und Laptops weit verbreitete STR zu nutzen. Ein großer Serverhersteller begründete dies in einem Gespräch mit dessen Support damit, dass Serversysteme nicht dafür gedacht seien, abgeschaltet zu werden und diese Funktionalität deshalb nicht benötigt werde.

Für das gegebene Szenario kann die optimale Downtime durch die vereinfachte Formel 5.3 berechnet werden.

$$T_{opt\_down} = T_{Dauer} - \left(\left(T_{letztes} - T_{erstes} \right) + T_{An} + T_{Aus} \right) \tag{5.3}$$

Dabei ist T_{Dauer} die Dauer des gesamten Traces, $T_{letztes}$ der Zeitpunkt, an dem zum ersten Mal beide Maschinen benötigt werden, T_{erstes} der Zeitpunkt, nachdem nur noch eine Maschine benötigt wird, T_{An} die Zeit, die es benötigt, um eine Maschine anzuschalten und T_{Aus} die Zeit, die es benötigt, eine Maschine auszuschalten.

$$T_{opt\_down} = 1800s - \left(\left(1226s - 447s \right) + 120s + 5s \right) = 896s \tag{5.4}$$

Die optimale Downtime für das verwendete Szenario beträgt somit 896 Sekunden (14,93 Min). Die in den folgenden Messungen erzielten Ergebnisse müssen mit diesem Optimum verglichen werden.

Unterschied zur Publikation In [146] wurde ein anderer Wert als optimale Downtime publiziert. Hier wurde T_{Aus} mit 60 Sekunden angeben, da *CHERUB* durch seine Konfiguration eine Minute warten muss, bevor die Maschine abgeschaltet wird. Da ein optimaler, Orakel getriebener Algorithmus diese Zeit nicht benötigen würde, wurde nachträglich entschieden, die Zeit auf 5 Sekunden (die Zeit zum Herunterfahren) zu reduzieren. Dadurch ändern sich ebenfalls die Ergebnisse der Messungen bezüglich der Metrik *Abweichung vom Optimum* (Abschnitt 5.6.4, Seite 76) leicht zu Ungunsten von *CHERUB*. Die erzielten Ergebnisse der Publikation sind trotzdem uneingeschränkt gültig. ■

Unterschied von optimaler Downtime zu optimaler Energieeinsparung Obwohl mit erhöhter Downtime eine erhöhte Energieeinsparung einhergeht, ist es nicht trivial, die genaue eingesparte Energie zu ermitteln. Die genaue eingesparte Energie hängt vom Verbrauchsprofil der einzelnen Maschinen ab. Wie in verschiedenen Arbeiten gezeigt (z.B. [20], [147]), verbrauchen Maschinen, abhängig von ihrer Ausstattung und der aktuellen Auslastung, verschieden viel Energie. Um die genaue optimale Energieeinsparung zu ermitteln, wäre es notwendig, die exakte Auslastung der Maschinen zu jedem Zeitpunkt und das exakte Verbrauchsprofil der Maschinen zu kennen. Da der Informationsgewinn zwischen optimaler Downtime und optimaler Energieeinsparung bei der Größe des verwendeten Szenarios nicht sehr groß ist, wird an dieser Stelle nur die optimale Downtime verwendet.

In den Simulationen, die später in Kapitel 9 (ab Seite 134) durchgeführt werden, ist dies anders. Im dort verwendeten Simulator (*ClusterSim*) ist sowohl das Verbrauchsprofil als auch die momentane Auslastung bekannt. Dadurch kann exakt simuliert werden, wie hoch der Verbrauch ist. Mehr dazu später in Abschnitt 8.6 (Seite 124).

5.6 Evaluationsmetriken

Um die getesteten Verfahren zu vergleichen, werden vier Metriken verwendet. Die ersten drei sind bekannte Metriken aus dem Webserver-Umfeld und die letzte (*Abweichung vom Optimum*) ist selbst definiert.

5.6.1 Service Level Agreement / Quality of Service

Im Leistungsvertrag (engl. *Service Level Agreement, SLA*) wird unter anderem vereinbart, wie die Qualität des Dienstes (engl. *Quality of Service, QoS*) eines Dienstleisters auszusehen hat. Eine *SLA*-Verletzungen liegt vor, wenn dieser Vertrag nicht eingehalten wird. *SLA* und *QoS* werden häufig als Synonyme verwendet, so auch in dieser Arbeit. Eine Verletzung des *SLA* liegt in den folgenden Experimenten dann vor, wenn die Antwortzeit einer Anfrage die Länge von 5 Sekunden überschreitet und damit als Timeout gewertet wird. Diese Zeitspanne ist eine Schätzung, bei der angenommen wird, dass ein Nutzer nicht länger auf eine Anfrage warten würde, bevor er eine Webseite wieder verlässt oder eine neue Anfrage startet. Wenn alle Rechner angeschaltet sind, sollte das *SLA* (bzw. die *QoS*) bei dem gegebenen Szenario bei mehr als 99 % liegen. *SLA* bzw. *QoS* sollte immer möglichst hoch sein.

5.6.2 First Response Time

Als weitere Metrik wird die Antwortzeit (engl. *First Response Time, FRT*) verwendet. Dabei handelt es sich um die Dauer vom Absenden einer Anfrage beim Client bis hin zum Erreichen des ersten Bits der Antwort eines angefragten Servers. Angegeben als *FRT* ist bei allen Messungen der Median aller Antworten eines Traces. Alle Messungen wurden dreimal wiederholt und der Endwert ergibt sich aus dem Mittelwert dieser drei Mediane. Die *FRT* sollte immer so klein wie möglich sein.

5.6.3 Downtime

Die Downtime (engl. Stillstandszeit) gibt an, wie lange Maschinen abgeschaltet waren. Die erreichte Downtime der einzelnen Verfahren kann mit der in Abschnitt 5.5 ermittelten optimalen Downtime verglichen werden, um zu bewerten, wie gut die Verfahren Energie einsparen. Wenn mehrere Maschinen gleichzeitig abgeschaltet sind, addieren sich deren Zeiten. Wenn also zwei Maschinen für 10 Minuten aus wären, ergäbe sich eine Downtime von 20 Minuten. Obwohl eine hohe Downtime viel Energieeinsparung bedeutet, ist diese nicht immer wünschenswert. Es muss sichergestellt werden, dass trotz hoher Downtime das *SLA* noch ausreichend erfüllt wird. Ansonsten könnten Maschinen abgeschaltet sein, obwohl sie eigentlich benötigt werden, nur um eine hohe Downtime zu erreichen. Dieses Verhalten kann durch die vierte Metrik bewertet werden.

5.6.4 Deviation from Optimum

Um die Experimente möglichst einfach vergleichen zu können, wird eine neue Metrik definiert. Bei der *Abweichung vom Optimum (engl.* Deviation from Optimum, DfO*)* handelt es sich um eine Formel, die sowohl eine schlechte *QoS* als auch eine Abweichung von der optimalen Downtime

bestraft. Sie ist wie folgt definiert:

$$D = \frac{|T_{opt\_down} - T_{value}|}{T_{opt\_down}} * 100 \tag{5.5}$$

$$DfO = \max(D, 100 - SLA) \tag{5.6}$$

Gleichung 5.5 beschreibt die Abweichung von der optimalen Downtime in Prozent. T_{value} ist dabei der im Experiment erreichte Wert. Da, wie bereits beschreiben, die erreichte Downtime auf Kosten der *QoS* höher sein kann als die optimale Downtime, wird die *QoS* ebenfalls mit in die Gleichung 5.6 einbezogen. Dadurch erreicht die *DfO* sowohl bei einer schlechten *QoS* als auch bei einer hohen Abweichung der optimalen Downtime einen schlechten (hohen) Wert.

5.7 Fazit und Zusammenfassung

In diesem Kapitel wurde das für die folgenden Messungen genutzte Szenario beschrieben. Darunter fällt das Wikipedia Webserver-Cluster, bestehend aus 2 Back-End-Rechnern und die verwendete Arbeitslast. Für das Cluster wurde eine Höchstleistung von 50 Anf/s ermittelt. An dieser Leistung orientierend, wurde ein kombinierter 30 minütiger Trace erarbeitet und vorgestellt. Dieser wird in den kommenden Experimenten verwendet, um die gewünschte Lastspitzen-Situation zu erzeugen. Es wurde außerdem gezeigt, dass die Lastspitzen-Situation das schwierigste Szenario für ein Webserver-Cluster ist und andere mögliche Situationen nicht mehr betrachtet werden müssen, da sie entweder trivial sind oder durch die Lastspitzen-Situation mit abgedeckt sind. Es wurde zusätzlich berechnet, dass in dem verwendeten Trace die optimale Downtime bei 14,93 Minuten liegt. Abschießend wurden die vier im nächsten Kapitel verwendeten Metriken vorgestellt und beschrieben. Zusätzlich wurden die initialen Parameter von *CHERUB* vorgestellt.

6 Messungen und Analyse verschiedener Verfahren zur Lastermittlung

Nach dem im letzten Kapitel das Setup und seine Konfiguration vorgestellt wurde, sollen in diesem Kapitel verschiedene Verfahren zur Lastermittlung auf ihre Effizienz hin untersucht werden. Die Lastermittlung ist entscheidend, da auf ihrer Basis die Entscheidungen über das An- oder Abschalten von Maschinen getroffen werden. Um eine sinnvolle Einschätzung über den Nutzen der einzelnen Verfahren vornehmen zu können, wird zunächst eine Referenzmessung durchgeführt, in der *CHERUB* nicht aktiv ist. Anschließend werden die einzelnen Verfahren vorgestellt, gemessen, bewertet und gegebenenfalls erweitert. Bei der Bewertung werden die erzielten Ergebnisse nicht nur mit den Referenzmessungen verglichen, sondern auch mit dem in Abschnitt 5.5 vorgestellte mögliche Optimum, was erreicht werden könnte.

6.1 Referenzmessung

Um die Vor- und Nachteile der späteren Verfahren bewerten und diese vergleichen zu können, müssen zuerst Referenzmessungen durchgeführt werden. Bei diesen Messungen ist *CHERUB* nicht involviert, demzufolge sind beide Back-End-Maschinen während der gesamten Messung angeschaltet und ihre volle Kapazität kann genutzt werden. In Abschnitt 5.2 wurde bereits das Verhalten der beiden Scheduling-Verfahren RR und LC untersucht. Dies soll im Rahmen der Referenzmessung noch einmal mit dem vorhandenen Trace mitgemessen werden.
Es wird außerdem die Situation gemessen, in der nur ein einzelnes Back-End angeschaltet ist. Dieses Messergebnis stellt praktisch die untere Schranke für die *QoS* dar und soll ein Gefühl dafür vermitteln, wie stark das Setup überlastet würde, wenn es nicht richtig verwaltet wird.
Es wurden je Scheduling-Verfahren drei Messungen durchgeführt und Mittelwerte gebildet. Tabelle 6.1 zeigt die Ergebnisse der Messungen. RR schneidet im Gegensatz zu den in Abschnitt 5.2 gemachten Messungen am besten ab. Der Grund dafür könnte der realistische Trace sein der das Setup nicht ununterbrochen an seine Grenzen bringen soll. LC schneidet mit einem Abstand von

Metrik	Methode	Arithmetisches Mittel	Standardabweichung
QoS	Round-Robin	99,89 %	0,03 %
	Least-Connection	99,63 %	0,21 %
	einzelne Maschine	23,75 %	0,64 %
FRT	Round-Robin	9,5 ms	1,5 ms
	Least-Connection	15,1 ms	5,1 ms
	einzelne Maschine	6055,0 ms	108,0 ms

Tabelle 6.1: *QoS* und *FRT* im Referenz-Szenario ohne *CHERUB*.

Metrik	Methode	Arithmetisches Mittel	Standardabweichung
QoS	Round-Robin	86,66 % (99,89 %)	2,84 % (0,03 %)
	Least-Connection	94,96 % (99,63 %)	4,01 % (0,21 %)
FRT	Round-Robin	137 msec (9,5 msec)	7 msec (1,5 mesc)
	Least-Connection	107 msec (15,1 msec)	38 msec (5,1 msec)
Downtime	Round-Robin	19,75 min (0 min)	3,63 min (0 min)
	Least-Connection	19,95 min (0 min)	4,77 min (0 min)
DfO	Round-Robin	35,38 %	15,98 %
	Least-Connection	37,29 %	20,90 %

Tabelle 6.2: Messergebnisse bei der Nutzung von `ipvsadm`. In Klammern sind die Referenzwerte angegeben.

nur 0,26 % ebenfalls sehr gut ab. Da nur für einen kurzen Moment eine Hochlastsituation besteht (siehe Abbildung 5.3, um Sekunde 800), kann LC nicht von seinem zusätzlichem Wissen profitieren. Ähnliche Beobachtungen wurden auch in [142] gemacht. Erwartungsgemäß liefert nur die Variante mit der einzelnen Maschine extrem schlechte Werte. Diese Messung zeigt, dass die untere Schranke für die *QoS* bei 23,75 % liegt.

6.2 Schwellwertbasierter Ansatz

Zu Beginn der Messungen werden klassische, schwellwertbasierte Verfahren getestet und verbessert. Solange nicht explizit erwähnt, gelten für alle Messungen die Parameter wie sie in Kapitel 5 vorgestellt wurden (Apache siehe Tabelle 5.3, Seite 65, *CHERUB* siehe Tabelle 5.4, Seite 68).

6.2.1 Nutzung des aktuellen Wertes - via IPVSADM

Hierbei handelt es sich um die erste Messung, in der *CHERUB* aktiv ist. Zur Lastermittlung wurde das Administrationstool vom LVS verwendet (`ipvsadm`, Abschnitt 2.2.1.5, Seite 20). LVS, als zentrale Lastverteilungskomponente, liefert alle Informationen, die für eine erfolgreiche Verwaltung des Clusters benötigt werden. Als Last wurde hier die Anzahl der aktiven Verbindungen verwendet, welche in Abbildung 2.5 (Seite 20) in Zeile 6 und 7 mit 0 und 552 angegeben sind.
Tabelle 6.2 zeigt die Ergebnisse der Messungen und in Klammern den direkten Vergleich mit den Werten der Referenzmessung.
Die Werte der Messungen liegen in einem inakzeptablen Bereich. Es treten vor allem bei der Nutzung von RR signifikant mehr *SLA*-Verletzungen auf (8,3 % mehr) als bei LC. Aber auch LC liefert mit 94,96 % kein überzeugendes Ergebnis und ist 4,67 % schlechter als ohne *CHERUB*. Auch die durchschnittliche *FRT* liegt wesentlich höher als der Referenzwert (127,5 ms bzw. 91,9 ms höher). Die Downtime hingegen ist mit einer durchschnittlichen Zeit von fast 20 Minuten sehr hoch und 5 Minuten höher als das Optimum, welches bei 14,93 Minuten lag (siehe Abschnitt 5.5, Seite 72). Die hohen Standardabweichungen zeigen die Anfälligkeit der *ipvsadm*-Variante gegenüber Schwankungen.
Die Ursache für die schlechten Ergebnisse ist, dass die von *ipvsadm* gelieferten Werte Momentaufnahmen sind und diese nicht der Anzahl der Anfragen der letzten Sekunde entsprechen. Daher

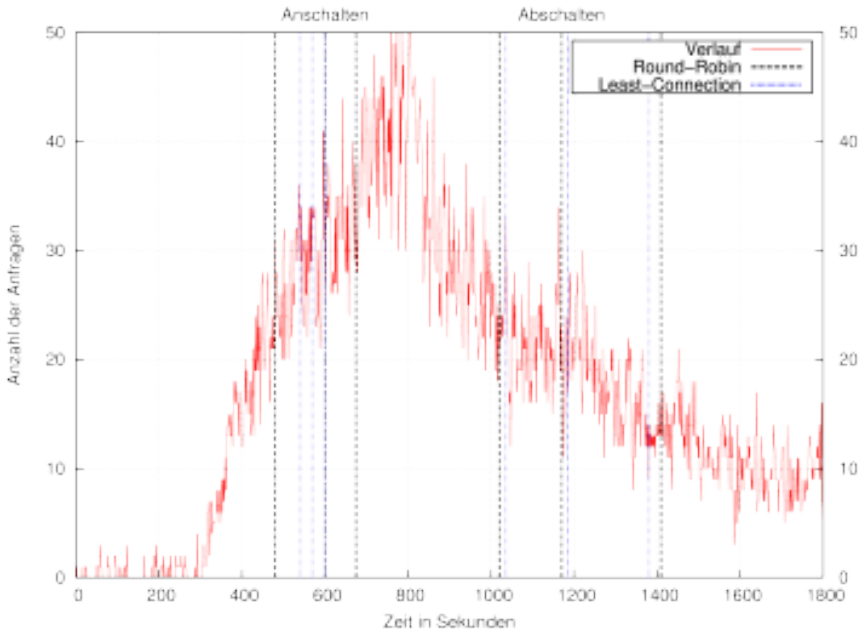


Abbildung 6.1: An- und Abschalt-Zeitpunkte unter Verwendung von `ipvsadm`.

ist der Wert nicht nur niedriger als der eingestellte Überlastparameter, er ist auch noch anfälliger gegenüber Schwankungen, da Anfragen in den meisten Fällen nur Millisekunden zur Abarbeitung benötigen (vergleiche Abschnitt 5.4.3) und deshalb schnell wieder aus der Ausgabe von `ipvsadm` verschwinden können. Da `servload/0.5` kein HTTP/1.1 unterstützt, werden alle Anfragen wie HTTP/1.0 behandelt, daher gibt es keine persistenten Verbindungen und jede Anfrage entspricht genau einer Verbindung. Die aktiven und inaktiven Verbindungen zu zählen und über eine Differenz den gesuchten Wert zu ermitteln ist ebenfalls nicht möglich, da inaktive Verbindungen auch wieder gelöscht werden und es sich nicht um einen stetig steigenden Wert handelt. Die Inkonsistenz zwischen Last-Parameter und verwendetem Wert schlägt sich in den gemessenen Werten nieder. Die hohe *DfO* spiegelt ebenfalls gut wider, dass das verwendete Verfahren in dieser Form ungeeignet ist.

Abbildung 6.1 zeigt die Zeitpunkte, an denen *CHERUB* einen Befehl zum Starten bzw. zum Herunterfahren einer Maschine erteilt hat (senkrechte Markierungen). Sie macht noch einmal deutlich, dass der Knoten zu spät angeschaltet bzw. zu früh wieder abgeschaltet wird.

Zwischenfazit Da die Schwellwerte auf *Anf* (innerhalb einer Sekunde) geeicht sind, `ipvsadm` aber eine Momentaufnahmen liefert, sind die Werte nur bedingt brauchbar. Die Parameter anzupassen könnte bessere Ergebnisse erzielen, wäre aber immer noch sehr anfällig für Schwankungen und damit trotzdem nicht geeignet.

```
[root@ib1 ~]# ipvsadm -l --rate
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port               CPS    InPPS    OutPPS    InBPS    OutBPS
  -> RemoteAddress:Port
TCP ib1.leibniz.cluster:http           6       87        48     5417     31837
  -> ib7.ib.cluster:http               3       39        21     2479     10137
  -> ib8.ib.cluster:http               3       48        28     2938     21700
```

Abbildung 6.2: `ipvsadm` mit der Option -l --rate (-l listet alle Server auf und ist gleichzeitig der Standard-Parameter, falls keine Parameter angegeben werden).

```
Timestamp      Load:ib7    Load:ib8
1383836588        4:ib7       5:ib8
1383836589        6:ib7       4:ib8
1383836590        2:ib7       2:ib8
1383836591        9:ib7       5:ib8
```

Abbildung 6.3: Format des verwendeten Logfiles, welches mit Hilfe von `tcpdump` erzeugt wurde.

6.2.2 Nutzung des aktuellen Wertes - via Protokoll

Die in Abschnitt 6.2.1 beschriebenen Probleme mit `ipvsadm` lassen sich umgehen. `ipvsadm` bietet die Option `-l --rate` (siehe Abbildung 6.2) an, diese liefert einen Wert, der die Verbindungen pro Sekunde angibt. Dieser Wert ist allerdings mit Vorsicht zu verwenden, da er, dem Code zufolge (`Linux/net/ipv4/ipvs/ip_vs_est.c`), den Durchschnittswert der letzten 8 Sekunden bildet. Eine andere Möglichkeit ist, eine Protokoll-Datei (engl. Logfile) zur Hilfe zu nehmen. Bei dem Logfile handelt es sich um die aufwändigere Methode, da dieses erst erzeugt werden muss. Der große Vorteil ist allerdings, dass es eine RMS-unabhängige Lösung ist. Bei einem Wechsel des Dienstes müsste keine Anpassung bezüglich der Lastermittlung an *CHERUB* erfolgen. Um ein Logfile wie in Abbildung 6.3 zu erzeugen, wurde ein einfaches Shell-Skript und das Tool `tcpdump` (Abschnitt 2.2.4, Seite 30) verwendet. Mit Hilfe von `tcpdump` wird jedes eingehende Paket darauf überprüft, ob es sich um eine GET-Request handelt. Der GET-Request ist die initiale Anfrage an den Server und beinhaltet das namensgebende HTTP-Schlüsselwort `GET`. Wird ein `GET` gefunden, wird die Zeit und dessen Ziel (IB7 oder IB8) protokolliert. Mit diesen beiden Informationen kann die Last der letzten Sekunde von allen beteiligten Back-Ends RMS unabhängig ermittelt werden.

Metrik	Methode	Arithmetisches Mittel	Standardabweichung
QoS	Round-Robin	98,38 % (86,66 %)	1,43 % (2,84 %)
	Least-Connection	98,93 % (94,96 %)	0,46 % (4,01 %)
FRT	Round-Robin	16,3 ms (137 ms)	2,0 ms (7 ms)
	Least-Connection	23,6 ms (107 ms)	3,6 ms (38 ms)
Downtime	Round-Robin	8,26 min (19,75 min)	0,26 min (3,63 min)
	Least-Connection	9,34 min (19,95 min)	1,80 min (4,77 min)
DfO	Round-Robin	44,72 % (35,38 %)	1,41 % (15,98 %)
	Least-Connection	37,43 % (37,29 %)	9,82 % (20,90 %)

Tabelle 6.3: Ergebnisse bei der Verwendung eines Logfiles.

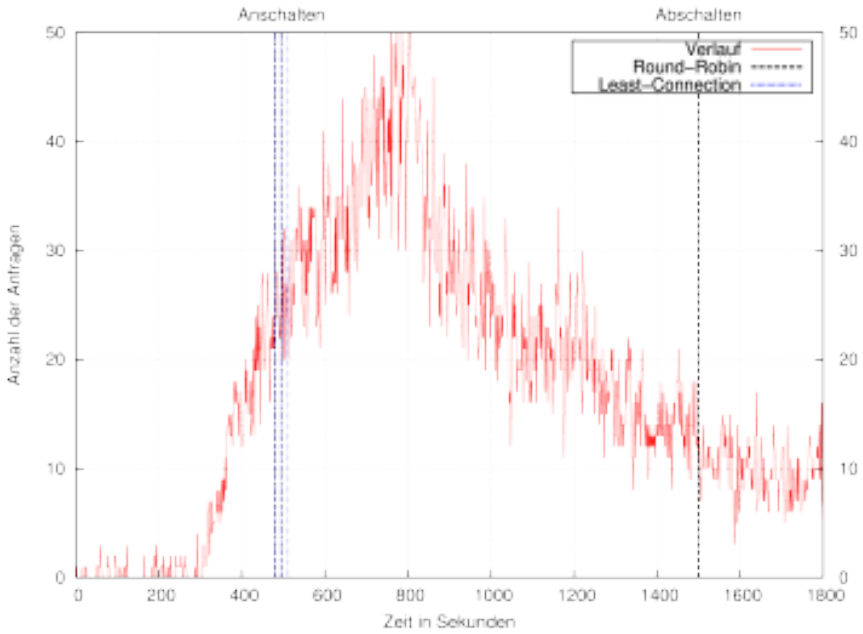


Abbildung 6.4: An- und Abschalt-Zeitpunkte unter Verwendung der Last der letzten Sekunde mit Hilfe eines Logfiles.

Tabelle 6.3 zeigt die Ergebnisse der Messungen und in Klammern die Ergebnisse bei Verwendung von `ipvsadm` zum direkten Vergleich. Wie in Tabelle 6.3 zu sehen, sind die mit der Nutzung des Logsfiles verbundenen Werte bereits besser. Insbesondere das *SLA* ist nur noch um ~1 % schlechter als in der Referenzmessung. Die *FRT* ist nur noch 6,8 ms bzw. 8,5 ms langsamer als bei der Referenz. Für einen Anwender ist diese Verzögerung praktisch nicht zu bemerken. Die gesparte Zeit, und damit die gesparte Energie, hat sich erwartungsgemäß halbiert. Die Abweichung vom Optimum ist allerdings immer noch sehr hoch. Dies liegt an der nun viel kürzeren Downtime, welche sich direkt auf die *DfO* auswirkt. Die Ergebnisse sind aber, vor allem wegen der hohen *SLA*, der niedrigen *FRT* und der viel geringeren Standardabweichung wesentlich besser geworden.

Weshalb die Downtime so weit vom Optimum entfernt ist, kann man in Abbildung 6.4 sehr gut nachvollziehen. Während die Befehle zum Starten der Rechner zuverlässig und beinahe zum gleichen Zeitpunkt gegeben wurden, wurde nur in einem der sechs Fälle eine Maschine wieder abgeschaltet. Das weist auf einen schlecht eingestellten Unterlast-Schwellwert hin. Auf Grund der in Abschnitt 4.2 (Seite 59) beschriebenen Gefahr des *State Flappings*, muss dieser allerdings mit Bedacht angepasst werden.

Zwischenfazit Bei der Verwendung der richtigen Werte kann auch ein einfaches Verfahren schon gute Einsparungen erzielen, es wird aber noch viel Potenzial verschwendet. Ein Problem bleibt die richtige Einstellung der Schwellwerte.

6.2.3 Nutzung des aktuellen Wertes in Kombination mit einem dynamischen Schwellwert

Um das im vorherigen Abschnitt identifizierte Problem eines ungünstig gewählten Schwellwertes möglichst nachhaltig zu lösen, wird dieser nicht einfach nur angepasst. Es wird stattdessen ein dynamischer Schwellwert eingeführt. Dieser soll abhängig von der Anzahl der aktuell aktiven Maschinen sein und verhindern, dass eine ungeeignete Konfiguration *State Flapping* begünstigt. Gleichzeitig hilft er dem Administrator des Systems, da dieser einen wichtigen Parameter weniger zu konfigurieren hat.
Der dynamische Schwellwert wird wie folgt definiert:

$$Unterlast = \frac{\ddot{U}berlast * (\#Maschinen - \texttt{sequential\_shutdown})}{\#Maschinen} \tag{6.1}$$

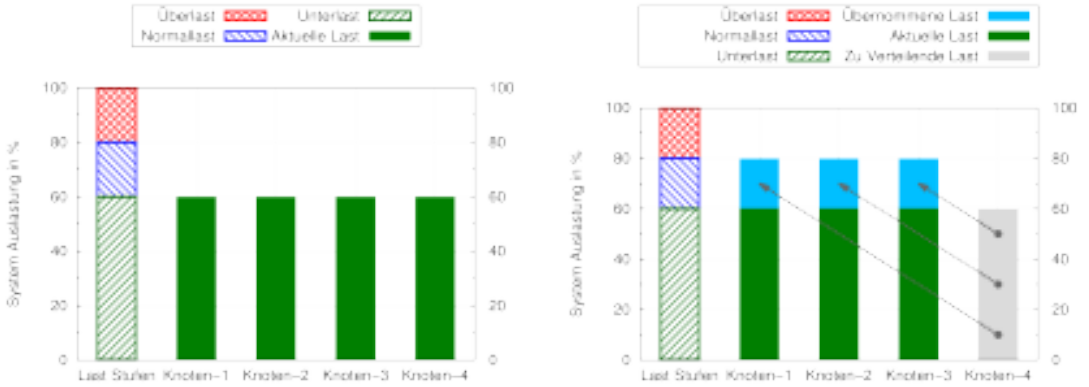
$$Unterlast = min(Unterlast, \ddot{U}berlast * 0,8) \tag{6.2}$$

Bei #*Maschinen* handelt es sich um die Anzahl aller Rechner im Zustand *BUSY* und *ONLINE*. Diese sind in der Lage, ohne Verzögerung Last zu bearbeiten, da sie am RMS angemeldet sind. Bei `sequential_shutdown` handelt es sich um die Anzahl der Rechner, die laut Konfiguration maximal gleichzeitig abgeschaltet werden dürfen. $Unterlast$ bzw. $\ddot{U}berlast$ bezeichnen entsprechend den Unterlast- und Überlast-Schwellwerte. Für Formel 6.1 gilt die Bedingung, dass $\#Maschinen > 0$ sein müssen. Andernfalls sind bereits alle aktiven Rechner abgeschaltet, was in jedem Fall verhindert werden muss. Diese Bedingung kann bei Verwendung der Formel und ohne Fremdeinwirkung allerdings nicht verletzt werden. Vor dem Abschalten der letzten Maschinen würde der Term $\ddot{U}berlast * (\#Maschinen - \texttt{sequential\_shutdown})$ und damit der dynamische Schwellwert zu <= 0 evaluiert werden. Da die ermittelte Last nie unter 0 fallen kann, würde *CHERUB* keine Maschine mehr abschalten und es bleibt mindestens eine Maschine angeschaltet. Formel 6.2 erweitert den Schwellwert und führt eine obere Schranke ein. Diese ist für größere Setups wichtig. Wenn #*Maschinen* größer wird und `sequential_shutdown` klein bleibt, würde der Unterschwellwert beinahe identisch dem Überlast-Schwellwert werden. Dieses Verhalten würde *State Flapping* stark begünstigen und muss in jeden Fall verhindert werden. Aus diesem Grund kann der Unterlast-Schwellwert nie größer werden als 80 % des Überlast-Schwellwertes. Für die in diesem Kapitel durchgeführten Messungen ist dieser Umstand allerdings nicht relevant. Die 80 % Marke ist eine Schätzung, die nicht näher untersucht wurde.
Die Verwendung des dynamischen Schwellwertes vereinfacht die Handhabung von *CHERUB* und bringt mehrere Vorteile mit sich:

1. Es muss nur noch ein einzelner Schwellwert vom Administrator eingestellt werden.
2. Es muss kein Knoten explizit als *immer an* konfiguriert werden. Die Formel verhindert, wie beschrieben, dass der letzte aktive Knoten abgeschaltet werden kann.
3. Der Schwellwert stellt sicher, dass Knoten erst dann abgeschaltet werden, wenn alle anderen Knoten gemeinsam die Last der abzuschaltenden Maschinen tragen können. Dadurch wird der Überlast-Schwellwert nicht sofort wieder überschritten und *State Flapping* verhindert.
4. Für die folgenden Messungen ändern sich der *CHERUB*-Parameter für den Unterlast-Schwellwert, wie angegeben in Tabelle 6.4 auf 7,5 Anf. Dadurch können die Knoten früher abge-

Parameter	neuer Wert	alter Wert
Unterlast-Schwellwert	Automatisch (7,5 Anf)	5 Anf

Tabelle 6.4: Veränderte CHERUB-Parameter.



(a) Situation VOR dem Abschalten von Knoten-4. **(b)** Situation NACH dem Abschalten von Knoten-4.

Abbildung 6.5: Grundidee des dynamischen Schwellwertes.

schaltet werden und die Ergebnisse sollten sich vor allem in Bezug auf die Downtime und die *DfO* verbessern.

Unterschied zur Publikation In [146] wurde der dynamische Unterlast-Schwellwert mit $Unterlast_D = Überlasst - \frac{Überlast}{\#Maschinen}$ definiert. Diese Definition berücksichtigt nicht die Möglichkeit, dass in größeren Setups mehrere Maschinen gleichzeitig abgeschaltet werden könnten. Zusätzlich beinhaltete sie nicht die 80 % Schranke, welche ebenfalls für größere Setups benötigt wird. ■

Beispiel 10 zeigt die Veränderung des aktuellen Setups und Abbildung 6.5 illustriert die Situation für vier Maschinen.

Beispiel 10 Es werden zwei Rechner und eine Konfiguration mit Überlast = 15 und `sequential_shutdown` = 1 angenommen. Dann würde, wenn beide Maschinen angeschaltet sind, ein Unterlast-Schwellwert von $Unterlast = \frac{15*(2-1)}{2} = 7,5$ gelten. Dadurch würde die zweite Maschine beim Abschalten einer Maschine noch genügend freie Kapazität für die Last der abgeschalteten Maschine besitzen. Wenn nur eine der Maschinen an ist, ergibt sich $Unterlast = \frac{15*(1-1)}{1} = 0$. Auch bei 0 Anf würde die letzte Maschine von *CHERUB* als *BUSY* eingestuft werden und damit angeschaltet bleiben.

□

Da zwischen RR und LC ein eher kleiner Unterschied besteht und LC in den letzten Messungen etwas besser abgeschnitten hat, wurde in den kommenden Messungen darauf verzichtet, die RR-Variante ebenfalls zu messen. Bei Interesse bietet [143] einen Vergleich zwischen RR und LC.

Metrik	Methode	Arithmetisches Mittel	Standardabweichung
QoS	Least-Connection	98,82 % (98,93 %)	0,52 % (0,46 %)
FRT	Least-Connection	34,3 ms (23,6 ms)	11,9 ms (3,6 ms)
Downtime	Least-Connection	9,63 min (9,34 min)	1,4 min (1,8 min)
DfO	Least-Connection	35,45 % (37,43 %)	7,63 % (9,82 %)

Tabelle 6.5: Messergebnisse bei der Nutzung eines Logfiles und dynamischem Schwellwert. In Klammern sind zum besseren Vergleich die Ergebnisse der letzten Messung (Tabelle 6.3) vermerkt.

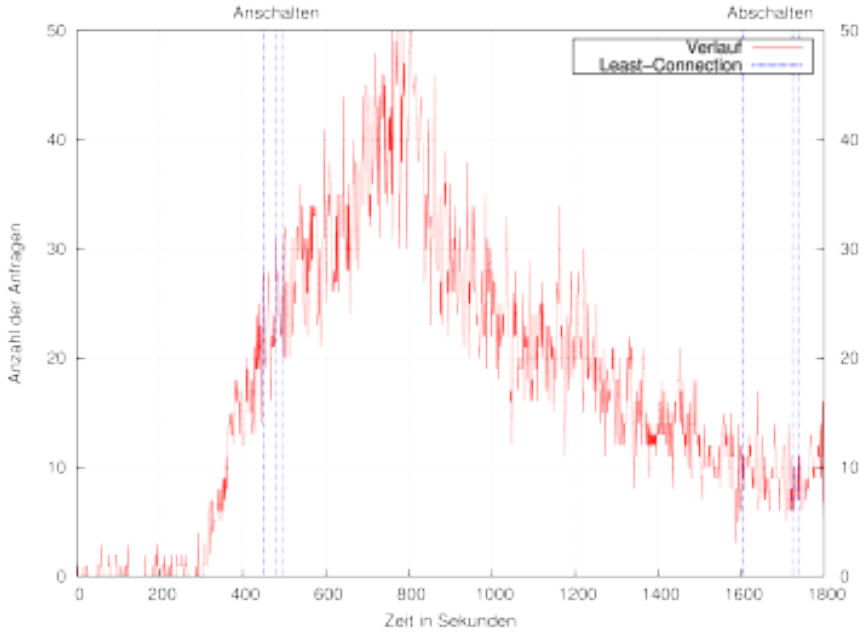


Abbildung 6.6: An- und Abschalt-Zeitpunkte unter Verwendung der Last der letzten Sekunde und unter Verwendung eines dynamischen Schwellwertes.

Wie in Tabelle 6.5 zu sehen, haben sich die Werte kaum verändert. Die eingesparte Zeit sowie die Abweichung vom Optimum haben sich leicht verbessert. Die *FRT* und die *QoS* haben sich hingegen leicht verschlechtert. Wie man in Abbildung 6.6 deutlich erkennt, werden die Rechner nun, im Gegensatz zur vorherigen Messung, bei jedem Durchlauf der Tests wieder abgeschaltet. Daraus resultiert die etwas höhere Downtime. Insgesamt bleibt aber weiterhin das Problem bestehen, dass die Maschinen zu spät abgeschaltet werden. Man könnte für das gegebene Szenario die Parameter zwar weiter optimieren und damit vermutlich zu einem guten Messergebnis kommen, das Grundproblem wäre damit aber nicht behoben und bei einem andersartigen Verlauf könnten erneut Probleme entstehen.

Zwischenfazit Eine Vereinfachung des Verfahrens durch die Einführung eines dynamischen Schwellwertes hat zwar diverse Vorteile gebracht, es wird aber weiterhin Potenzial verschwendet.

Dieses Potenzial kann nur situationsabhängig, durch Optimierung von Parametern, ausgeschöpft werden. Für eine allgemeinere Lösung, die besser funktioniert und das verschwendete Potenzial ausschöpft, muss ein intelligenteres Verfahren angewendet werden. Durch die Einfachheit von Schwellwerten ist ein starker Lastanstieg sehr problematisch. Damit im gegebenem Szenario das *SLA* nicht zu stark sinkt, müsste der Überlast-Schwellwert viel kleiner konfiguriert werden, als intuitiv nötig (15 Anf, obwohl die Maschine eine Last von 25 Anf/s bewältigt). Wenn im gegebenen Setup eine konstante Last von 16-20 Anf/s herrschen würde, wäre permanent eine zweite Maschine angeschaltet, obwohl dies aus Sicht der Performance nicht notwendig wäre. Wenn der Schwellwert auf einen realistischen Wert, z.B. >= 20 Anf, gestellt würde, würde die zweite Maschine allerdings zu spät gestartet und ein großer Teil der Last könnte nicht rechtzeitig bearbeitet werden. Für den allgemeinen Einsatz sind die Möglichkeiten eines schwellwertbasierten Verfahrens erschöpft.

6.3 Ansatz mit Hilfe von Lastvorhersage

Um die Probleme von schwellwertbasierten Verfahren zu lösen oder abzuschwächen, soll nun eine einfache Methode des maschinellen Lernens bzw. der Statistik auf ihren Nutzen hin untersucht werden, die lineare Regression.

6.3.1 Nutzung von linearer Regression zur Lastvorhersage (2 Minuten Historie)

Da, wie im vorherigen Abschnitt gezeigt, ein einfaches schwellwertbasiertes Verfahren einen Lastanstieg nicht als solchen erkennen und deshalb nicht auf ihn reagieren kann, soll nun eine lineare Regressionsanalyse zur Lasterkennung verwendet werden. Bei der linearen Regression handelt es sich um ein Verfahren, bei dem versucht wird, eine Gerade, möglichst passend, durch eine Menge von Datenpunkten zu legen.
Da es bei Messungen praktisch nie einen genauen linearen Zusammenhang zwischen den Messpunkten x_i, y_i und x_j, y_j (mit $i, j \in \{1..n\}$ und $i \neq j$) gibt, kann eine solche Gerade nur geschätzt werden.

$$\hat{y} = a + bx \tag{6.3}$$

Eine Gerade wird definiert durch ihren Schnittpunkt mit der Y-Achse a und ihrem Anstieg b. Diese stellen einen linearen Zusammenhang zwischen Ein- und Ausgabe der Funktion her, zu sehen in Formel 6.3.
Der Abstand zwischen dem durch die Gerade geschätzten Wert $\hat{y}_i$ und dem tatsächlich gemessenen Wert y_i wird als Störgröße ε_i oder auch als Residuum bezeichnet. Ziel der linearen Regression ist es, die Parameter a und b so zu wählen, dass die Summe der Residuen möglichst klein ist, dargestellt in Formel 6.4.

$$s = \sum_{i=1}^{n} (y_i - \hat{y}_i) \tag{6.4}$$

Dieses Kriterium lässt allerdings mehrere Lösungen zu, weshalb die Vorzeichen berücksichtigt werden müssen. Verwendung findet daher entweder eine *MAD*-Schätzung (engl. minimum absolute deviation), bei der die absoluten Werte der Residuen verwendet werden, oder die *OLS*-Schätzung (engl. ordinary least squares), bei der die Werte quadriert werden. Da die Quadrierung

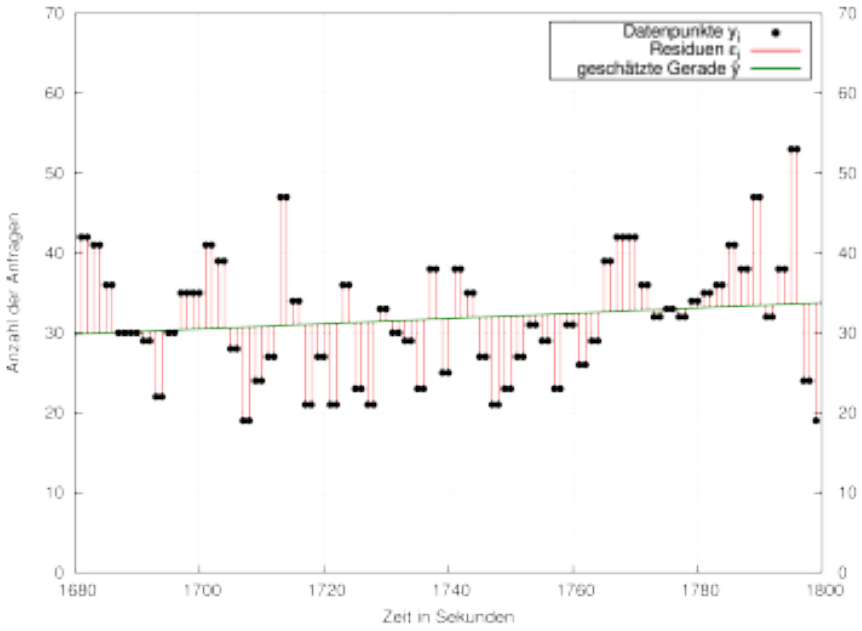


Abbildung 6.7: Beispiel einer linearen Regression.

weitere statistische Vorteile hat, wird im Normalfall auf die *OLS*-Schätzung zurückgegriffen, dargestellt in Formel 6.5.

$$s = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{6.5}$$

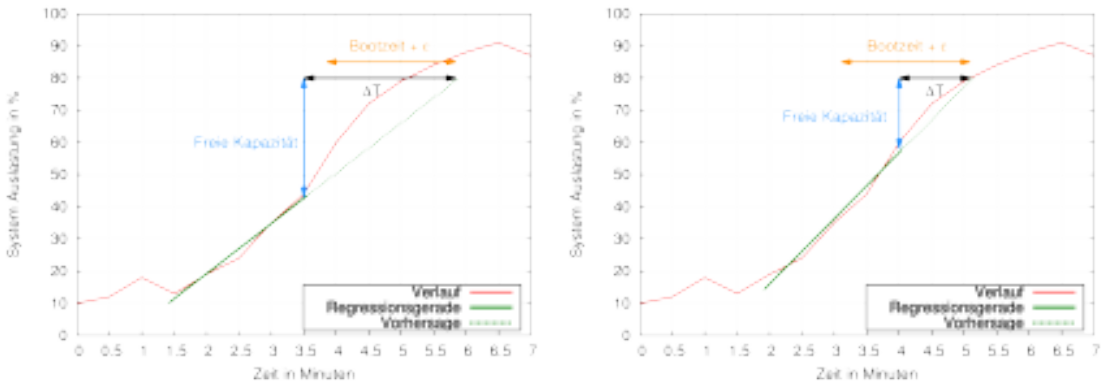
Man kann nun Formel 6.3 in Formel 6.5 einsetzen und erhält Formel 6.6.

$$s = \sum_{i=1}^{n} (y_i - a - bx_i)^2 \tag{6.6}$$

Da Formel 6.6 minimiert werden soll, können die Parameter a und b mit Hilfe von Differenzialrechnung ermittelt werden. Die genaue Herleitung kann sehr gut in Urban und Mayerls *Regressionsanalyse: Theorie, Technik und Anwendung* [148] (Seite 40 ff.) oder in Draper und Smiths *Applied Regression Analysis* [149] (Seite 8 ff) nachgeschlagen werden.
Für *CHERUBs* Verfahren wird nur der Parameter b, die Steigung der Regressionsgeraden, benötigt, welcher sich schlussendlich nach Anwendung der Differenzialrechnung und Umstellung mit Formel 6.7 berechnen lässt.

$$b = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \tag{6.7}$$

Dabei entspricht $\bar{x}$ dem Mittelwert der Werte von x_i und analog dazu entspricht $\bar{y}$ dem Mittelwert der Werte von y_i. Der Anstieg der Last wird von *CHERUB* in jedem Intervall überprüft. Als Datenpunkte dient die Last der letzten 2 Minuten (120 Datenpunkte). Abbildung 6.7 stellt ein Beispiel einer solchen geschätzten Gerade von *CHERUB* dar. Über die Steigung b kann errechnet werden, wie lange es bei gleichbleibendem Anstieg dauert, bis der Überlast-Schwellwert über-



(a) Es bleibt genug Zeit zum Anschalten, nichts muss unternommen werden.

(b) Hier ist die Zeit zum Anschalten bereits sehr knapp, ein Bootbefehl müsste umgehend umgesetzt werden.

Abbildung 6.8: Grundidee der Lastvorhersage.

schritten wird. Da *CHERUB* die Dauer des Bootvorgangs bekannt ist, kann berechnet werden, ob ein weiterer Rechner gestartet werden muss oder ob die bestehende Kapazität genügt. So können zusätzlich benötigte Maschinen rechtzeitig angeschaltet werden, damit sie helfen können, die zusätzliche Last zu bearbeiten. Die Dauer, bis bei gleichbleibendem Anstieg eine Überlastsituation auftritt, errechnet sich wie folgt:

$$\Delta T = \frac{sw - l}{b} \tag{6.8}$$

Wobei sw den Überlast-Schwellwert darstellt und l die aktuelle Last angibt. Spätestens wenn ΔT kleiner als die Bootzeit der Maschine ist, muss ein Befehl zum Starten einer weiteren Maschine gegeben werden, damit die neue Ressource schnell genug einsatzbereit ist. Abbildung 6.8 zeigt dies noch einmal anschaulich. In 6.8a ist die Zeit, die noch bleibt, bis die Überlast eintritt (ΔT, der schwarze Pfeil), immer noch größer als die Zeit, die der Parameter `boot_duration` vorgibt (der orangene Pfeil). Der Parameter `boot_duration` umfasst dabei die Zeit, die ein Rechner zum Hochfahren benötigt, und ein ε, welches mindestens so groß gewählt werden sollte, wie das eingestellte Intervall ist. Dadurch kann die Dauer des Intervalls nicht zu Schwierigkeiten führen. Abbildung 6.8b stellt dieselbe Situation eine halbe Minute später dar. Die freie Kapazität ($sw - l$, blauer Pfeil) ist bereits weniger geworden und der Anstieg in den letzten 2 Minuten (grüne durchgezogene Linie) ist angestiegen. Die Last-Prognose (grüne gestrichelte Linie) ist demzufolge auch steiler und ΔT verkürzt sich so stark, dass die Zeit zum Booten + ε nun kleiner ist als ΔT. In diesem Beispiel müsste nun ein Bootbefehl erfolgen, damit das *SLA* auch bei prognostizierter Last eingehalten wird. Beispiel 11 gibt dazu ein Rechenbeispiel an.

Beispiel 11 Man nehme an, die aktuelle Last liegt bei 10 Anf und der Überlast-Schwellwert ist auf 20 Anf eingestellt. Daraus ergibt sich eine freie Restkapazität von 10 Anf. Ist diese Restkapazität aufgebraucht, tritt eine Überlastsituation auf. Das Booten eines Rechners beträgt 78 Sekunden und das *CHERUB*-Intervall ist auf 15 Sekunden eingestellt. Durch den Parameter `wait_for_boot` = 1 dauert es allerdings noch länger, bis eine Maschine wirklich gestartet wird. Um dem gerecht zu

Parameter	neuer Wert	alter Wert
Überlast-Schwellwert	20 Anf	15 Anf
Unterlast-Schwellwert	Automatisch (10 Anf)	Automatisch (7,5 Anf)
Bootzeit	120 Sekunden	-

Tabelle 6.6: Veränderte CHERUB-Parameter.

Metrik	Methode	Arithmetisches Mittel	Standardabweichung
QoS	Least-Connection	99,4 % (99,63 % / 98,82 %)	0,16 % (0,21 % / 0,52 %)
FRT	Least-Connection	33 ms (15,1 ms / 34,3 ms)	27,2 ms (5,1 ms / 11,9 ms)
Downtime	Least-Connection	12,87 min (0 min / 9,63 min)	0,52 min (0 min / 1,4 min)
DfO	Least-Connection	13,84 % (100 % / 35,45 %)	2,83 % (0 % / 7,63 %)

Tabelle 6.7: Ergebnisse unter Verwendung von Lastvorhersage mit Hilfe einer linearen Regressionsanalyse.

werden, wird die `boot_duration` auf 120 Sekunden gesetzt. Das ε beträgt an dieser Stelle somit 42,4 Sekunden. Bei dieser Konstellation muss ein Rechner angeschaltet werden, wenn der Anstieg der berechneten Geraden 0,083 Anf/s oder größer wäre. Bei einem gleichbleibenden Anstieg von 0,083 Anf/s wäre nach 120 Sekunden die restliche Kapazität aufgebraucht und die *QoS* würde sinken.

□

Wie groß das ε gewählt werden sollte, kann nicht pauschal gesagt werden und hängt davon ab, ob der Administrator eher eine hohe *QoS* halten oder eher Kosten sparen möchte. Es sollte allerdings der Wert des `wait_for_boot`-Parameters bei der Einstellung berücksichtigt werden.
Die Grundlage für die Entscheidung, ob eine Ressource angeschaltet wird, ist nun nicht mehr nur die aktuelle Last der Maschinen. Von nun an wird zusätzlich der Anstieg der Last ermittelt und in die Entscheidung einbezogen. Da mit diesem Verfahren mehr Wissen über die Last vorhanden ist, ist es nicht mehr notwendig, einen größeren Puffer im Überlast-Schwellwert zu berücksichtigen. Für die folgende Messung wird daher der Überlast-Schwellwert von 15 Anf auf 20 Anf angehoben. Dies entspricht 80 % der Kapazität der Knoten. In [15] wurde ein oberer Schwellwert von 60 % der Kapazität als optimale Einstellung ermittelt, wenn nur klassische Schwellwerte verwendet werden und keine Lastvorhersage stattfindet. 60 % (15 Anf) entspricht der Einstellung, die in den vorherigen Messungen genutzt wurde. Durch das Anheben der oberen Schranke erhöht sich auch die untere Schranke auf 10 Anf (entspricht 40 %), die weiterhin dynamisch ermittelt wird.
Die Änderungen der Parameter für diese Messung, sind in Tabelle 6.6 zusammengefasst.
Tabelle 6.7 zeigt die Ergebnisse der Messung und in Klammern die Werte der Referenzmessung (siehe Abschnitt 6.1, Seite 78) sowie die Werte der Messung ohne Vorhersage und Verwendung eines dynamischen Schwellwerts (vorheriger Abschnitt, Seite 83).
Die Werte, die mit der linearen Regressionsanalyse erreicht wurden, sind wesentlich besser als die der vorher verwendeten Methoden. Die *QoS* liegt nur noch 0,23 % unter dem Wert, der ohne *CHERUB* erreicht wurde. Dies entspricht nur 73 Anfragen, die mit *CHERUB* zusätzlich nicht rechtzeitig ausgeliefert wurden. Die *FRT* hat sich im Mittel nicht weiter verbessert. Die drei einzelnen Werte sind 18 ms, 16,6 ms und 64,4 ms. Mit Ausnahme der 3. Messung liegt die *FRT*

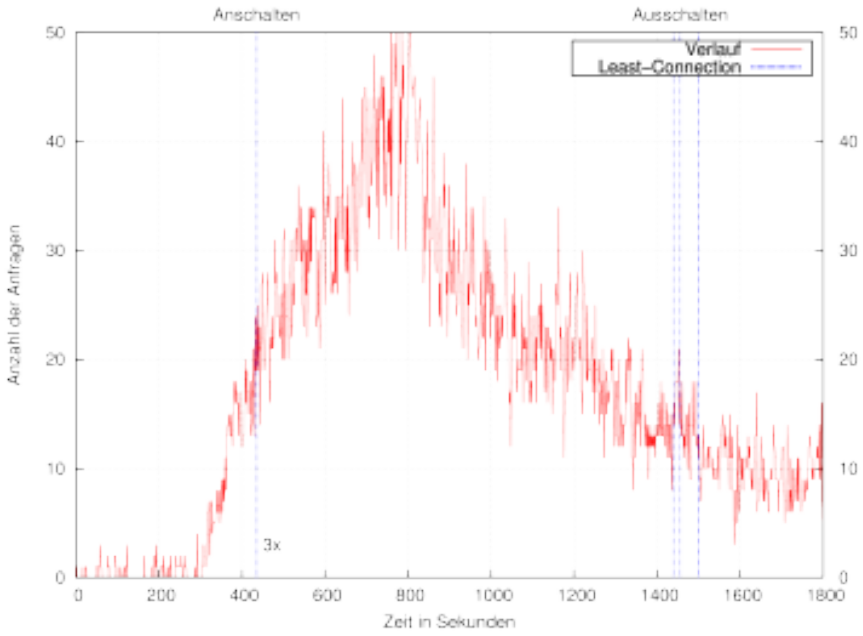


Abbildung 6.9: An- und Abschalt-Zeitpunkte unter Verwendung von linearer Regression mit 2 minütiger Historie.

somit auf einem Niveau mit der Referenzmessung. Allerdings liegen auch die 64,4 ms in einem Bereich, der für einen Nutzer nicht spürbar ist. Obwohl der *QoS* und die *FRT* gestiegenen sind, ist ebenso die Downtime gestiegen. Was kontraintuitiv klingt, zeigt nur, wie viel besser die Variante mit Lastvorhersage gegenüber der bisherigen Methode funktioniert. Die sehr guten Ergebnisse werden durch die geringe *DfO* noch unterstrichen. Die Methoden ohne Vorhersage haben maximal 35,45 % Abweichung erreicht und sind damit 21,6 Prozentpunkte schlechter. Diese Verbesserung auf 13,84 % *DfO* ist erheblich.
Abbildung 6.9 zeigt eindrucksvoll, wie präzise und zuverlässig der Anstieg der Last erkannt wird. In allen 3 Messungen wird der Rechner im selben Zyklus zugeschaltet. Auch beim Abschalten liegen die Zeiten sehr nahe beieinander, was die geringe Varianz der Downtime von 0,52 min (entspricht der Länge von 2 Intervallen) unterstreicht.

6.3.2 Nutzung von linearer Regression zur Lastvorhersage (5 Minuten Historie)

Eine weitere Messung soll feststellen, wie sich eine Vergrößerung der Historie der linearen Regression auf die Messung auswirkt. Die Einstellungen der Länge und damit auch, wie viele Datenpunkte für die Berechnung verwendet werden, ist ein wichtiger Parameter. Ist die Historie zu kurz, können sich schon kleinere lokale Schwankungen stark auf das Verhalten von *CHERUB* auswirken. Ist die Historie hingegen zu lang, wird ein *echter* Lastanstieg womöglich zu spät erkannt und das ganze System droht zu träge zu reagieren. Tabelle 6.8 zeigt die Ergebnisse der Messung und in Klammern die Werte mit der 2 Minuten Historie zum direkten Vergleich. Die Werte aus Tabelle 6.8 bestätigen die Befürchtung hinsichtlich einer zu langen Historie. Die Nutzung der längeren

Metrik	Methode	Arithmetisches Mittel	Standardabweichung
QoS	Least-Connection	97,18 % (99,4 %)	3,129 % (0,161 %)
FRT	Least-Connection	65,9 ms (33 ms)	31,7 ms (27,2 ms)
Downtime	Least-Connection	10,32 min (12,87 min)	1,52 min (0,518 min)
DfO	Least-Connection	30,92 % (13,84 %)	8,31 % (2,83 %)

Tabelle 6.8: Ergebnisse unter Verwendung von Lastvorhersage mit 5 Minuten Historie.

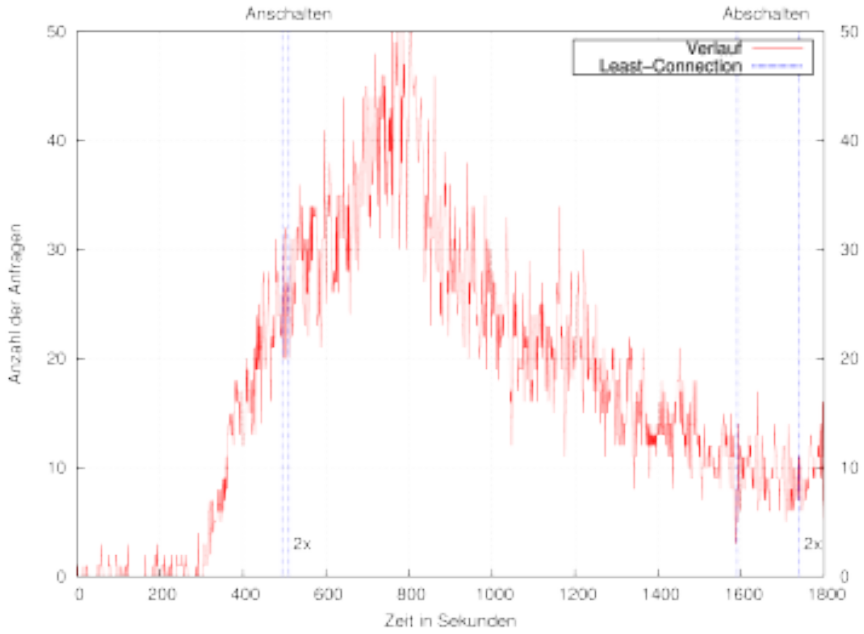


Abbildung 6.10: An- und Abschalt-Zeitpunkte unter Verwendung von linearer Regression mit 5 minütiger Historie.

Historie führt dazu, dass die Werte aller Kategorien schlechter ausfallen. Eine Historie, die etwas länger ist als der Bootvorgang des zu verwaltenden Systems, scheint in dem vorliegenden Fall eine gute Wahl zu sein. Ob zwischen der Bootdauer einer Maschine und der Länge der Historie wirklich ein Zusammenhang besteht, soll hier nicht näher überprüft werden, da mit den vorliegenden Daten keine stichhaltige Aussage darüber getroffen werden kann.
Abbildung 6.10 zeigt noch einmal, dass die Maschinen zwar zuverlässig angeschaltet werden, das Anschalten aber, wie erwartet, zu spät durchgeführt wird. Auch das Abschalten geschieht in dieser Konfiguration wesentlich später als in der vorherigen Konfiguration. Dies könnte daran liegen, dass die leichte Steigung um Sekunde 1200 noch zu lange Einfluss auf die Regressionsgerade hat und die Maschinen deshalb noch vorgehalten werden.

6.3.3 Lastvorhersage mit linearer Regression und mit Verwendung des Mittelwerts

Wie die bisherigen Messungen gezeigt haben, ist die Technik mit linearer Regression am präzisesten. Formel 6.8 (Seite 88) besitzt allerdings noch eine potenzielle Schwäche. Um die freie

Metrik	Methode	Arithmetisches Mittel	Standardabweichung
QoS	Least-Connection	99,79 % (99,4 %)	0,071 % (0,16 %)
FRT	Least-Connection	34,1 ms (33 ms)	25,8 ms (27,2 ms)
Downtime	Least-Connection	10,89 min (12,87 min)	1,52 min (0,52 min)
DfO	Least-Connection	27,04 % (13,84 %)	7,55 % (2,83 %)

Tabelle 6.9: Ergebnisse unter Verwendung von Lastvorhersage und Mittelwerten.

Kapazität zu ermitteln, wird die aktuelle Last des Systems (Last der letzten Sekunde) ermittelt und verwendet. Hier kann sich eine lokale Schwankung negativ auf das Verfahren auswirken, da es sich bei dem ermittelten Wert um einen weniger repräsentativen Wert der aktuellen Last handeln kann. Dieser Effekt ist bekannt und wird als Lastindex, beispielsweise vom Unix Kernel, in Form des *avenrun*-Index ausgenutzt ([150], Seite 138).
Um solche Schwankungen zu unterbinden, wird in der folgenden Messung die aktuelle freie Kapazität nicht mehr mit Hilfe der Last der letzten Sekunden ermittelt, sondern mit Hilfe des Mittelwerts der Last-Werte des letzten Intervalls. Im gegebenen Fall ist dies der Mittelwert aus 15 Werten. Die Historie wird wieder, wie zuvor, auf 2 Minuten gesetzt, da dieser Wert sehr gute Ergebnisse erzielt hat.
Es wurden ebenfalls Tests mit einer Historie von 60 Sekunden durchgeführt. Die Ergebnisse dieser Tests waren allerdings ebenfalls zu schlecht, daher wurde eine kürzere Historie nicht weiter untersucht. Die Erwartung, dass das System mit zu kurzer Historie zu schnell reagiert, wurde dabei bestätigt.
Tabelle 6.9 zeigt die Ergebnisse und den direkten Vergleich ohne die Verwendung des Mittelwerts, sondern mit der Last der letzten Sekunde in Klammern. Die Nutzung des Mittelwerts verlagert den Zeitpunkt des Bootbefehls kaum nach hinten (siehe Abbildung 6.11). Im Gegensatz dazu wird das Abschalten verhältnismäßig stark verzögert. Dies sorgt auch für die schlechteren Werte bezüglich der Downtime und der *DfO*. Die *QoS* ist allerdings sehr gut und nicht mehr von der ohne *CHERUB* zu unterscheiden (99,79 % mit vs. 99,63 % ohne). Die Messschwankungen sorgen hier sogar dafür, dass die *QoS* in den Messungen mit *CHERUB* besser ist als die *QoS* ohne *CHERUB* (LC-Variante). Die *FRT* ändert sich gegenüber der Messung ohne Mittelwerte praktisch nicht. Sie verhält sich wie in der Messung ohne Mittelwerte, es gibt wieder einen Ausreißer nach oben (18,71 ms, 63,89 ms, 19,61 ms). Dieser führt erneut zu einem erhöhten Mittelwert und einer erhöhten Standardabweichung. Auch der Mittelwert ist in einem guten Bereich.

6.4 Fazit und Zusammenfassung der Messungen

Die Messungen lassen sich noch einmal gut mit den Abbildungen 6.12a -6.12d zusammenfassen. Sie zeigen alle wichtigen Metriken im direkten Vergleich. Abgetragen sind jeweils die Messungen, bei denen Least-Connection zum Einsatz kam. Abgetragen sind die Werte in vorgestellter Reihenfolge. Darunter fallen die Messungen ohne *CHERUB* (Referenz), mit Nutzung von `ipvsadm`, mit Nutzung eines Protokolls, mit Nutzung eines Protokolls und Hilfe des dynamischem Schwellwerts (Prot. + dyn.), mit Nutzung der linearen Regressionsanalyse und 120 Sekunden Historie (LR 120 s), der linearen Regressionsanalyse und 300 Sekunden Historie (LR 300) und zuletzt der linearen Regressionsanalyse und 120 Sekunden Historie unter Verwendung von Mittelwerten (LR 120 s ∅).

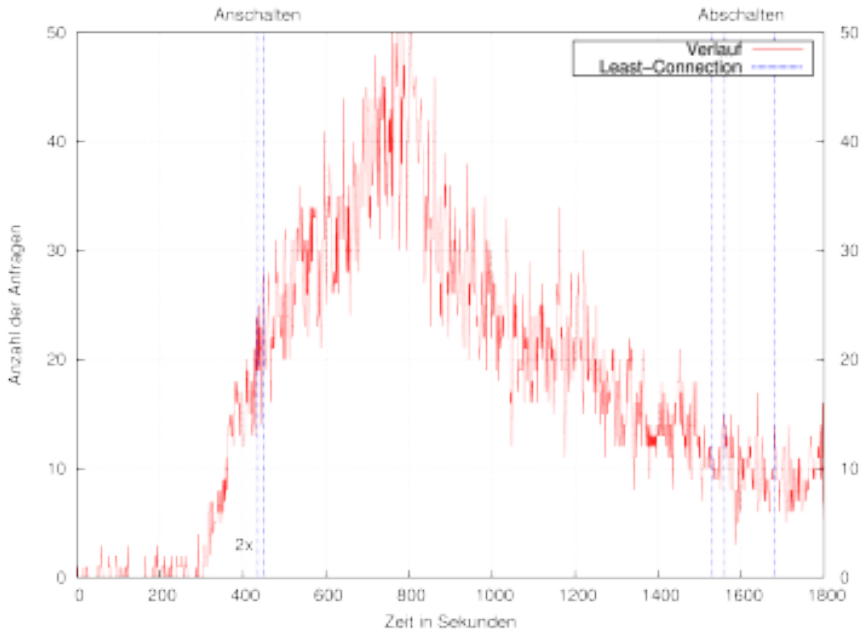


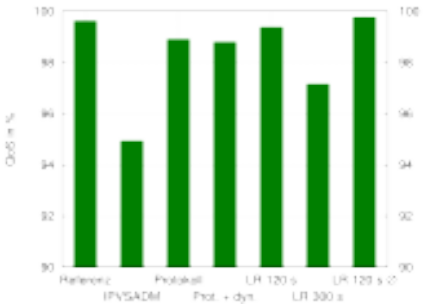
Abbildung 6.11: An- und Abschalt-Zeitpunkte unter Verwendung von linearer Regression mit 120 Sekunden Historie und Nutzung von Mittelwerten.

Für die Verwendung in einem realen Umfeld ist die *QoS* als wichtigste Metrik zu betrachten. Der Betreiber eines Webservers wird es in den meisten Fällen nicht riskieren, wegen einem ESS Kunden zu verlieren. Abbildung 6.12a zeigt, dass deshalb die beste Alternative für gute Resultate ein System mit Lastvorhersage ist. Man erkennt auch, dass man diese Verfahren vorsichtig umsetzen muss. Besonders die Länge der Historie ist entscheidend für das Resultat. Ist sie zu lang, reagiert das System zu langsam und die *QoS* leidet, ist sie zu kurz, regiert es zu schnell und man verliert Einsparpotenzial. Klassische schwellwertbasierte Verfahren funktionieren noch ausreichend gut, wenn mit den richtigen Last-Werten gearbeitet wird. Sie sind aber nicht in der Lage, einen Lastanstieg zu erkennen, weshalb sie in schwierigen Situationen schlechter abschneiden.

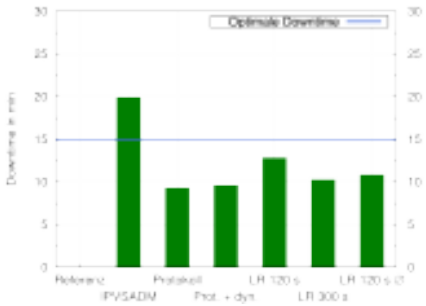
Die *FRT* (vgl. Abbildung 6.12c) konnte mit keiner der getesteten Methoden im Mittel das Level der Referenzmessung erreichen. Bei den beiden Messungen mit der linearen Regression und der Historie von 2 Minuten waren jedoch 2/3 der Messungen auf dem Level der Referenzmessung. Alle Zeiten (mit Ausnahme von `ipvsadm`), die erreicht wurden, liegen aber immer noch im geringen Millisekundenbereich. Diese Verzögerungen dürften für den Endnutzer nicht spürbar sein.

Abbildung 6.12b stellt die Energie dar, die eingespart werden konnte. Da *ipvsadm* seine hohen Einsparungen auf Kosten der *QoS* macht, steht diese Methode mit ihren hohen Einsparungen außen vor. Demzufolge schneidet die lineare Regression mit 120 Sekunden Historie am besten ab und liegt am nächsten am errechneten Optimum (siehe hierfür 5.5, Seite 72). Die einfachen Verfahren ohne Vorhersage schneiden hier deutlich schlechter ab.

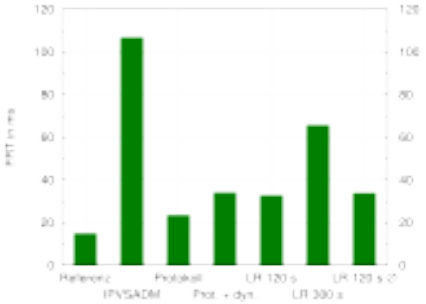
Auch Abbildung 6.12d unterstreicht noch einmal deutlich die Überlegenheit der LR 120 s Varianten. Ein Betreiber, der maximale *QoS* anbieten möchte, könnte sich dann eher für die LR 120 s ∅ Variante entscheiden. Sie hat noch eine leicht höhere *QoS*, auf Kosten einer geringeren Downtime.



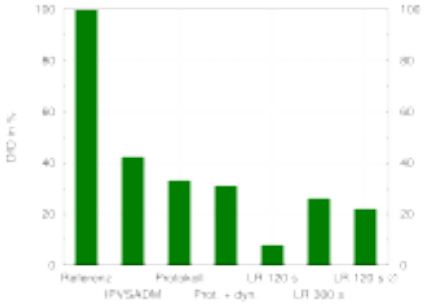
(a) Quality of Service (höher ist besser).



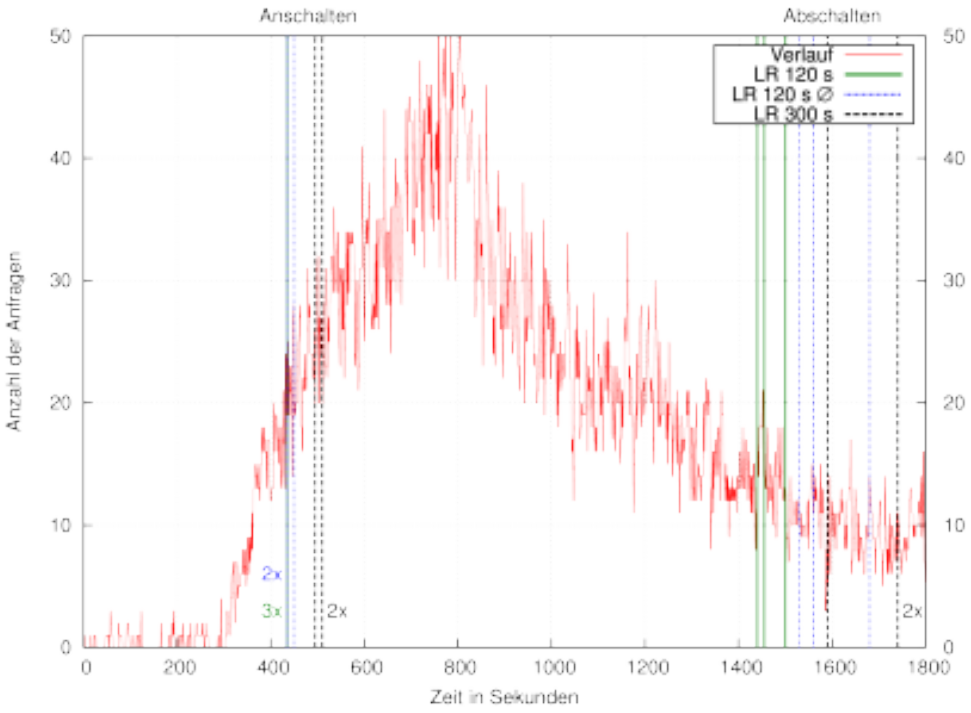
(b) Downtime (näher an Optimum Wert ist besser).



(c) First Response Time (niedriger ist besser).



(d) Deviation from Optimum (niedriger ist besser).



(e) An- und Abschalt-Zeitpunkte, unter Verwendung der linearer Regression mit 120 s/120 s ∅/300 s Historie.

Abbildung 6.12: Zusammenfassung aller Messungen.

Aufgrund des verwendeten Mittelwerts ist auch zu erwarten, dass diese Variante weniger anfällig für stärkere Lastschwankungen ist, als es im verwendeten Szenario der Fall war.
Abschließend zu diesem Kapitel stellt Abbildung 6.12e noch einmal alle Schaltzeitpunkte von *CHERUB* bei der Verwendung der linearer Regressionsanalyse gegenüber. Wie man erkennen kann, liegen jeweils die 120 Sekunden Läufe beim Anschalten gleich auf. Sie schalten fast identisch zum fast idealen Zeitpunkt die Maschinen an. Die LR 300 s Methode schaltet, wie bereits diskutiert, zu spät an. Beim Abschalten verhindert die Mittelwertbildung der LR 120 s ∅ Variante ein früheres Abschalten. Da die verwendeten Tools keine Möglichkeit bieten herauszufinden, wann genau das *SLA* verletzt wurde, kann leider nur spekuliert werden, an welcher Stelle überhaupt die Verluste auftreten. Da in der Referenzmessung allerdings auch *SLA*-Verletzungen auftreten, liegt die Vermutung nahe, dass dies während der Lastspitze geschieht. Erkenntnisse aus den späteren Kapiteln legen nahe, dass, zumindest in größeren Setups, ein beträchtlicher Teil der Timeouts kurz vor der Betriebsbereitschaft weiterer Knoten verursacht werden. Dies zeigt noch einmal die Wichtigkeit des richtigen Zeitpunktes zum Anschalten weiterer Ressourcen. Um den fehlenden 13,84 % zum Optimum noch näher zu kommen, könnte man darauf verzichten, beim Abschalten länger zu warten. Dies könnte aber in allgemeineren Fällen *State Flapping* begünstigen.

7 Skalierung der Laufzeit von CHERUB

Im vorherigen Kapitel konnte eindrucksvoll gezeigt werden, wie gut der vorgestellte Ansatz mit Lastvorhersage für das gewählte Setup funktioniert. Es steht allerdings die berechtigte Frage im Raum, ob dieser Ansatz auch skaliert und in größeren Setups nicht nur 1. mit sämtlichen Berechnungen in einer akzeptablen Laufzeit bleibt, sondern 2. auch genauso überzeugende Ergebnisse liefert. Ziel soll dabei der Nachweis für ein 100 Knoten großes Cluster sein. Bei einem noch größeren Cluster ist die Chance hoch, dass das Cluster eine komplexere Architektur (Caching-Nodes, Multi-Tier-Layer etc.) verwendet, welche nicht im Fokus dieser Arbeit steht.
Um die 1. Anforderung zu überprüfen, wurden alle Funktionen der CHERUB-API analysiert. In der prototypischen Version steht die Laufzeit aller Funktionen der API direkt in Abhängigkeit zur Anzahl der verwalteten Knoten. Auch wenn manche Funktionen nur für einen einzelnen Knoten ausgeführt werden (z.B. senden eines Bootbefehls) wird die Funktion im Worst-Case für alle Knoten aufgerufen, womit sich auch für diese Funktionen eine direkte Abhängigkeit ergibt. In diesem Kapitel soll deshalb überprüft werden, ob diese Abhängigkeit in großen Setups zu Problemen führt und wie gegebenenfalls auftretende Probleme gelöst werden können.
Alle in diesem Kapitel durchgeführten Messungen wurden auf der IB1 (siehe Tabelle 5.1, Seite 65) durchgeführt. Als Metrik wird die Laufzeit in Sekunden verwendet. Bei allen Messungen wurden 100 oder 1000 Iterationen durchgeführt (an X-Achse ablesbar). In jeder Iteration entspricht die Anzahl der Aufrufe der Nummer der Iteration (1. Iteration 1 Aufruf, 2. Iteration 2 Aufrufe usw.). Pro Iteration wurde die Messung 10 Mal wiederholt und ein Durchschnitt gebildet. Für eine bessere Übersicht wird auf die Darstellung der Standardabweichung in manchen Fällen verzichtet.
Als erstes werden die Python-Funktionen zur Prozesserzeugung (Abschnitt 7.1) untersucht. Diese werden häufig in der API verwendet und sollten deshalb möglichst performant sein. In diesem Zusammenhang wird ebenfalls die Performance von IPMI-1.5 und IPMI-2.0 verglichen. Anschließend, in Abschnitt 7.2, werden die zustandsverändernden Funktionen zum Anschalten, Abschalten, Anmelden und Abmelden der Knoten analysiert. In Abschnitt 7.3 und 7.4 wird auf die beiden informationssammelnden Funktionen eingegangen. Diese Funktionen werden in jedem Zyklus aufgerufen und berechnen wichtige Entscheidungen bzw. liefern Daten für selbige.

7.1 Prozesserzeugung und IPMI

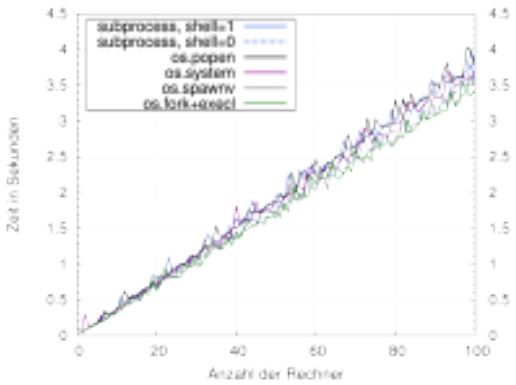
Alle CHERUB-API-Funktionen des prototypischen LVS-Moduls müssen in irgendeiner Weise mit dem darunterliegenden System interagieren, um entweder Informationen zur Weiterverarbeitung zu extrahieren oder um Kommandos an die Back-Ends abzusetzen. Für diese Zwecke müssen neue Prozesse erzeugt werden, die dann die entsprechenden Befehle (beispielsweise den Befehl `ipmitool -I lanplus -H <IP> -U root -P <password> power soft`, um einen Rechner per IPMI-2.0 abzuschalten) ausführen und ihre Ausgabe an das LVS-Modul zurück liefern. Für den Prototypen wurde auf das Pythonmodul `subprocess` zurückgegriffen, um Befehle auf Systeme-

bene auszuführen. Alex Martelli beschreibt in seinem Buch *Python in a nutshell* [151] fünf Wege, um Kommandos in einem neuen Prozess auszuführen. Da die Prozesserzeugung eine eher aufwendige Operation darstellt und, wie beschrieben, häufig vorkommt, sollte deren Laufzeit überprüft werden.
Neben dem verwendeten `subprocess`-Modul werden die Aufrufe `system`, `spawnv|spawnve`, `popen|popen2|popen3|popen4` sowie `execv|execl` vorgestellt und beschrieben. Es gibt noch diverse Varianten von `execv` und `execl`, aber diese nehmen nur andere Parameter bzw. Parameter in anderer Form entgegen und sind technisch identisch mit ihrer Basisvariante. Sie werden deshalb im Folgenden nicht weiter betrachtet. In der aktuellen online-Dokumentation von Python [152] wird darauf hingewiesen, dass die `popen`-Varianten seit Python 2.6 veraltet (*deprecated*) sind und stattdessen das `subprocess`-Modul verwendet werden soll. Derselbe Hinweis (ohne jedoch eine *deprecated* -Warnung) findet sich bei den Funktionen `system` und `spawnv|spawnve`. Diese drei Varianten basieren im Kern auf einer Kombination aus `fork()` und `exec()` [1] und abstrahieren diese auf verschiedene Art und Weise für den Nutzer. Eine Abstraktion findet auch im `subprocess`-Modul statt. Der Vorteil ist aber, dass durch seine Vielseitigkeit die vorherigen Varianten zu einem Interface zusammengefasst werden.
Obwohl ein Teil der Varianten bereits als *deprecated* eingestuft sind, werden diese interessehalber ebenfalls der Performanceanalyse unterzogen.
Um die Performance experimentell nachzuweisen, wurde die Funktion zum Abschalten von Rechnern exemplarisch mit allen genannten Varianten implementiert und getestet (Zielrechner war jedes Mal die IB5 [2]).
Da mehrere Funktionen IPMI-Kommandos verwenden, wurden zusätzlich zur Performance zwei verschiedene IPMI (siehe 2.1.2, Seite 11) Interfaces getestet. Mit dem Kommandozeilenbefehl `ipmitool` kann über eine Option entschieden werden, ob IPMI-1.5 (die Standardeinstellung) oder IPMI-2.0 (auch als *lanplus* bezeichnet) verwendet werden soll. IPMI-2.0 erweitert IPMI um Authentifikation und Verschlüsselung. Obwohl beides für die Messung nicht konfiguriert wurde, wird erwartet, dass die Nutzung von IPMI-2.0 Performancenachteile mit sich führt.

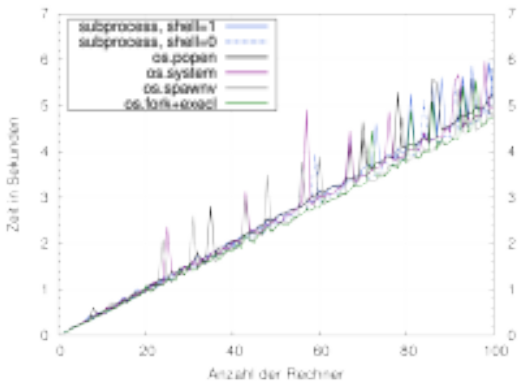
IPMI: Abbildung 7.1a und 7.1b zeigen die Ergebnisse der Messungen für IPMI-1.5 und IPMI-2.0. Man sieht, dass die Messungen sich innerhalb einer IPMI Variante kaum unterscheiden. Außerdem sieht man, dass für IPMI-2.0 mit zunehmender Knotenanzahl die Messpunkte immer mehr schwanken. Diese Schwankungen stehen mit IPMI-Fehlermeldungen in Verbindung, die nur bei der Verwendung von IPMI-2.0 aufgetreten sind. Einen direkten Vergleich der beiden IPMI Varianten sieht man in Abbildung 7.1c. Hier wurde der Durchschnitt der 6 IPMI-1.5 Varianten dem Durchschnitt der 6 IPMI-2.0 Varianten gegenübergestellt. Die Erwartung bezüglich der Performance bestätigt sich. IPMI-2.0 ist bei Verwendung von 100 Knoten bereits 1,5 Sekunden langsamer, was einem Unterschied von etwa 30 % entspricht. Wenn die zusätzlichen Eigenschaften von IPMI-2.0 nicht benötigt werden, sollte IPMI-1.5 verwendet werden, so wie es der Prototyp bereits tut.

[1] Das System geht dabei wie folgt vor: Mit `fork()` wird eine Kopie des Prozesses erzeugt und durch eine Verzweigung im Programmcode wird anhand der zurückgegebenen PID entweder dem normalen Programmablauf gefolgt (Elternprozess) oder der Programmcode wird mit Hilfe von `exec()` durch den Code des auszuführenden Kommandos ersetzt (Kindprozess).

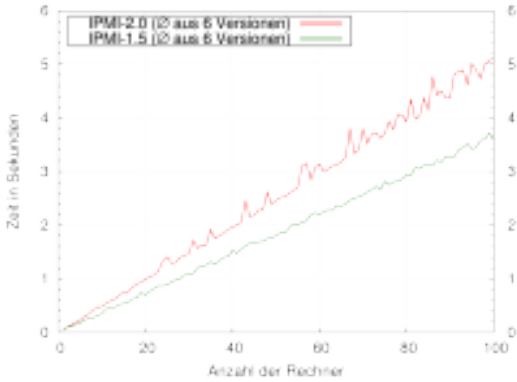
[2] IB5: 4 Kern Intel®Xeon®CPU E5520 mit 2.27GHz, 6 GB RAM, GNU/Linux 2.6.18-308.el5



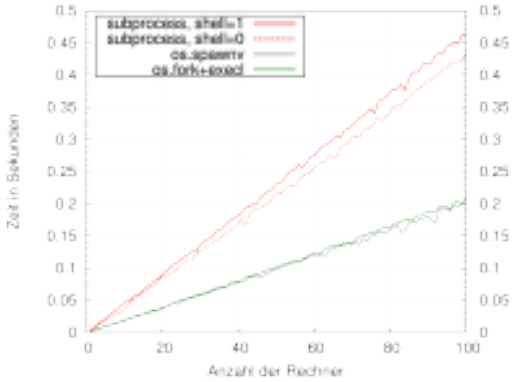
(a) IPMI-1.5 Varianten.



(b) IPMI-2.0 Varianten.



(c) Unterschied zwischen IPMI-1.5 und IPMI-2.0.



(d) NoWait Varianten.

Abbildung 7.1: Untersuchungen bezüglich IPMI-1.5, IPMI-2.0 und verschiedenen Prozesserzeugungsmethoden.

Prozesserzeugung: Wie bereits erwähnt, lassen sich anhand von Abbildung 7.1a und 7.1b nur geringe Unterschiede zwischen den Prozesserzeugungsmethoden feststellen. Bis auf *os.popen()* und *os.system()* können alle Befehle aber auch nicht blockierend ausgeführt werden. Durch ein erneutes Messen ohne blockieren kann der Unterschied der reinen Prozesserzeugung ermittelt werden. Die verwendete IPMI-Variante ist hier egal, da der Unterschied nicht mitgemessen wird. Die Performance der verschiedenen Varianten können in Abbildung 7.1d betrachtet werden. Man stellt dabei fest, dass die Prozesserzeugung nur 4 %-11 % des gesamten Befehls ausmacht. Der Großteil scheint Netzwerkkommunikation zu sein.

Von dem geringen Gesamtanteil abgesehen, erkennt man eindeutige Vorteile von *os.spawnv* und *os.fork+execl* gegenüber dem verwendeten *subprocess*, welches etwa doppelt so viel Zeit benötigt. Einen geringen Unterschied stellt noch die Verwendung des `shell` Parameters dar (nur *subprocess*). Ist dieser auf `True` gestellt, wird zunächst eine Shell erzeugt, in der wiederum das übergebene Kommando ausgeführt wird. Die Erzeugung der Shell kostet, wie man deutlich sehen kann,

extra Zeit und wird auch aus Sicherheitsgründen nicht empfohlen [3].

Fazit: Zusammenfassend kann gesagt werden, dass bei IPMI Kommandos das 1.5 Interface Verwendung finden sollte. Die Zusatzfunktionen, die IPMI-2.0 bietet, sind für den Anwendungsfall uninteressant und haben, auch durch das Auftreten von Fehlern, zu einer Performanceeinbuße von etwa 30 % geführt.
Für die reine Prozesserzeugung konnte zwar festgestellt werden, dass das verwendete *subprocess*-Modul am langsamsten war, aber da die Prozesserzeugung nur 4 %-11 % der gesamt Zeit des Befehls ausmacht, ist eine Änderung nur in sehr großen Setups notwendig. Für diesen Fall ist eine Reimplementierung an dieser Stelle nicht notwendig, zumal die Python-Dokumentation schreibt, dass das *subprocess*-Modul die älteren Varianten ersetzen soll.

7.2 Die zustandsverändernden Funktionen

Die vier zustandsverändernden Funktionen zum Starten, Herunterfahren, Registrieren und Abmelden der Knoten können hier gemeinsam betrachtet werden, da sie sich vor allem in der Häufigkeit ihrer Ausführung stark ähneln. Auch der Aufbau der Funktionen ähnelt sich stark. Sie bestehen im Kern jeweils entweder aus dem Aufruf eines IPMI-Befehls (Starten und Herunterfahren, via `ipmitool`), der an den entsprechenden Knoten gesendet wird, oder aus einem LVS-Befehl (Registrieren und Abmelden, via `ipvsadm`), der lokal ausgeführt wird. Die Funktionen zum Abmelden und Registrieren eines Knotens speichern zusätzlich zu ihrem LVS-Kommando noch das Gewicht für den jeweiligen Knoten und stellen dieses beim Registrieren wieder her. Die IPMI-Befehle werden für die Messung an die IB5 gesendet.
Abbildung 7.2 zeigt alle vier Funktionen im direkten Vergleich. Zu erkennen ist, dass die beiden Funktionen, die IPMI verwenden, länger benötigen als die beiden Befehle für LVS. Dies ist naheliegend, da bei den IPMI-Befehlen das Netzwerk involviert ist, welches um ein Vielfaches langsamer ist als eine Abfrage auf dem lokalen System. Außerdem erkennt man deutlich den linearen Zusammenhang zwischen der Anzahl der Rechner und der Ausführungszeit.

Fazit: Der Worst-Case, oder eine Worst-Case ähnliche Situation kommt nur sehr selten vor. Wenn sie auftritt, ist die gemessene Zeit bei 100 Knoten immer noch mehr als ausreichend, um einen problemlosen Ablauf zu garantieren. Man bedenke, dass für *CHERUB* im SLB-Szenario ein Zyklus von 15 Sekunden konfiguriert wurde. So lange die Abarbeitung aller Funktionen unterhalb dieses Intervalls bleibt, sind keine Probleme zu erwarten. Der Prototyp wird deshalb an dieser Stelle nicht angepasst.
Für ein größeres Setup könnte allerdings eine Parallelisierung des Verfahrens implementiert werden. Hierdurch würde vor allem die Netzwerkkommunikation überlappt werden, was einen deutlichen Performancegewinn ausmachen sollte.

[3] Als Standard-Shell wird `/bin/sh` verwendet. Wenn das Kommando, welches dann in der Shell ausgeführt wird, z.B. durch Nutzereingaben zusammengesetzt wird und diese nicht ausführlich geprüft werden, kann es zur sogenannten *shell injection* kommen [153]. Dabei handelt es sich um die Ausführung von ungewolltem, meist schadhaftem Code oder Kommandos, die einem Angreifer Zugriff auf das System verschaffen.

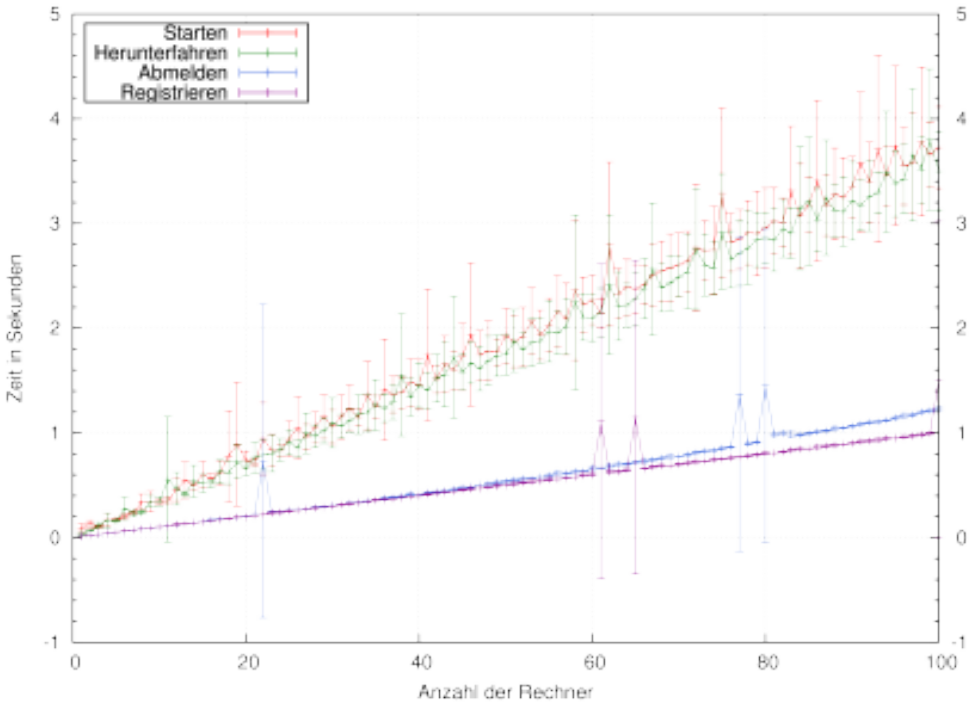


Abbildung 7.2: Performance der vier zustandsverändernden Funktionen.

7.3 Die Status-Funktion

Die erste der beiden zu betrachtenden informationssammelnden Funktionen ist die Status-Funktion. Sie übernimmt die Aufgabe, in jedem Zyklus den Zustand jedes einzelnen Knoten zu ermitteln. Abbildung 7.3 zeigt den Ablauf der prototypischen Status-Funktion. Rot markiert ist der Pfad im Ablauf, der besondere Laufzeit-Nachteile mit sich führt. Schuld daran ist die rot umrandete Box. An dieser Stelle weiß die Funktion bereits, dass der Knoten nicht am LVS angemeldet ist und muss noch die Unterscheidung zwischen den Zuständen *OFFLINE* und *DOWN* vornehmen. Daher muss überprüft werden, ob der Rechner bzw. der Dienst des Knotens erreichbar ist. Im vorliegenden Falle ist das der `HTTPD` (der Daemon des HTTP-Dienstes und somit der Apache), dessen Status in der prototypischen Version via `SSH` abgefragt wird. Dafür wird das lokale *Init-Skript* [4] mit der Option *Status* aufgerufen, welches dann der Status-Funktion zurückmeldet, ob der Dienst läuft oder nicht. Alternativ, könnte auch das Status-Modul des Apache-Webservers abgefragt werden.
Der Test gibt dem Knoten 1 Sekunde Zeit, um zu antworten. Wird innerhalb der Sekunde nicht geantwortet, wird davon ausgegangen, dass der Dienst nicht erreichbar ist und der Knoten abgeschaltet ist. Im Worst-Case muss deshalb davon ausgegangen werden, dass alle Knoten abgeschaltet sind und somit eine Laufzeit von mindestens der *Anzahl von Knoten* in Sekunden vorliegt (Bei 100 Knoten, 100 Sekunden). Obwohl eine lineare Laufzeit ($\mathcal{O}(n)$) in vielen Fällen akzeptabel ist,

[4] Ein *Init-Skript* (auch als Start-Skript oder Run-Skript bezeichnet) wird typischerweise für Systemdienste verwendet, um Nutzern ein möglichst einheitliches und einfaches Interface zum Staren, Beenden, Neustarten, Statusabfrage etc. zu geben.

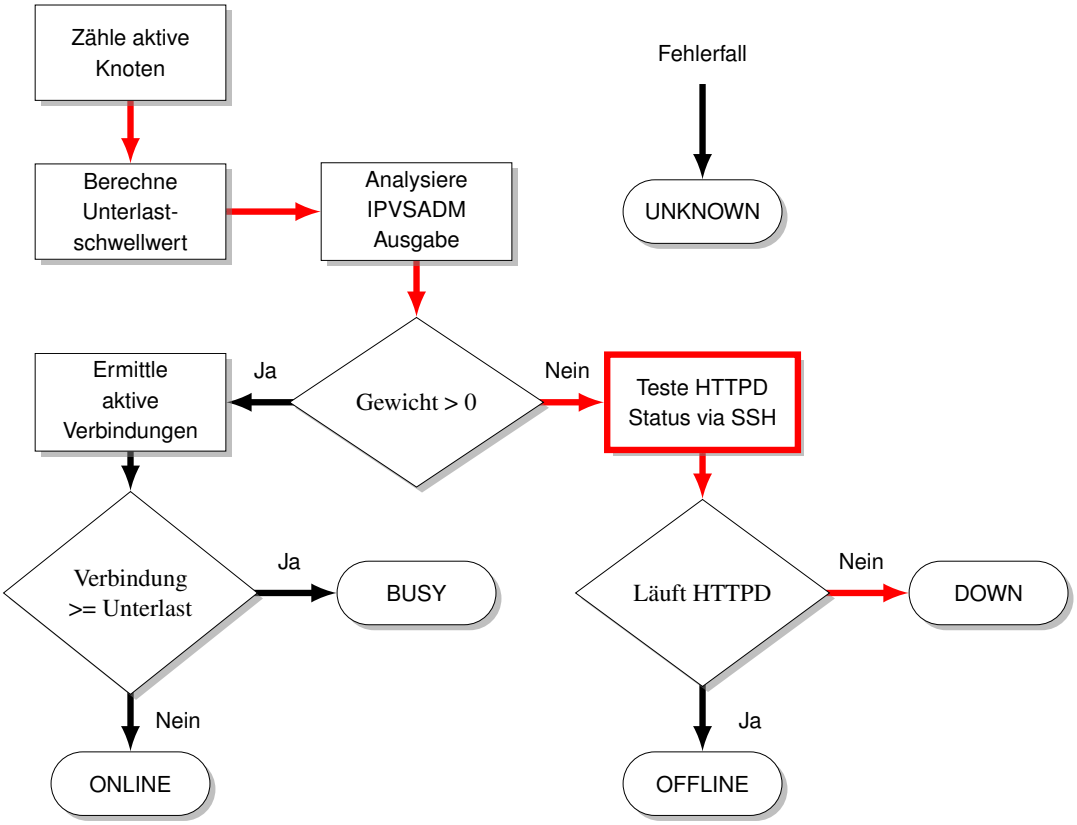


Abbildung 7.3: Ablaufdiagramm der prototypischen Status-Funktion.

ist im vorliegenden Fall die Dauer pro Knoten so hoch, dass eine lineare Laufzeit nicht mehr in Frage kommt. Die sequenzielle Abarbeitung muss deshalb durch einen parallelen Ansatz ersetzt werden.

Es wurden daher zwei parallele Alternativen implementiert. Die erste und simple parallele Version verwendet die vorliegende prototypischen Status-Funktion und führt diese pro Knoten in einem eigenen Thread aus. Hierdurch wird die Sekunde Wartezeit bereits überlagert und somit massiv Zeit gespart.

Die zweite Version ist eine parallele Reimplementierung, die nicht blockierende Sockets verwendet. Diese Variante überprüft den `HTTPD` nicht via `SSH`, sondern indem es eine Anfrage an das Status-Modul des Apache-Webservers schickt. Um das Status-Modul (siehe Abschnitt 2.2.2.2, Seite 23) zu verwenden, muss es in der Apache-Konfiguration geladen und aktiviert werden. Danach stellt es Informationen wie z.B. die Summe aller Zugriffe seit dem Start des Servers bereit. Der genaue Aufbau und Inhalt der Status-Seite, ist in Abbildung 8.5 (Seite 115) zu sehen.

Diese Anfrage wird nun für alle Knoten durchgeführt, nicht mehr nur für die, deren Gewicht auf 0 gestellt sind. Die Antwort des Servers enthält hier die aktuelle Anzahl der Anfragen an den Server und somit die aktuelle Last. Diese Information wird von der Last-Funktion direkt genutzt. In der prototypischen Version wurde diese Information durch ein `tcpdump`-Skript auf dem Load Balancer protokolliert und durch ein Log bereitgestellt. Aufgrund von Performance-Bedenken hinsichtlich

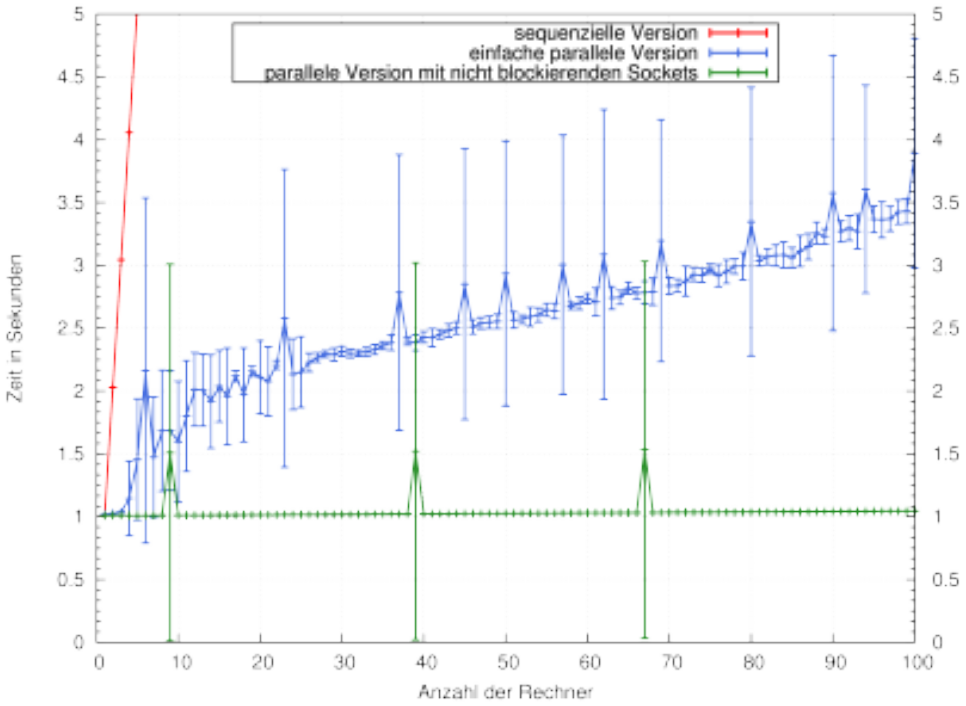


Abbildung 7.4: Performance Vergleich der Status-Funktionen.

`tcpdump` setzt die Reimplementierung auf die Status-Modul-Abfrage. Eine genauere Analyse der Performance von `tcpdump` findet sich in Abschnitt 7.5 (Seite 103).
Die Ergebnisse der Messungen sind in Abbildung 7.4 dargestellt. Man kann erkennen, wie alle Funktionen mindestens eine Sekunde warten. Diese Sekunde kann nicht umgangen werden. Bei der sequenziellen Version sieht man allerdings, wie sich die Sekunden aufaddieren. Die einfache parallele Version (die die sequenzielle Version in Threads ausführt) kann das Problem schon stark reduzieren und würde bereits eine akzeptable, wenn auch nicht optimale Lösung darstellen. Die Version mit den nicht blockierenden Sockets ist allerdings auch bei vielen Maschinen konstant schnell, da sie unter anderem auf die Erzeugung der Threads verzichtet.

Fazit: Ein 100 Knoten Setup wäre bereits mit der einfachen Parallelisierung Handhabbar. Die nicht blockierende Socket-Version ist allerdings in jeder Hinsicht der einfachen parallelen Variante überlegen. Sie hat nicht nur eine annähernd konstante Laufzeit, sondern gewinnt, praktisch nebenbei, noch wertvolle Lastinformationen, die in der Last-Funktion genutzt werden können.

7.4 Die Last-Funktion

Die zweite der beiden zu betrachtenden informationssammelnden Funktionen ist die Last-Funktion. Diese wird verwendet, um die aktuell vorherrschende Last des Systems zu ermitteln und zu analysieren. Obwohl *CHERUB* über mehrere API-Funktionen zur Lastbestimmung verfügt, ver-

wendet das LVS-Modul im Prototypen nur die Funktion, die pro Knoten ausgeführt wird. Sie berechnet anhand der Last des Knoten der letzten zwei Minuten eine lineare Regression und leitet abhängig von der ermittelten Steigung ab, ob der Knoten zukünftig in eine Überlastsituation kommen wird oder nicht. Wird eine Überlastsituation erkannt, müssen neue Knoten bereitgestellt werden. Da die Funktion pro Knoten ausgeführt wird, hat die verwendete Implementation in größeren Setups folgende Nachteile: 1) Die Berechnung einer linearen Regression ist aufwendig. Pro Knoten ausgeführt, kann dies zu Performanceproblemen führen. 2) Es gibt keine globale Sicht auf die Gesamtlast. Bei einer gleichmäßigen Verteilung der Last auf die Knoten würden alle Knoten gleichzeitig Überlast oder keine Überlast verzeichnen. Bei dem verwendeten 2-Knoten-Szenario spielt dies keine Rolle, da die Last des einzelnen Knoten gleich der Gesamtlast entspricht (zweiter Knoten aus). Sind beide Knoten an, muss keine Berechnung mehr stattfinden, da alle verfügbaren Ressourcen an sind.
Die reimplementierte Funktion behebt beide Probleme, indem sie die Gesamtlast des Systems betrachtet. Dazu summiert sie die Last der einzelnen Knoten auf und kann so die restliche freie Kapazität des gesamten Systems ermitteln. Diese globale freie Kapazität kann dann, ähnlich zu Formel 6.8 (Seite 88), verwendet werden, um mit Hilfe des Lastanstiegs und der Bootzeit, die aktuelle Situation zu bewerten. Der Vorteil ist, dass nun genau ermittelt wird wie viele Knoten abgeschaltet oder gestartet werden müssen. Aus diesem Vorgehen ergibt sich der Nachteil, dass das System sich jetzt auf ein optimales Scheduling der Last verlassen muss. Ist dies in einem Cluster allerdings nicht gewährleistet, würde es auch ohne *CHERUB* zu erheblichen Problemen kommen. Die bereits erzielten Ergebnisse ändern sich durch die Reimplementierung nicht, da, wie bereits erwähnt, in einem Cluster mit zwei Knoten das Verhalten dasselbe ist.
Abbildung 7.5 zeigt die gemessenen Ergebnisse. Die X-Achse ist hier logarithmisch, da sich die Laufzeiten der beiden Versionen stark voneinander unterscheiden. Auch wenn die Last-Funktion in einem 100-Knoten-Setup immer noch eine vertretbare Laufzeit aufweisen würde, wäre das Problem der globalen Lastermittlung noch vorhanden. Da die reimplementierte Version nicht mehr in Abhängigkeit zur Anzahl der Knoten steht, liegt ihre Laufzeit jetzt in $\mathcal{O}(1)$ und bleibt deshalb auch bei 1000 Knoten genauso performant wie bei sehr wenigen Knoten.

Fazit: Die reimplementierte Version behebt sowohl ein mögliches Performanceproblem in größeren Setups als auch das Problem der Unbekanntheit der globalen Last der prototypischen Version. Daher wird im weiteren Verlauf die Reimplementierung Verwendung finden.

7.5 TCPDump-Performance

Wie in Abschnitt 7.3 (Seite 100) angedeutet, wurde auf die Verwendung von `tcpdump`, aus Sorge um dessen Performance, verzichtet. Die Performance von `tcpdump` und der Einfluss von `tcpdump` auf andere laufende Dienste auf demselben Server, wurde daher separat in einer Studienarbeit von Sebastian Menski in [154] untersucht.
Bei der Untersuchung wurden drei wesentliche Merkmale identifiziert, welche die Leistung von `tcpdump` maßgeblich beeinflussen. Bei den Merkmalen handelt es sich um 1) den eingestellten Paket-Filter, welcher im Kernel bereits Pakete verwerfen kann, 2) die Anzahl der Bytes, die pro Paket weitergegeben werden (als *snaplen* bezeichnet), und 3) die Größe des Puffers, in dem die Pakete abgelegt werden.

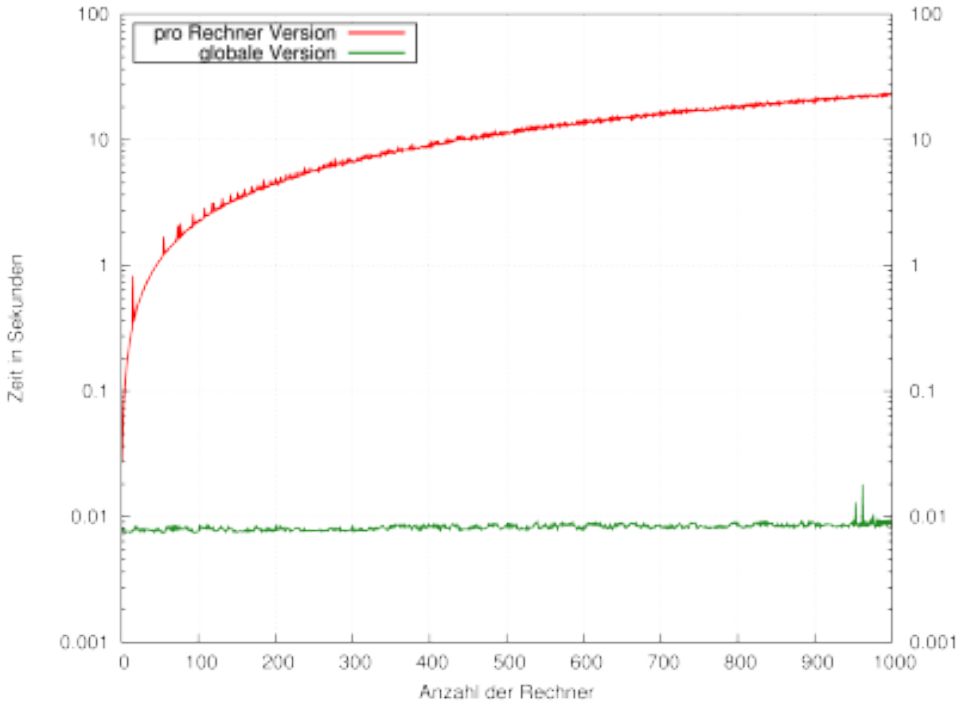


Abbildung 7.5: Performance Vergleich der Last-Funktion.

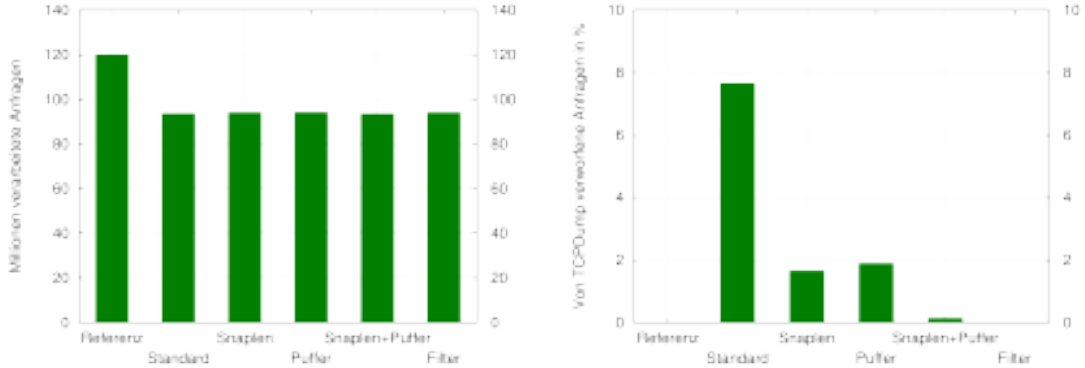
Um deren Einfluss zu messen, wurde in einem einfachen Messaufbau ein einzelner `nginx` Webserver von dem HTTP-Benchmark `wrk` [155] für 5 Minuten unter Last gesetzt. `nginx` wurde für die Messungen so konfiguriert, dass er mit einem *204 No Content Response* antwortet. Dadurch kann der Durchsatz des Webservers maximiert werden und `tcpdump`, unter der gegebenen Hardware, maximal belastet werden. Weitere Details der Arbeit können in [154] nachgeschlagen werden.
Die verschiedenen untersuchten Parameterkonfigurationen waren:

- **Referenz:** Messung ohne `tcpdump`.
- **Standard:** Messung mit Standardeinstellung und einem einfachen Filter.
- **Snaplen:** Stark verkleinerte *snaplen*, so dass nur noch die relevanten Daten für die Identifikation eines GET-Request aufgezeichnet werden.
- **Puffer:** Verdoppelung der Größe des Puffers, in dem Teile der Pakete zwischengespeichert werden.
- **Snaplen + Puffer:** Kombination aus **Snaplen** und **Puffer**.
- **Filter:** Einsatz eines zusätzlichen sehr spezifischen Filters.

Die verwendeten Parameter finden sich in Tabelle 7.1. Der in der Filter-Messung zusätzlich verwendete Filter überprüft genau die 4 Bytes in der TCP Nutzlast, die das „`GET␣`" eines GET-Request

Name	Snaplen	Puffer	Filter
Referenz	-	-	-
Standard	65535	2048	ip dst host 172.16.0.26 and tcp dst port
Snaplen	142	2048	ip dst host 172.16.0.26 and tcp dst port
Puffer	65535	4096	ip dst host 172.16.0.26 and tcp dst port
Snaplen + Puffer	142	4096	ip dst host 172.16.0.26 and tcp dst port
Filter	142	4096	ip dst host 172.16.0.26 and tcp dst port and 'tcp[((tcp[12:1] & 0xf0) » 2):4]=0x47455420'

Tabelle 7.1: Parameter für die `tcpdump` Experimente.



(a) Durch `tcpdump` verursachter Performanceverlust. **(b)** Von `tcpdump` verworfene Anfragen.

Abbildung 7.6: Zusammenfassung der `tcpdump`-Analyse.

enthält. Ein solcher Filter wurde von `tcpdump` bei der Protokollierung der Last, ab Abschnitt 6.2.2, verwendet.

Die Ergebnisse, dargestellt in Abbildung 7.6a und 7.6b, sind sehr aufschlussreich. In Abbildung 7.6a ist der Performanceverlust des laufenden Dienstes dargestellt. Die Einstellung von `tcpdump` ist dabei egal, sobald `tcpdump` verwendet wird, ist ein Performanceverlust von etwa 22 % zu beobachten. Dies ist ein deutliches Zeichen dafür, dass die Aufzeichnung der Anfragen nicht auf derselben Maschine geschehen sollte, auf der der Dienst läuft. In Abbildung 7.6b ist dargestellt, wie viele Anfragen `tcpdump` verwirft, welche aber den Webserver erreichen. Sowohl die Reduktion der *snaplen*, als auch die Vergrößerung des Puffers führen bereits zu einer starken Verringerung der Verluste. Die Kombination beider Parameter führt dazu, dass nur noch 0,145 % der Anfragen verlorengehen. Die zusätzliche Anwendung des sehr speziellen Filters lässt die Verluste auf 0,004 % sinken. Bei dieser geringen Verlustrate wäre eine Verwendung von `tcpdump` problemlos möglich.

Zusammenfassend kann gesagt werden, dass die Verluste sich für den konkreten Anwendungsfall in dieser Arbeit (Zählen von GET-Requests) so stark reduzieren lassen, dass auch eine RMS/Apache-unabhängige Lösung mit Hilfe von `tcpdump` möglich und eine Umstellung diesbezüglich nicht

notwendig gewesen wäre.

7.6 Fazit

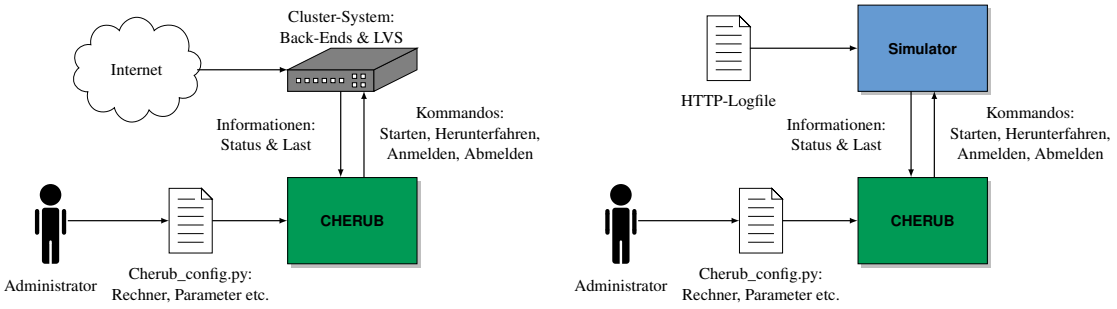
In diesem Kapitel wurde die Performance von *CHERUB* anhand von Komponententests überprüft. Es wurden alle Funktionen überprüft, die ein Problem in einem größeren Setup darstellen könnten. Dabei handelte es sich um Funktionen, deren Laufzeit direkt oder indirekt von der Anzahl der zu verwaltenden Knoten abhängt. Es konnte gezeigt werden, dass jene Komponenten entweder bereits performant genug waren oder durch eine performante Version ersetzt werden konnten. Es konnte außerdem gezeigt werden, dass eine Protokollierung der Last mit Hilfe von `tcpdump` alternativ in dem vorliegenden Anwendungsfall möglich wäre. Damit konnte Anforderung 1 (akzeptable Laufzeit in großem Setup), welche am Anfang des Kapitels aufgestellt wurde, nachgewiesen werden.

8 ClusterSim

Nachdem im vorherigen Kapitel die Performance von *CHERUB* nachgewiesen werden konnte, soll nun Anforderung 2 (Funktionalität des Algorithmus in größeren Setups, siehe Kapitel 7, Seite 96) nachgewiesen werden.
Ein Nachweis anhand von Messungen kann in diesem Fall leider nicht durchgeführt werden, da die Hardware bzw. das Budget dafür nicht vorhanden ist. Auch eine Messung mit Hilfe von Cloud-Diensten (z.B. Amazon Cloud [156], Microsoft Azure [157] oder STRATO ServerCloud [158]) liegt nicht im Budget der Universität. Eine Messung mit Hilfe von Cloud-Diensten wäre zusätzlich nicht empfehlenswert, da man bei den Anbietern typischerweise nur virtuelle Maschinen (VMs) mieten kann und man daher keine vollständige Kontrolle über die physikalischen Maschinen hat. Messungen in einer Cloud wären deshalb nicht aussagekräftig. Obwohl man bei z.B. Amazon extra dedizierte Maschinen mieten kann, würde das Mieten von 100 der kleinsten möglichen dedizierten Maschinen-Instanzen (*Linux on m4.large Dedicated*), bei einer geplanten Auslastung von 50 %, über einen Monat hinweg bereits 5747 $ kosten [159].
Raj Jain beschreibt in seinem Standardwerk zur Performance Analyse in [160] neben dem Messen zwei weitere Verfahren, die Modell-Analyse und die Simulation. Bei den Vor- und Nachteilen zwischen der Modell-Analyse und der Simulation stellt sich die Modell-Analyse als schnelles und billiges, aber dafür ungenaues Verfahren heraus. Jain bemerkt dazu:

> *"In general, analytical modeling requires so many simplifications and assumptions that if the results turn out to be accurate, even the analysts are suprised."*
>
> aus [160], Seite 31

Da die Analyse der Skalierung möglichst genau sein soll, wird die Simulation als Methodik gewählt.
Das weitere Kapitel ist wie folgt aufgebaut. Es wird damit begonnen, Anforderungen aufzustellen, die ein Simulator erfüllen muss, wenn er für den Zweck geeignet sein soll, eine Skalierbarkeit von *CHERUBs*-Algorithmus nachzuweisen. In Abschnitt 8.2 (Seite 109) werden anschließend mögliche bestehende alternative Simulatoren gezeigt und anhand der aufgestellten Anforderungen ausgeschlossen. Abschnitt 8.3 (Seite 110) stellt die Architektur von *ClusterSim* vor, welche die aufgestellten Anforderungen erfüllt. Nachfolgend, in Abschnitt 8.4 (Seite 113), werden notwendige Anpassungen an *CHERUB* vorgestellt, damit eine Koppelung zu *ClusterSim* funktioniert. Anschließend wird in Abschnitt 8.5 (Seite 115) die genaue Funktionsweise des Simulators und dessen Algorithmus präsentiert und durch ein Beispiel verdeutlicht. In Abschnitt 8.6 (Seite 124) wird die Umsetzung des Energieverbrauchs in Abhängigkeit zur Auslastung der simulierten Maschinen gezeigt. Abschließend werden in Abschnitt 8.7 (Seite 124) die Grenzen des Simulators aufgezeigt und in Abschnitt 8.8 (Seite 126) wird der Simulator mit mehreren Experimenten validiert.



(a) Interaktion bei normalem Betrieb.

(b) Interaktion mit einem Simulator.

Abbildung 8.1: Vergleich der Interaktion von *CHERUB* mit und ohne Einsatz eines Simulators.

8.1 Anforderungen an einen Simulator

Damit der Einsatz eines Simulators glaubhaft und möglich ist, muss dieser bestimmte Bedingungen erfüllen. Darunter fallen:

- Die Größe des zu simulierenden Clusters muss möglichst groß sein (mindestens 100 Maschinen).
- Der simulierte Dienst muss einen Apache-Webserver abbilden.
- Die simulierten Maschinen müssen Muticore-Eigenschaften besitzen und abbilden.
- Die simulierten Maschinen müssen die Möglichkeit besitzen, gestartet und heruntergefahren zu werden.
- Die simulierten Maschinen müssen ihren Energieverbrauch protokollieren.
- Die Ergebnisse müssen validiert und nah an der Realität sein.
- Es muss sich um einen *trace driven* Simulator handeln, der einen Trace als Eingabe bekommt und diesen simuliert abspielt.
- Der Simulator muss über eine API zu *CHERUB* verfügen und diesem über die API eine realistische Clusterumgebung vortäuschen.

Nur wenn diese Eigenschaften erfüllt sind, kann ein realistischer Vergleich mit den bereits erzielten Ergebnissen vorgenommen werden.
Abbildung 8.1 zeigt, welche Aufgabe ein Simulator übernehmen müsste. In Abbildung 8.1a sind *CHERUBs* Schnittstellen zu sehen und die Interaktion mit dem Cluster-System, welches seine Anfragen aus dem Internet erhält. Abbildung 8.1b zeigt, wie das gesamte Cluster-System nun von einem Simulator imitiert wird, welcher seine Anfragen aus einem HTTP-Logfile (Trace) erhält. Dieser Trace könnte auch eine Aufzeichnung aus einem echten Setup sein. Der Simulator muss nun genau die Schnittstellen von *CHERUB* bereitstellen, damit dieser so handelt, als würde er ein reales Cluster verwalten.

8.2 Alternativen

Neben einer eigenen Entwicklung muss zunächst sondiert werden, ob es bereits alternative, frei verfügbare Simulatoren gibt, welche ohne wenig Aufwand um fehlende Komponenten ergänzt werden könnten. Insbesondere die Schnittstelle zu *CHERUB* kann eine bereits bestehende Implementation nicht bieten. Es wurden drei mögliche Alternativen zu einer eigenen Implementierung überprüft, die hier kurz vorgestellt werden sollen.

Schneidenbach-Simulator: Die erste Alternative stellte der von Lars Schneidenbach in seiner Dissertation selbst entwickelte Simulator dar ([161], Kapitel 6.5.4). Da Lars Schneidenbach selber Doktorand des Lehrstuhls von Prof. Schnor war, lag es nahe, seine Arbeit eventuell nachzunutzen. Bei genauerer Untersuchung des Quellcodes wurde relativ schnell deutlich, dass eine Nachnutzung nicht geeignet ist. Punkte hierfür sind unter anderem, dass es sich dabei um einen klassischen ereignisgetriebenen (*event driven*) Simulator handelt, der die simulierten Maschinen nicht detailliert genug für das Vorhaben modelliert. Weiterhin ist der Simulator nicht in der Lage, einen Trace als Eingabe zu verwenden, sondern behilft sich für die Arbeitslast einer entweder negativ exponentiellen oder einer gleichmäßigen Verteilung. Der Simulator unterstützt außerdem keine Multicore-Rechner und in der Dissertation ist keine Validierung zu finden. Der mit einer Anpassung verbundene Aufwand und die Unkenntnis über das genaue Vorgehen in einem fremden Simulator sind zusätzlich starke Argumente für eine Eigenentwicklung.

CloudSim: Die zweite Alternative wurde im Rahmen der Lehrveranstaltung „Leistungsanalyse: Messen, Modellieren und Simulation" (Sommersemester 2015, Universität Potsdam) von der Studentin Nadine Falkmann untersucht. Es handelt sich dabei um das Tool *CloudSim* [91], welches von der Universität Melbourne in der Gruppe um Prof. Buyya entwickelt wurde. Es findet vor allem Anwendung in Arbeiten innerhalb der Gruppe (acht aufgelistete Publikationen auf [91]), aber wurde auch schon von Externen verwendet (z.B. von Geronimo et al. in [162]). Es handelt sich bei *CloudSim* um ein auf Java basierendes Simulations-Framework, welches die Grundstrukturen für ein Cloud-Rechenzentrum bietet. Es können z.B. Repräsentationen von physikalischen Maschinen erzeugt werden, denen dann VMs zugewiesen werden können, die wiederum Tasks, sogenannte Cloudlets, abarbeiten. Bei der von Frau Falkmann vorgenommenen Exploration stellte sich heraus, dass eine Nutzung des Frameworks eine erhebliche Einstiegsbarriere besitzt. Es wurden außerdem nicht nachvollziehbare Schedulingergebnisse beobachtet. Auch der Versuch, Hilfe in der *CloudSim* GoogleGroup zu finden, in der auch *CloudSim*-Entwickler Mitglieder sind, brachte keinen Erfolg.
Frau Falkmann hat, trotz der anfänglichen Schwierigkeiten, ein kleines Energiesparszenario implementiert. Dieses Szenario entspricht in etwa dem Umfang des Setups aus Kapitel 5. So besitzt es zwei Back-Ends, bei dem eines abgeschaltet wurde. Um besagtes Szenario abzubilden, mussten allerdings recht grobe Abstraktionen und Notlösungen genutzt werden. So kann in *CloudSim* beispielsweise nicht die Ebene der VMs eingespart werden und um ein Cluster abzubilden, musste pro physikalischer Maschine trotzdem immer eine einzelne VM erzeugt werden, welche die physikalische Maschine dann aber dediziert verwendet. Besonders stark war die Abstraktion hinsichtlich des Einspielens des Traces. Um diesen abzubilden, musste für die VMs in *CloudSim* ein Task erzeugt werden, der mit einem gewissen Auslastungsprofil verbunden ist (dieses gibt die

Auslastung pro Sekunde an). Dieses Auslastungsprofil musste nun mit der durch den Trace im echten Szenario erzeugten Auslastung in Einklang gebracht werden und dann auf die Tasks der beiden VMs aufgeteilt werden. Insgesamt war dies eine sehr umständliche, unpraktische und für große Szenarien praktisch nicht durchführbare Aufgabe. Weitere Details können der Ausarbeitung entnommen werden [163]. Abschließend wurde von Frau Falkmann festgestellt, dass eine Nachnutzung von *CloudSim* nicht empfohlen werden kann. *CloudSim* ist, wie der Name schon sagt, für Simulationen gedacht, deren Fokus auf der Cloud liegt und entsprechend ungeeignet für das vorliegende Szenario. Als Betreuer der Arbeit bin ich zu demselben Ergebnis gekommen.

DESDSim: Bei *DESDSim* handelt es sich um eine Erweiterung von *CloudSim*, welche von Xiong et al. in [164] vorgestellt wird. Die Autoren erweitern *CloudSim* um die Möglichkeiten, Maschinen abzuschalten, einzelne Kerne abzuschalten und die Leistung einzelner Kerne per DVFS anzupassen. Um *CloudSim* so zu erweitern, mussten die Autoren, ähnlich wie Frau Falkmann, bestimmte Abstraktionen vornehmen. Wie bereits erwähnt, lässt sich die VM-Schicht von *CloudSim* nicht umgehen, so dass *DESDSim*, wie auch Frau Falkmann, für die Simulation eines einzelnen Webservers, eine einzelne physikalische *CloudSim* Maschine mit einer einzelnen VM verwendet. Diese VM stellt dann den Webserver dar.
Die Validierung von *DESDSim* fällt sehr gering aus, es wird lediglich eine Testmessung mit 50 simulierten Webservern durchgeführt und überprüft, ob die von ihnen berechneten Metriken (durchschnittliche Antwortzeit, verworfene Anfragen, Energieverbauch) plausibel sind.
Da *DESDSim* nicht frei verfügbar ist und die Autoren auf eine Anfrage nach Verfügbarkeit des Quellcodes nicht reagiert haben, konnte *DESDSim* nicht weiter überprüft werden.

Eigenentwicklung: Wie gezeigt, wäre eine Anpassung der bestehenden Alternativen so aufwendig, fehleranfällig oder stark abstrahiert, dass eine Eigenentwicklung hier die praktikablere und genauere Lösung darstellt. Die Eigenentwicklung trägt den Namen *ClusterSim* und wurde entsprechend den aufgestellten Anforderungen (siehe Abschnitt 8.1, Seite 108) entwickelt.

8.3 Konzept und Architektur

Abbildung 8.2 zeigt eine Übersicht über die Architektur von *ClusterSim*, den wichtigsten Komponenten und deren Zusammenspiel.

Virtuelles Cluster: Der Kern von *ClusterSim* bildet das *virtuelle Cluster*, welches wiederum aus virtuellen Knoten (VNodes) und dem Lastverteiler besteht. Außerdem speichert es clusterspezifische Informationen und Datenstrukturen wie den gesamten Energieverbrauch, die Dauer, bis eine Zeitüberschreitung stattfindet, die Cluster-Queue (welche Anfragen speichert) etc.

Lastverteiler: Der Lastverteiler ist für das Scheduling der eingehenden Anfragen verantwortlich und bildet damit den LVS des realen Setup nach. Als Scheduling-Verfahren wurde u.a. Round-Robin umgesetzt, welches in allen späteren Experimenten auch verwendet wird. Wie bereits in Abschnitt 5.2.2 erwähnt, ist sich Wissenschaftler nicht einig, welches Verteilungsverfahren (Round-Robin bzw. Least-Connection) das bessere ist. Auf Grund der einfachen Implementation wurde sich daher für Round-Robin entschieden. Für spätere Experimente wurden bereits ebenfalls

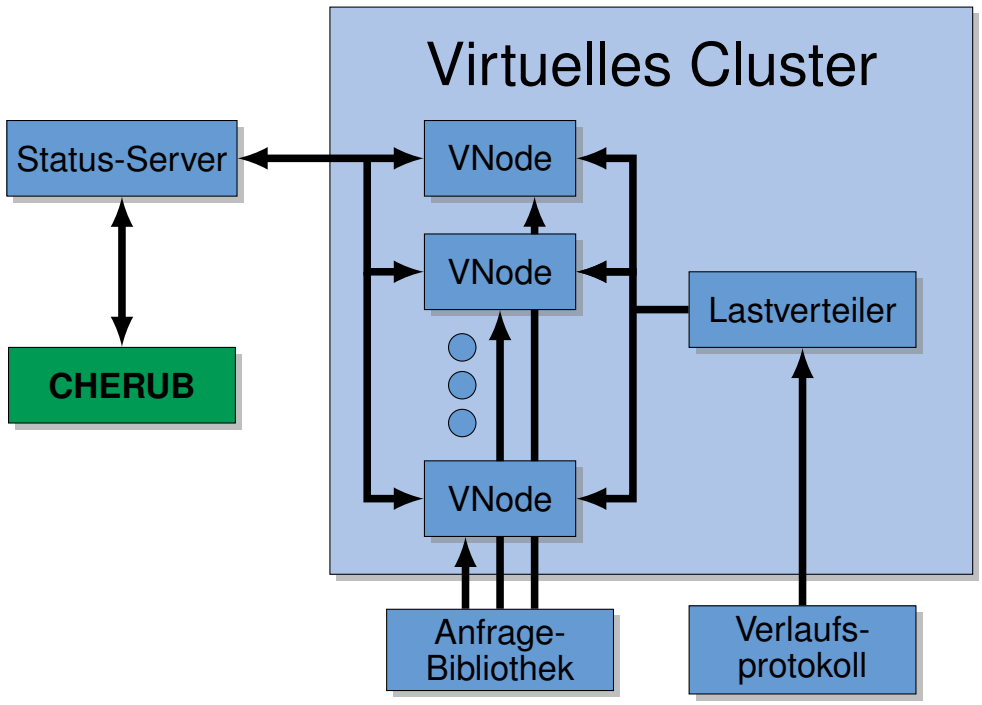


Abbildung 8.2: Architektur von *ClusterSim*.

Weighted Round-Robin, Highest Credits und Next Credits implementiert (siehe Abschnitt 10.2.4, Seite 10.2.4). An dieser Stelle kann der Code jedoch um jedes beliebige Verfahren erweitert werden.

Verlaufsprotokoll: Wann und welche Anfragen vom Scheduler verteilt werden müssen, hängt von dem verwendeten Trace ab. Dieser wird dem Simulator beim Start als Eingabe übergeben und muss im Apache Common Log Format (siehe Beispiel 7, Seite 70) vorliegen.

VNodes: Die virtuellen Knoten sind der eigentliche Kern des Simulators. Jeder VNode repräsentiert eine simulierte Maschine und übernimmt damit die Abarbeitung der vom Scheduler an ihn weitergereichten Anfragen. Jeder VNode besitzt und speichert eine Reihe von Attributen, darunter z.B. Zeit zum Booten, aktueller Zustand (äquivalent mit den *CHERUB*-Zuständen), Länge der BLQ, die BLQ selbst inklusive der darin befindlichen Anfragen, Anzahl CPUs, Anzahl der Apache-Worker, verbrauchte Rechenleistung, bereits gecachte Anfragen etc. Die VNodes besitzen, im Gegensatz zu anderen Simulatoren wie z.B. *CloudSim*, keine Attribute wie *Instruktionen pro Sekunde* oder *Leistung in Ghz*. Die Leistungsmerkmale werden indirekt durch die Anfrage-Bibliothek bereitgestellt. Jeder VNode besitzt außerdem einen simulierten Webserver-Cache. In diesem werden alle Anfragen gespeichert, die von dem VNode bereits bearbeitet wurden. Erhält ein VNode eine Anfrage, die sich bereits im Cache befindet, wird für sie die gecachte Bearbeitungszeit aus der Anfrage-Bibliothek verwendet. Der Cache wird geleert, wenn er abgeschaltet wird.

Anfrage-Bibliothek: In dieser Bibliothek müssen alle Anfragen verzeichnet sein, die in dem genutzten Trace Verwendung finden. Sie listet pro Anfrage die Berechnungsdauer der Anfrage, gecached und ungecached, auf. Diese Informationen werden dann von den VNodes verwendet, um die gesamte Dauer einer Anfrage zu berechnen. Hat ein VNode eine Anfrage bereits beantwortet, wird er für die Berechnung die gecachte Zeit verwenden.
Die Rechenleistung von jedem Kern eines VNodes beträgt durch diese Abstraktion genau 1 Sekunde Bearbeitungszeit pro Sekunde, welche unter den Arbeitern des Kerns gleichmäßig aufgeteilt wird. Beispiel 12 soll das allgemeine Vorgehen verdeutlichen.

Beispiel 12 Ein VNode mit einem Kern und einem Worker bekommt vom Scheduler zwei Anfragen A und B zugewiesen. Anfrage A benötigt laut Anfrage-Bibliothek 12 ms bzw. 2 ms (ungecached/gecached) zur Bearbeitung und Anfrage B 22 ms bzw. 15 ms.
Anfrage A ist nicht im Cache verzeichnet, da sie noch nie vom VNode bearbeitet wurde. Sie verbraucht 22 ms Arbeitszeit auf dem Kern des VNodes, welcher nach dessen Bearbeitung noch eine Kapazität für 978 ms Bearbeitungszeit hat. Anfrage B ist danach an der Reihe. Der VNode stellt fest, dass die Anfrage in seinem Cache verzeichnet ist und er sie daher schon einmal bearbeitet hatte. Die gecachte Anfragedauer beträgt 15 ms. Nach Abarbeitung von Anfrage B hat der VNode also noch 963 ms Bearbeitungszeit übrig.
Ist pro Sekunde die gesamte Dauer aller Anfragen an einem Kern größer als 1 Sekunde, kann erst in der nächsten Sekunde der Simulation weiter gerechnet werden. In dieser können aber bereits neue Anfragen am System ankommen.

□

Dieses Beispiel soll nur eine grobe Idee des Ablaufs geben. Wie die einzelnen Schritte des Algorithmus aussehen, wird in Abschnitt 8.5 (Seite 116) vorgestellt.

Status-Server: Die Schnittstelle zu *CHERUB* bildet der Status-Server. Dieser läuft in einem separaten Thread und bietet *CHERUB* die Möglichkeit, mit ihm über einen lokalen Unix-Socket zu kommunizieren. Der Status-Server ist dazu in der Lage, alle Anfragen von *CHERUB* zu erkennen und ihm entweder Informationen über den Zustand des Clusters zu senden oder die von *CHERUB* abgesetzten Befehle an die VNodes entgegenzunehmen und auszuführen. Um die Kommunikationen mit dem Status-Server zu ermöglichen, muss *CHERUB* erweitert werden. Die Erweiterungen sehen wie folgt aus.

Lebenszyklus einer Anfrage: Anfragen folgen einem gewissen Lebenszyklus innerhalb des Simulators. Abbildung 8.3 stellt diesen Zyklus anschaulich dar. Dabei stellen die Kugeln einzelne Anfragen dar (blau = neue Anfragen, orange = Retransmissions, grün = Mix).
Eine Anfrage beginnt ihre Existenz, indem sie aus dem Verlaufsprotokoll (Quelle) extrahiert wird und in die Cluster-Queue eingefügt wird. Diese Queue repräsentiert den globalen Zugang des laufenden Dienstes und hat im realen Setup als Äquivalent den LVS.
Der Lastverteiler übernimmt dann die Aufgabe, alle Anfragen, die sich in der Cluster-Queue befinden, abhängig vom gewählten Scheduling-Verfahren an die VNodes zu verteilen. Er tut dies, indem er die Anfragen in die BLQ der einzelnen VNodes hängt.
Kann eine Anfrage nicht rechtzeitig von einem VNode bearbeitet werden oder passt nicht mehr in seine BLQ, wird sie verworfen und in die Retransmission-Queue gehängt. Von hier wird sie

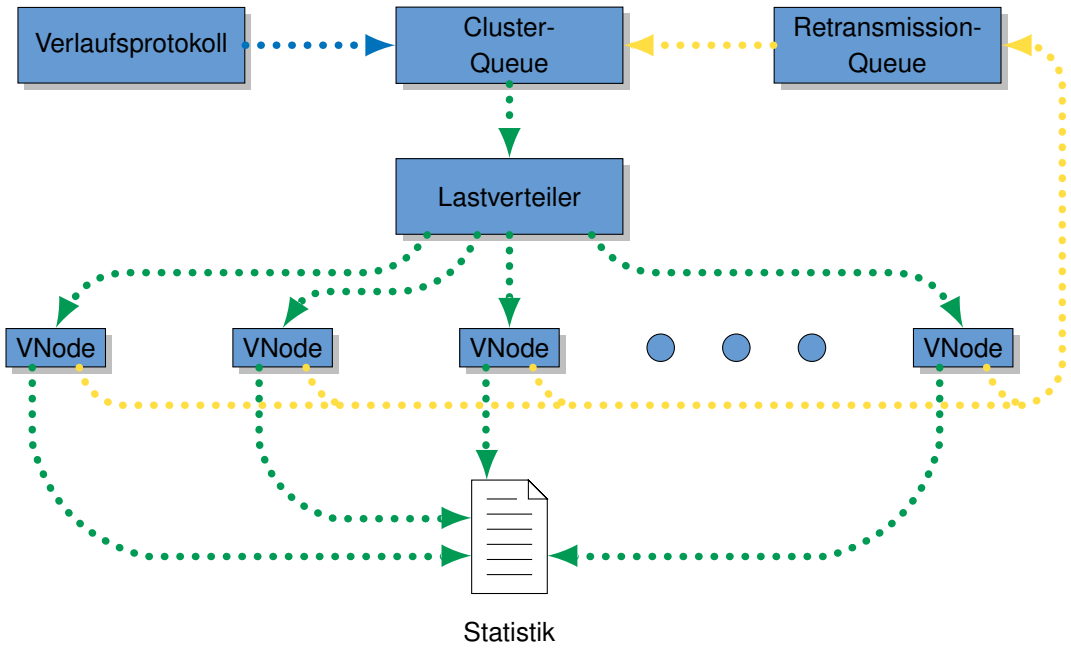


Abbildung 8.3: Der Lebenszyklus der Anfragen.

nach einer kurzen Wartezeit in die Cluster-Queue gehangen, von der aus sie wieder an die VNodes versendet wird. Hat die Anfrage keine Möglichkeit mehr, über diesen Umweg rechtzeitig und ohne eine Zeitüberschreitung zu verursachen bearbeitet zu werden, endet ihre Existenz und statistische Daten werden über sie protokolliert (Senke).

Wird eine Anfrage erfolgreich von einem VNode bearbeitet, endet deren Existenz ebenfalls mit der Erhebung der relevanten statistischen Daten (endgültige Bearbeitungszeit, Zeitpunkt der Fertigstellung).

8.4 Anpassungen an *CHERUB*

Um *CHERUB* mit *ClusterSim* verwenden zu können, müssen einige wenige Anpassungen vorgenommen werden. Bei den Anpassungen wurde darauf geachtet, diese so gering wie möglich zu halten, um bei den Simulationen einen zusätzlichen Eindruck von der Gesamtperformance von *CHERUB* zu erhalten.

Abbildung 8.4 zeigt die Interaktion zwischen *CHERUB* und *ClusterSim* nach den Anpassungen. Sie gleicht praktisch der geplanten Vorgabe aus Abbildung 8.1b (Seite 108) mit der Ausnahme der LVS-Abfrage. Diese hätte zwar ebenfalls durch den Simulator ersetzt werden können, um die Funktionen von *CHERUB* aber so wenig wie möglich verändern zu müssen, wurde diese Abfrage beibehalten. So ist die Vergleichbarkeit besser gegeben.

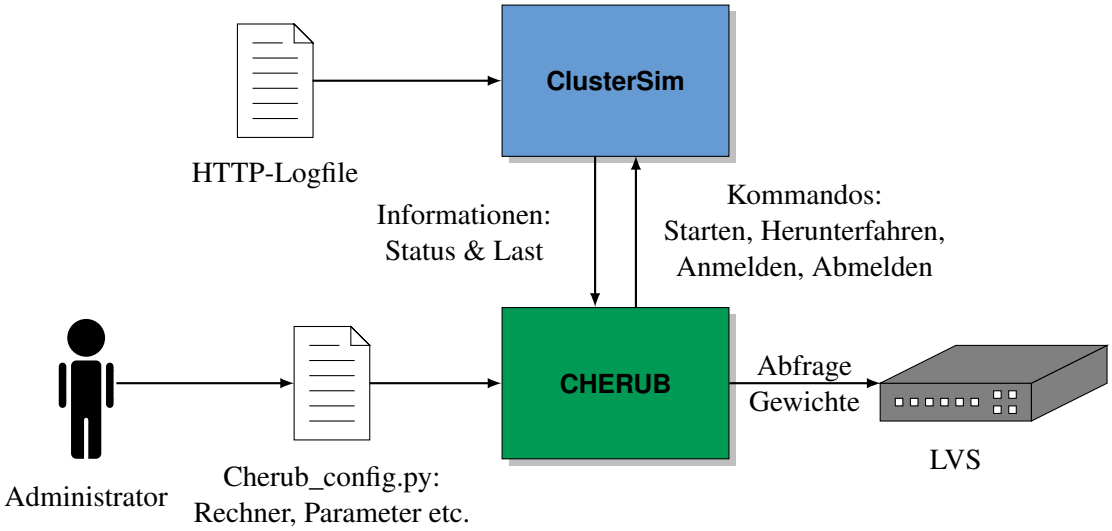


Abbildung 8.4: Interaktion von *CHERUB* mit *ClusterSim*.

8.4.1 Erweiterungen der Status-Funktion

Sowohl der Status der Rechner als auch deren Last werden in der reimplementierten Fassung (vorgestellt in Kapitel 7, Seite 96) gemeinsam durch die Abfrage des Status-Moduls der Apache-Webserver ermittelt. Deshalb gibt es im Falle der Informationsbeschaffung Erweiterungen des *CHERUB*-Codes. Die erste Erweiterung ist die Erkennung, ob es sich um einen virtuellen Knoten handelt oder nicht. Diese Unterscheidung wird anhand des Namens durchgeführt und jede Funktion, die angepasst werden musste, muss diese Identifikation durchführen. Die zweite Erweiterung sorgt dafür, dass die darauf folgende Kommunikation nicht mit einem Netzwerk-Socket (`AF_INET`), sondern mit einem lokalen Unix-Socket (`AF_UNIX`) durchgeführt wird. Über den lokalen Unix-Socket ist *CHERUB* mit dem Status-Server von *ClusterSim* verbunden, welcher die Anfrage von *CHERUB* entgegennimmt und eine Antwort erzeugt, die dem Schema des Status-Moduls des Apache-Webservers gleicht. Abbildung 8.5 zeigt die so erzeugte Antwort. *ClusterSim* verändert lediglich den in Zeile 8 angegebenen Wert, da dieses Feld von *CHERUB* als einziges verwendet wird.

Ist der VNode nicht am LVS registriert, schickt der Status-Server den Zustand des VNodes direkt, ohne dass eine weitere Interpretation stattfinden muss.

Die Änderungen beeinflussen die Laufzeit der Funktion nicht. Die Detektion eines VNodes erfolgt in $\mathcal{O}(1)$, da nur der erste Buchstabe überprüft werden muss, und die Änderung des Sockets sorgt für keinen zusätzlichen Aufwand. Durch die Ersetzung des Sockets fällt zwar die Netzwerkkommunikation weg, diese ist aber nicht wichtig für die Analyse des Algorithmus und würde bei einer Zeitüberschreitung von einer Sekunde ohnehin, mit der Annahme eines abgeschalteten Knotens, abgebrochen werden.

```
1  HTTP/1.1 200 OK
2  Date: Fri, 07 Nov 2014 16:03:18 GMT
3  Server: Apache/2.2.3 (CentOS)
4  Content-Length: 557
5  Connection: close
6  Content-Type: text/plain; charset=ISO-8859-1
7
8  Total Accesses: 385
9  Total kBytes: 2
10 Uptime: 24569
11 ReqPerSec: .0156702
12 BytesPerSec: .0833571
13 BytesPerReq: 5.31948
14 BusyWorkers: 1
15 IdleWorkers: 19
16 Scoreboard: ..._.........................................
       ..................................................._.......
       ..........................................................
       ..........................................................
       ..........................................................
       ..........................................................
       ..........................................................
       ..................................................___W____
       __________
```

Abbildung 8.5: Imitierte Status-Modul Antwort.

8.4.2 Erweiterungen der zustandsverändernden Funktionen

Die zustandsverändernden Funktionen müssen alle erweitert werden, da alle mit dem nun ersetzten Cluster-System interagieren. Jede muss deshalb die lexikografische Unterscheidung zwischen virtuellem und realem Knoten vornehmen (wie bereits erwähnt $\mathcal{O}(1)$). Anstatt die Kommandos an das Cluster-System zu senden, wird im virtuellen Fall mit dem Status-Server von *ClusterSim* kommuniziert. Der Status-Server erhält von *CHERUB* über den lokalen Socket eine Zeichenfolge, die das auszuführende Kommando und den Zielknoten angibt. Der Status-Server erkennt dieses Muster und setzt die erhaltenen Befehle um.
Auch hier entfällt die Netzwerkkommunikation, was die Ausführungszeit des Kommandos verringert. Insgesamt sind die Funktionen aber so klein, dass ihre Performance vergleichbar gering bleiben sollte.

8.5 Funktionsweise und Algorithmus

Der Simulator modelliert die Verwendung des Apache *MPM-prefork*-Moduls (siehe Abschnitt 2.2.2, Seite 20) auf einer Multicore-Maschine. Da das *prefork*-Modul auch in den Messungen dieser Arbeit eingesetzt wurde, bot es sich an, selbiges auch zu simulieren. Da die Simulation die einzelnen Ressourcen und deren Auslastung auf den Maschinen nicht simuliert, könnte auch das *worker*-Modul des Apache auf gleiche Weise modelliert werden.

ClusterSim verwendet als Scheduling-Verfahren nicht den im aktuellen Kernel 4.2.6 [1] verwendeten *Completely Fair Scheduler* (CFS) [165], sondern eine vereinfachte Form des $\mathcal{O}(1)$ Schedulers mit festem 100 ms Zeitscheibe/Quantum [2] und Round-Robin-Prozess Auswahl. Der $\mathcal{O}(1)$ Scheduler wird in Bovet und Cesatis *Understanding the LINUX KERNEL* [166] ab Seite 258 beschrieben. Er wurde mit Kernel 2.6 (18. Dezember 2003) eingeführt und in Kernel 2.6.23 (9. Okt. 2007) durch den CFS ersetzt.
Die Simulationsschritte sind eine Sekunde lang und Ereignisse innerhalb dieser Sekunde werden so genau wie möglich berechnet. Die wichtigsten Konfigurationen der VNodes sind die Anzahl der Rechenkerne, die Anzahl der verwendeten Apache-Worker und die Länge der BLQ. Die Rechenkerne geben dabei an, wieviel Rechenkapazität vorhanden ist. Hat ein VNode 4 Kerne, kann er innerhalb einer Sekunde Anfragen im Wert von 4 Sekunden bearbeiten. Dabei können Anfragen aber nicht parallel bearbeitet werden, wodurch pro Anfrage maximal 1 Sekunde Bearbeitung stattfinden kann. Die Anzahl der Apache-Worker gibt an, wie viele Anfragen gleichzeitig verarbeitet werden können. Die Apache-Worker werden möglichst gleichmäßig auf die Kerne verteilt. Dabei kann ein Apache-Worker nicht mehrere Kerne gleichzeitig verwenden. Die Länge der BLQ bestimmt, wie viele Anfragen vom System zwischengespeichert werden können, bevor sie verworfen werden. Der nachfolgende Algorithmus beschreibt das Verhalten des Simulators in jedem Simulationsschritt.

1. Modellierung der Ankunft der Anfragen und deren Verteilung:
   a) Füge alle Anfragen, die in der aktuellen Sekunde am System eintreffen, in die Cluster-Queue ein.

      Die Anzahl und Art der Anfragen wird mit Hilfe des vom Nutzer übergebenen Traces herausgefunden. Die Anfragen werden als 3-Tupel gespeichert und bestehen aus der Ankunftszeit, der Verarbeitungszeit und der URL selbst. Die Verarbeitungszeit wird durch die Anfrage-Bibliothek bereit gestellt.

   b) Überprüfe die Retransmission-Queue auf Anfragen, die erneut übertragen werden müssen (sogenannte Retransmissions). Füge Retransmissions, möglichst gleich verteilt, in die Cluster-Queue ein.

      Eine Retransmission ist ein 5-Tupel, bestehend aus den Elementen einer normalen Anfrage und zusätzlich der Zeit, die sie noch warten muss, bis sie erneut gesendet wird und der gesamten Zeit, die sie ursprünglich warten musste. Da die Wartezeit sich pro Retransmission erhöht, ist es notwendig, sich die aktuelle Wartezeit zu merken, um eine eventuelle neue Wartezeit zu berechnen. Das Verhalten beim Auftreten von Retransmissions wird unter anderem in RFC 2988 [167] beschrieben. Hier wird beschrieben, dass bei einem verlorengegangenen `SYN`-Paket die initiale Wartezeit 3 Sekunden beträgt. RFC 6298 [168] hat zwar RFC 2988 nach 11 Jahren im Jahre 2011 abgelöst, aber da das Test-Systen sich gemäß RFC 2988 verhält, wurde der Simulator ebenso entwickelt. Der Unterschied zum neuen RFC bezüglich der Wartezeit für verlorene `SYN`-Pakete ist außerdem gering. So schlägt der neue RFC eine initiale Wartezeit von 1 Sekunde (anstelle von 3 Sekunden) vor, da die Netzwerke sich im Laufe der Zeit stark

[1] Stand: 23.11.2015
[2] Die Zeit, die ein Prozess am Stück die CPU zum Rechnen zugewiesen bekommt.

genug entwickelt haben, so dass eine Wartezeit von 3 Sekunden nicht mehr notwendig erscheint, um die Last durch Retransmissions gering genug zu halten.

c) Verteile alle Anfragen aus der Cluster-Queue nach dem Round-Robin-Prinzip unter allen verfügbaren VNodes. Ein VNode gilt als verfügbar, wenn er angeschaltet und am RMS angemeldet ist (*ONLINE*). Die Anfrage wird an das Ende der BLQ des entsprechenden VNodes gehängt.

Man beachte, dass der Simulator, im Gegensatz zu *CHERUB*, nicht über einen Zustand *UNKNOWN* verfügt, da der Zustand dem Simulator immer bekannt ist. Er verfügt auch nicht über den Zustand *BUSY*, da es nicht in der Verantwortung des Simulators liegt, zu entscheiden, wie stark belastet ein VNode ist. Ansonsten entsprechen die *CHERUB* Definitionen von *DOWN*, *OFFLINE* und *ONLINE* denen von *ClusterSim*.

2. Modellierung der Anfrageverarbeitung pro VNode:

a) Überprüfe alle Anfragen in der BLQ, ob sie sich bereits im simulierten Webserver-Cache des VNodes befinden. Ist dies der Fall, wird die zu bearbeitende Zeit durch die zu bearbeitende gecachte Zeit ersetzt.

b) Verteile Anfragen in der BLQ nach dem First-In-First-Out (FIFO) Prinzip unter den freien Workern. Jeder Worker kann dabei nur eine Anfrage gleichzeitig bearbeiten.

c) Wenn die *early_blq_check* Option aktiv ist, überprüfe, ob es mehr Anfragen in der BLQ gibt, als die konfigurierte Länge zulässt. Ist dies der Fall, entferne alle überschüssigen Anfragen aus der BLQ. Falls eine Retransmission, wie in RFC 2988 beschrieben, keine Chance mehr hat, innerhalb der konfigurierten Timeout Zeit vom System angenommen zu werden, verwirf sie und protokolliere einen Timeout. Falls die Chance auf eine erfolgreiche Retransmission besteht, ergänze die Anfrage um die nötigen Retransmission-Attribute und hänge sie an die Retransmission-Queue.

d) Stelle die verfügbare Rechenkapazität aller Kerne wieder her (Rechenkapazität im Wert von einer Sekunde pro Kern). Verteile die Rechenkapazität gleichmäßig zwischen den dem Kern zugewiesenen Workern. Jede CPU besitzt nun n Worker, von denen jeder $1/n$ Sekunden Rechenkapazität besitzt. Markiere den ersten Worker in der Liste der Worker einer CPU als den aktiven Worker. Setze die verbrauchte Rechenzeit wieder auf null.

e) Solange wie die verbrauchte Rechenzeit weniger als einer Sekunde entspricht und es noch Worker mit Anfragen gibt, tue folgendes:

i. Ermittle die *Zeit zum nächsten Ereignis (ZZNE)*. Bei der ZZNE handelt es sich um den kleineren Wert von entweder der kleinsten Zeit, die die aktiven Worker noch rechnen dürfen, oder der kleinsten Zeit, die eine aktuell bearbeitete Anfrage noch zur Fertigstellung benötigt.

Die Zeit, die einem Worker zur Bearbeitung seiner Anfragen an einem Stück maximal zur Verfügung steht, entspricht 100 ms und wird als Quantum oder Zeitscheibe bezeichnet. Dieses Quantum entspricht dem Standardwert für konventionelle Prozesse im $\mathcal{O}(1)$ Scheduler (siehe [166], Tabelle 7-2, Seite 263). Im Umkehrschluss bedeutet dies, dass die ZZNE nie größer als 100 ms ist. Diese als Quantum bezeichnete Größe kann im Simulator auch umkonfiguriert werden.

ii. Reduziere die verbleibende Rechenzeit aller aktiven Worker um die ZZNE. Reduziere die verbleibende Rechenzeit aller Anfragen, die zur Zeit einem aktiven Worker zugeordnet sind, um die ZZNE. Erhöhe die verbrauchte Rechenzeit um die ZZNE.

iii. Wenn die verbleibende Rechenzeit einer Anfrage null erreicht, gilt die Anfrage als fertig bearbeitet. Protokolliere alle relevanten Daten der Anfrage (Dauer der Anfrage, handelt es sich um einen Timeout etc.). Wenn die Anfrage noch nicht gecached war, markiere die Anfrage als gecached und ändere die Bearbeitungszeit von wartenden gleichen Anfragen auf die gecachte Zeit. Entferne die Anfrage vom Worker und hole die nächste Anfrage, die in der BLQ wartet. Wenn die BLQ leer ist, wechsle zum nächsten Worker der CPU. Dies ist der nächste aktive Worker.

iv. Wenn die verbleibende Rechenzeit eines Workers null erreicht, ist entweder seine gesamte Rechenzeit verbraucht oder sein Quantum ist abgelaufen. Wechsle zum nächsten Worker der CPU, wenn nicht schon im vorherigen Schritt geschehen. Dies ist der nächste aktive Worker.

v. Wenn kein nächsten Worker auf der CPU existiert und es noch Worker mit Anfragen gibt und die Rechenzeit der CPU noch nicht null erreicht hat, verteile die verbleibende Rechenzeit der CPU gleichmäßig auf die verbleibenden Worker, die noch Anfragen haben. Der neue aktive Worker ist der erste in der Liste der Worker, der noch eine Anfrage zu bearbeiten hat.

f) Protokolliere die aktuelle Auslastung der Maschine und den dazugehörigen Energieverbrauch. Die Auslastung ergibt sich aus der verbrauchten Rechenzeit der einzelnen Kerne. Hat bei einer 4-Kern-Maschine z.B. nur 1 Kern seine Rechenzeit voll ausgeschöpft, war die Maschine zu 25 % ausgelastet (siehe auch Abschnitt 8.6, Seite 124).

g) Wenn die *early_blq_check* Option nicht aktiviert ist, überprüfe die BLQ jetzt entsprechend Schritt 2c.

Abbildung 8.6 bis 8.15 stellt die Verarbeitung innerhalb eines VNodes dar. Gezeigt wird die Verarbeitung von Anfragen innerhalb einer Sekunde. Aus didaktischen Gründen wurde ein kleines Beispiel mit einem einzelnen Kern gewählt, dem vier Worker (rote Rechtecke) zugeteilt sind. Es wird gezeigt, wie sechs Anfragen (blaue Rechtecke) nach und nach verarbeitet werden. In jedem Schritt wird die *Zeit zum nächsten Ereignis* (graues Rechteck) neu bestimmt. Die Länge der Anfragen wurde für das Beispiel sehr groß gewählt. Die meisten Anfragen liegen in der Realität mit ihrer Bearbeitungszeit im kleinen Millisekundenbereich und sind damit um ein Vielfaches kürzer. Deren Ende stellt daher, im Gegensatz zu den im Beispiel verwendeten Anfragen, sehr häufig die ZZNE dar. Da es in dem Beispiel vier Worker, aber sechs Anfragen gibt, werden die restlichen zwei Anfragen in die BLQ eingereiht. Die BLQ war vorher leer und wird in den Abbildungen nicht extra gezeigt.

Die Länge der graphischen Repräsentation der Anfragen entspricht der verbleibenden Bearbeitungszeit. Dies gilt nicht, wenn sie länger ist als die dem Worker zugewiesene Rechenzeit.

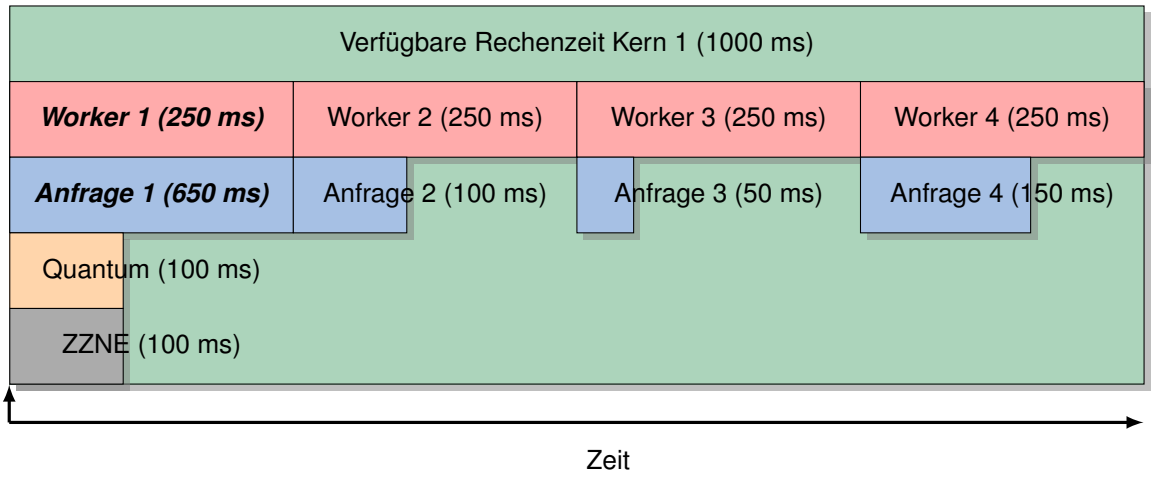


Abbildung 8.6: Schritt 1: Initiale Situation. Die BLQ hält zwei Anfragen (nicht zu sehen) und die restlichen 4 Anfragen wurden auf die 4 Worker aufgeteilt. Die ZZNE entspricht dem Quantum von Worker 1, da die Dauer von Anfrage 1 größer ist als das Quantum von Worker 1.

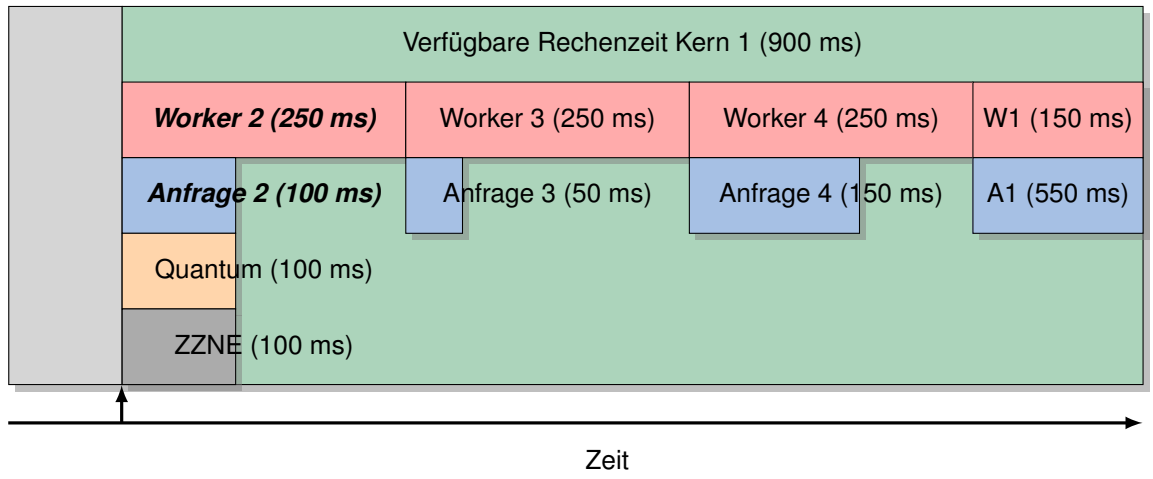


Abbildung 8.7: Schritt 2: Die ersten 100 ms sind abgearbeitet. Da das Quantum von Worker 1 aufgebraucht ist, wird dieser an das Ende der Scheduling-Queue verschoben. Der neue aktive Worker ist jetzt Worker 2. Die ZZNE entspricht nun dem Quantum von Worker 2 und gleichzeitig dem Rest von Anfrage 2.

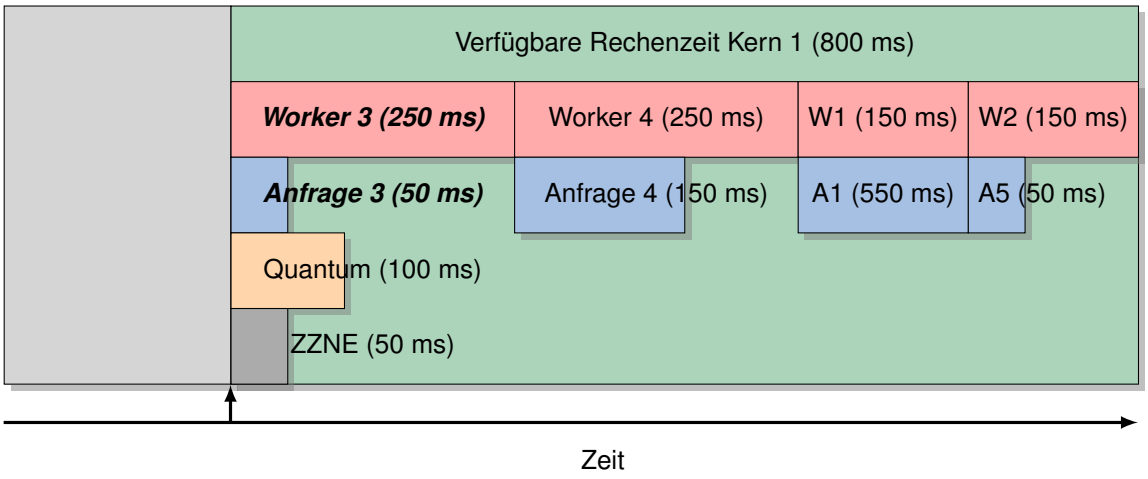


Abbildung 8.8: Schritt 3: Nach weiteren 100 ms hat Worker 2 Anfrage 2 fertig bearbeitet. Daher wurde ihm nun Anfrage 5 aus der BLQ zugewiesen. Da das Quantum von Worker 2 jetzt ebenfalls abgelaufen ist, wurde dieser hinter Worker 1 eingereiht. Die ZZNE entspricht jetzt der Restdauer von Anfrage 3 (50 ms), da diese kürzer ist als das Quantum von Worker 3.

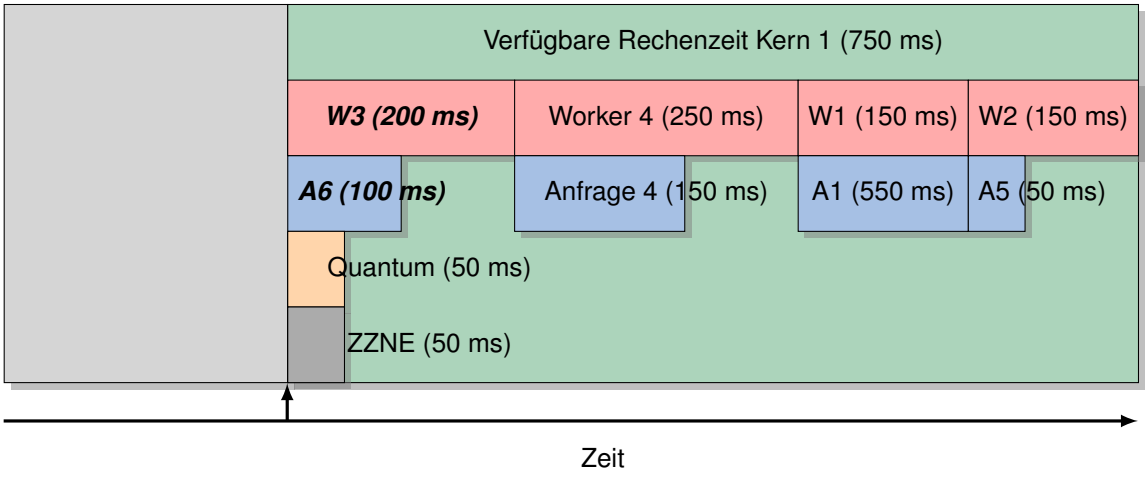


Abbildung 8.9: Schritt 4: Nach 50 ms hat Worker 3 Anfrage 3 fertig bearbeitet. Ihm wird deshalb Anfrage 6 aus der BLQ zugewiesen. Da das Quantum des Workers noch nicht abgelaufen ist, bleibt dieser der aktive Worker und rechnet an Anfrage 6 weiter. Die ZZNE entspricht jetzt dem Ende des Quantums von Worker 3 (50 ms), da dieses kürzer ist als dessen zu bearbeitende Anfrage.

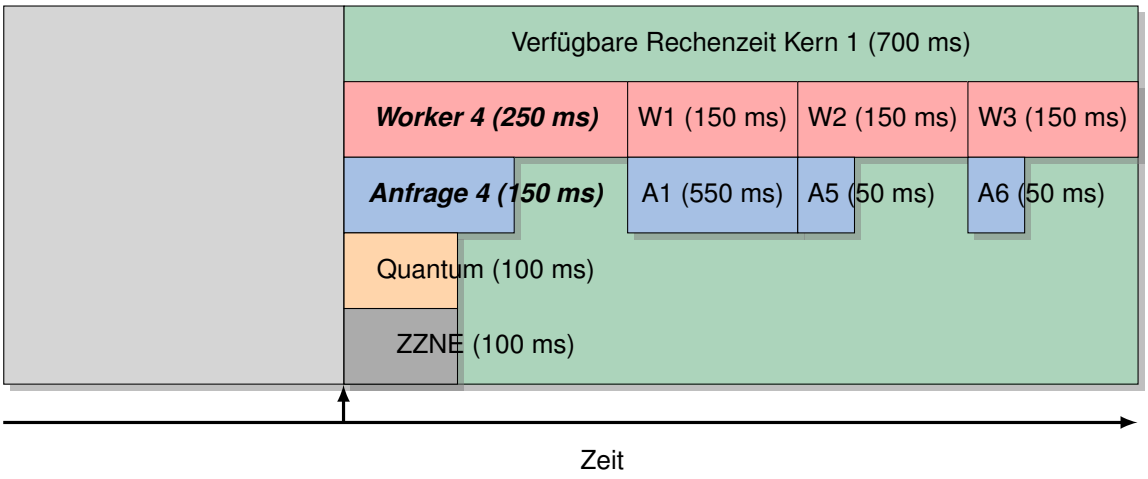


Abbildung 8.10: Schritt 5: Nach weiteren 50 ms (insgesamt sind 300 ms vergangen) ist das Quantum von Worker 3 abgelaufen, welcher an das Ende der Worker gehängt wird. Der nächste aktive Worker ist Worker 4. Die ZZNE entspricht wieder dem Quantum des momentan aktiven Workers.

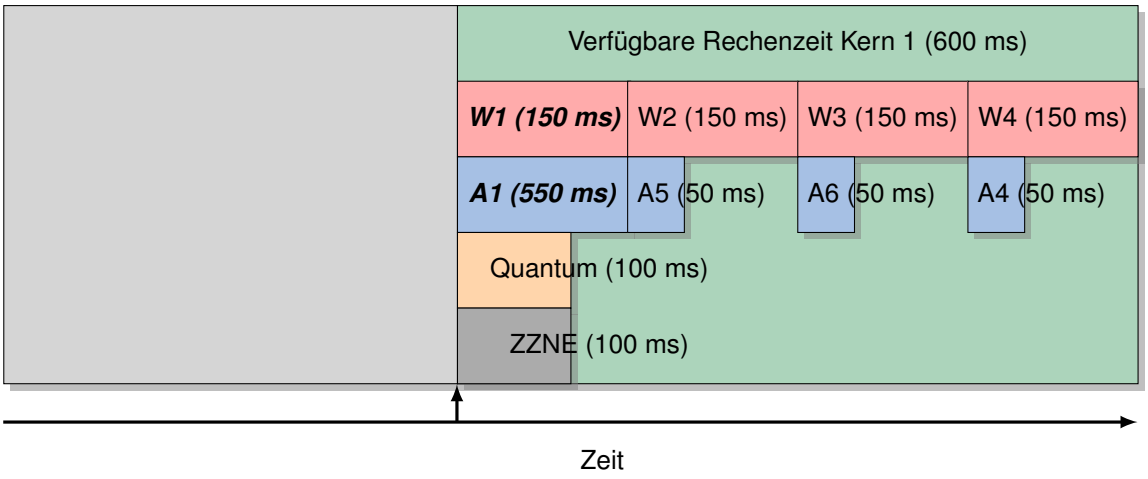


Abbildung 8.11: Schritt 6: Nach weiteren 100 ms ist das Quantum von Worker 4 abgelaufen, welcher daher an das Ende der Worker gehängt wird. Da alle Worker einmal an der Reihe waren, wird die verbleibende Rechenzeit neu unter den Workern verteilt. Da alle Worker noch Anfragen zu bearbeiten haben, bekommen auch alle wieder gleich viel Zeit zugewiesen (jeweils 150 ms). Der neue aktive Worker ist nun Worker 1, welcher an Anfrage 1 weiter rechnet.

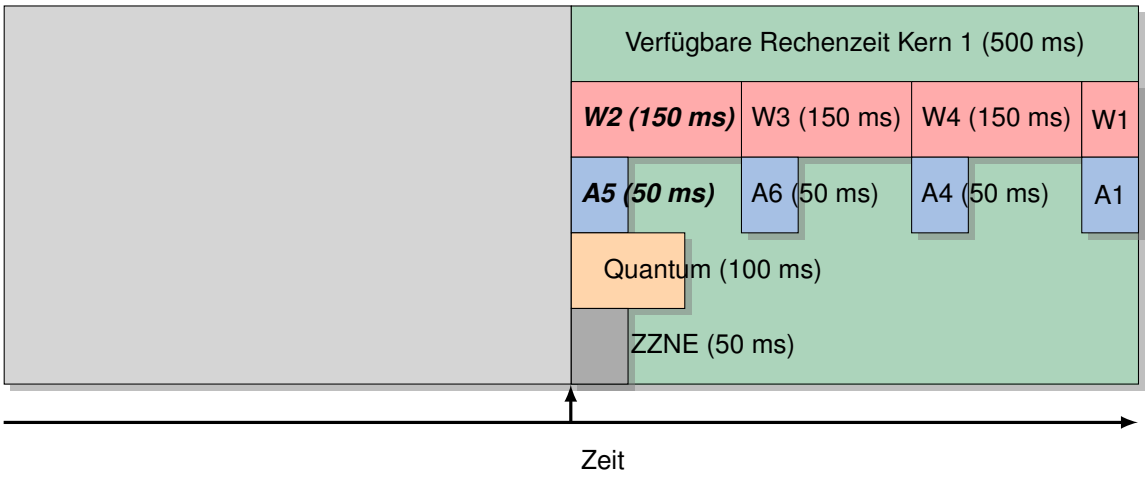


Abbildung 8.12: Schritt 7: Nach 100 ms läuft das Quantum von Worker 1 ab und er wird an das Ende der Worker gehängt. Der neue aktive Worker ist Worker 2 und die ZZNE entspricht jetzt der restlichen Bearbeitungszeit von Anfrage 5, welche kürzer ist als das Quantum von Worker 2.

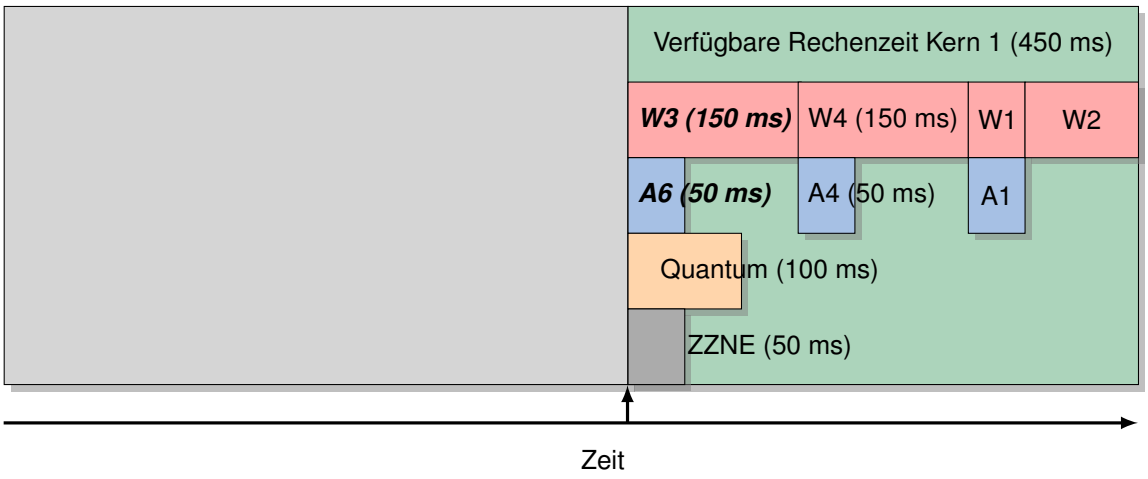


Abbildung 8.13: Schritt 8: Nach 50 ms ist Anfrage 2 fertig bearbeitet. Da die BLQ leer ist, kann Worker 2 keine weiteren Anfragen mehr erhalten und der aktive Worker wechselt zu Worker 3, obwohl das Quantum noch nicht abgelaufen ist. Mit Worker 3 und Anfrage 6 passiert nun das Gleiche wie in Schritt 7 mit Worker 2 und Anfrage 5.

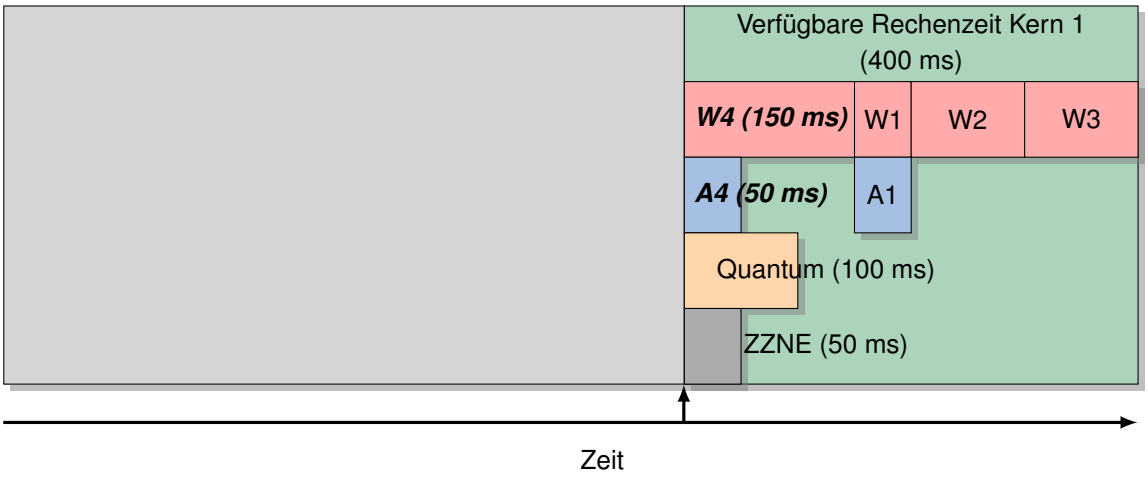


Abbildung 8.14: Schritt 9: Hier geschieht das Gleiche wie in Schritt 7 und 8, nur für Worker 4 und Anfrage 4.

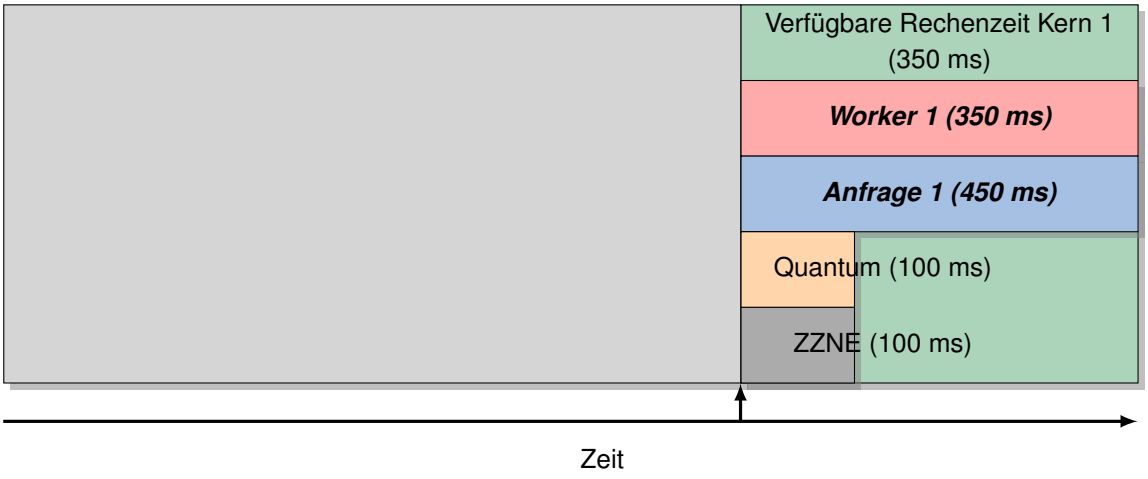


Abbildung 8.15: Schritt 10: Nachdem erneut alle Worker ein Mal an der Reihe waren, wird wieder die verbleibende Rechenzeit neu verteilt. Da nur noch Worker 1 eine Anfrage zu bearbeiten hat, wird ihm die gesamte verbleibende Rechenzeit zugewiesen. Worker 1 wird somit der aktive Worker und arbeitet Anfrage 1 weiter ab. Da es sich um den letzten Worker mit Anfrage handelt, wird diese, unabhängig vom Quantum, bis zum Ende der Sekunde weiter bearbeitet. Am Ende der simulierten Sekunde hat Anfrage 1 nun noch eine Bearbeitungszeit von 100 ms und Worker 1 würde die Bearbeitung im ersten Schritt der nächsten Sekunde beenden können.
Auf die Darstellung der restlichen Schritte wird auf Grund ihrer Einfachheit verzichtet.

8.6 Energieverbrauch und Auslastung in ClusterSim

Der in Schritt 2f des Algorithmus von *ClusterSim* beschriebene Vorgang zur Ermittlung des Energieverbrauchs soll hier noch einmal genauer beschrieben werden. In der für die Messungen verwendeten Version 1.06.05 werden Energieverbrauchsprofile für 4-, 8-, 16- und 32-Kern-Maschinen unterstützt. Dabei ist es sehr einfach, ein neues Profil zu ergänzen, wenn dies notwendig ist. Diese Energieverbrauchsprofile sagen aus, bei welcher prozentualen Auslastung die Maschinen wieviel Watt benötigen. Die Datenbasis für das Profil stammt vom SPECpower_ssj2008 Benchmark [169]. Die erhobenen Daten sind frei verfügbar und geben Aufschluss über den Verbrauch von aktuell 502 Systemen bei einer Auslastung von 0 % - 100 % in 10 %-Schritten. Der Benchmark lastet dabei alle verfügbaren Kerne gleichmäßig aus. Für die Profile im Simulator wurden nur Systeme verwendet, die aus einer einzelnen Maschine bestehen und ein gültiges Testdatum besitzen. Aus den übrigen 409 Systemen wurde erneut nach Anzahl von Kernen gefiltert. Für die finalen Profile wurde der Median aller Maschinen (mit jeweiliger Kernanzahl) pro 10 % Schritte gewählt. Für die später verwendeten simulierten 4-Kern-Maschinen ergibt sich das Profil aus dem Median von 67 Systemen, die alle aus einzelnen 4-Kern-Maschinen bestehen. Für die Profile für 8, 16 und 32 Kerne ergeben sich die Profile aus dem Median aus je 87, 88 bzw. 24 Systemen. Die verwendeten Daten wurden ab 2007 erhoben (3 Systeme davor) und die meisten Systeme wurden im Jahr 2012 registriert (26,9 %). Da die Daten in 10 % Schritten vorliegen, verwendet der Simulator für Zwischenwerte eine lineare Approximation. Es konnten allerdings für startende und herunterfahrende Systeme keine Daten gefunden werden. In der Bachelor-Arbeit von Herrn Pioch [147] wurde ein System bei beiden Vorgängen untersucht. Verglichen mit dem Verbrauch unter Volllast, hat das gemessene System (Acer Aspire XC600, Intel Core i5-3330 3,0GHz (4 Kerne), 8 GB DDR3, 500 GB SATA HDD) beim Herunterfahren 8 % und beim Starten des Systems 30 % Energie verbraucht. Diese Werte erscheinen plausibel und aufgrund mangelnder Daten in [169], bezüglich Starten und Herunterfahren, wurden diese für *ClusterSim* übernommen. Abbildung 8.16 zeigt die linear-approximierten Energieverbrauchsprofile der unterstützen Maschinen.

8.7 Grenzen

Da ein Simulator nur ein Modell der Realität ist, hat auch *ClusterSim* gewisse Grenzen. Die größten Limitierungen sind hier genannt:

Scheduler: Wie bereits angesprochen, wurde der CFS nicht implementiert, sondern eine Round-Robin Verteilung mit 100 ms Zeitscheiben (Quantum), die eher dem $\mathcal{O}(1)$ Scheduler ähnelt. Die Unterschiede in der Prozessverwaltung (Red-Black-Tree bzw. Run-Queue) sind vor allem für die Laufzeit des Schedulers wichtig. Da im Simulator nur sehr wenige Prozesse zu verwalten sind, kann die Laufzeit vernachlässigt werden. Ein Unterschied, der für die Simulationsergebnisse von Bedeutung ist, ist die Granularität. Der CFS arbeitet im Nanosekundenbereich. Beispiel 13 soll dies verdeutlichen.

Beispiel 13 Angenommen zwei Worker auf einer CPU haben beide eine Anfrage mit einer Rechendauer von 100 ms zu bearbeiten. Da das Quantum für Worker 1 100 ms beträgt, wird Worker 1 nach 100 ms mit Anfrage 1 fertig. Danach ist Worker 2 an der Reihe und bearbeitet Anfrage 2. Diese wird nach weiteren 100 ms fertig bearbeitet. Die Anfragen haben nun eine Bearbeitungszeit

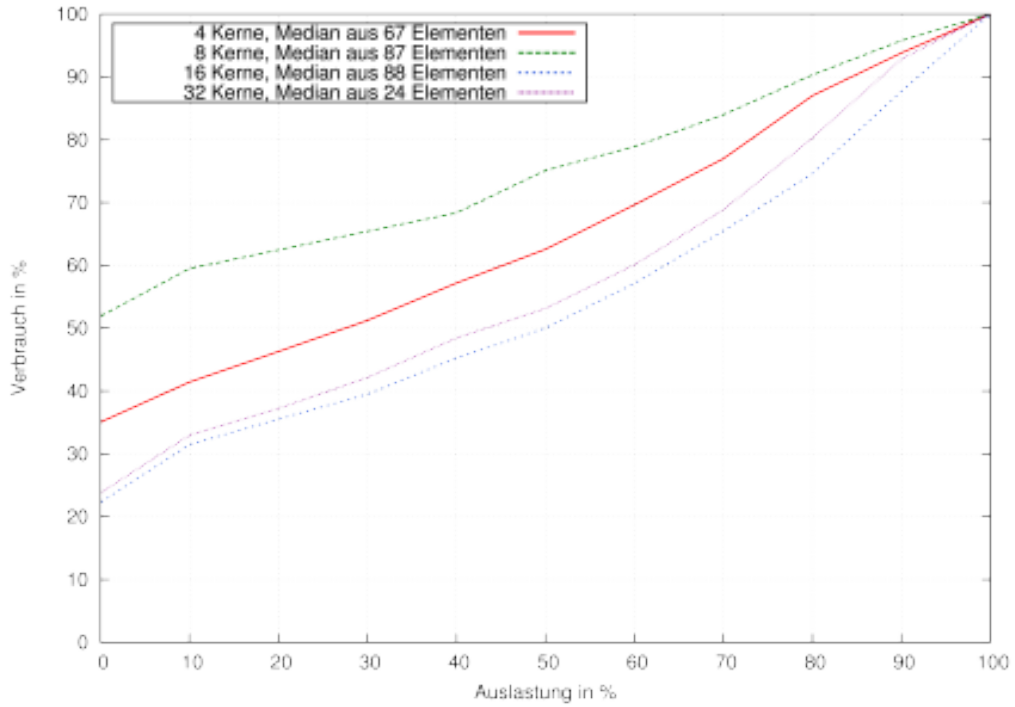


Abbildung 8.16: Approximierte Verbrauchsprofile.

von 100 ms und 200 ms (100 ms Rechenzeit + 100 ms Wartezeit) gehabt. Dies entspricht einer durchschnittlichen Bearbeitungszeit von 150 ms.
Bei einer feineren Granularität mit z.B. einem Quantum von 1 ms hätte unter denselben Bedingungen die erste Anfrage 199 ms benötigt (100 ms Rechenzeit + 99 ms Wartezeit) und die zweite Anfrage 200 ms (100 ms Rechenzeit + 100 ms Wartezeit). Dies entspricht einer durchschnittlichen Bearbeitungszeit von 199,5 ms. □

Das Beispiel macht außerdem klar, dass sich eine sehr feine Granularität zwar positiv auf die Fairness auswirkt, aber eher negativ auf die durchschnittliche Bearbeitungzeit, da insgesamt mehr gewartet werden muss. Durch diesen Umstand ist zu erwarten, dass die simulierten Ergebnisse für die durchschnittliche Anfragedauer eher besser abschneiden werden als in einem realen System, welches den CFS als Scheduler verwendet.

BLQ Überprüfung und Eintreffen von Anfragen: Da *ClusterSim* seine Berechnungen jede Sekunde durchführt, kommen Anfragen, die innerhalb einer Sekunde am System eintreffen, immer gebündelt zur vollen Sekunde am System an. Dadurch ergibt sich die Frage, wie mit ihnen verfahren werden soll, wenn die BLQ voll ist oder durch ihr Eintreffen voll wird. Es wurden zwei verschiedene Approximationen implementiert, zwischen denen über die *early_blq_check* Option gewählt werden kann. Ist die Option aktiv, wird direkt nach der Verteilung der Anfragen auf die Maschinen und nachdem alle Worker auf den Maschinen Anfragen bekommen haben überprüft, ob die BLQ übergelaufen ist. Liegt ein Überlauf vor, werden überschüssige Anfragen verworfen

bzw. in die Retransmission-Queue getan (siehe Schritt 2c). Dieses Verhalten kann zu mehr Timeouts als in der Realität führen. Ist die Option nicht aktiv, wird die Überprüfung erst am Ende der Routine (siehe Schritt 2g) vorgenommen und innerhalb der Sekunde wird die BLQ behandelt, als hätte sie keine Beschränkung in ihrer Länge. Dieses Verhalten kann zu weniger Timeouts als in der Realität führen. Obwohl ein gleichzeitiges Ankommen von Anfragen in diesem Maße nicht der Realität entspricht, sind gebündelte Ankünfte nicht unüblich. Moderne Browser öffnen typischerweise bis zu sechs Verbindungen gleichzeitig, um eine Webseite schneller laden zu können (siehe [170], Kapitel 11).

Systemrauschen: Auf jedem System laufen noch weiter Prozesse neben dem Webserver-Dienst im Hintergrund. Meist sind dies systemeigene Dienste wie beispielsweise ein SSH-Daemon oder ein Update-Service. Solche Dienste benötigen ebenfalls Ressourcen, die dann für die Anfragebearbeitung fehlen. Auf die Modellierung eines solchen Systemrauschens wurde verzichtet. Eine einfache Approximation könnte implementiert werden, indem jedem Kern anstelle der vollen Sekunde nur die Sekunde, abzüglich des erwarteten Rauschens, an Rechenzeit pro Sekunde zugewiesen würde. Eine etwas aufwendigere, dafür realistischere Implementation könnte neben den Workern noch Dummy-Prozesse einführen, welche die anderen Dienste repräsentieren. Diesen würde, wie den Workern auch, Rechenzeit zugewiesen. Die zugewiesene Rechenzeit stünde dann nicht mehr für die Verarbeitung von Anfragen zur Verfügung.
Das Ignorieren des Systemrauschens sorgt tendenziell für bessere Ergebnisse in der Simulation. Eine gewisse Menge an Systemrauschen ist allerdings bereits implizit in den Zeiten der Anfrage-Bibliothek enthalten, weshalb schwer abzuschätzen ist, wie stark der Unterschied hier wirklich ist.

Konkurrierende Ressourcenzugriffe: Obwohl ein wiederholter Zugriff auf eine Ressource typischerweise durch Cachingeffekte schneller ist, ist der parallele Zugriff auf Ressourcen, bedingt durch Flaschenhälse, meist langsamer. Wenn z.B. eine Datenbank mehr Anfragen bekommt, als sie gleichzeitig bedienen kann, kommt es zu Problemen. Außerdem würde in dem Falle die CPU, auf der die Datenbank läuft, stark beansprucht werden, was auch zu zusätzlichem Systemrauschen führen würde. Ein weiterer Flaschenhals könnte ebenfalls eine Festplatte sein, die mehrere Anfragen abarbeiten muss. Dies ist, physikalisch bedingt, nur teilweise, abhängig von der Anzahl der Leseköpfe und der Lokalität der Daten, möglich.
Solche parallelen Effekte wurden nicht modelliert. Insbesondere das Datenbank-Beispiel könnte, abhängig vom eingespielten Trace, in den Simulationen vorkommen. Diese Vereinfachung kann zu einer besseren Bearbeitungszeit in der Simulation führen.

8.8 Validierung

Nun soll überprüft werden, wie genau der Simulator, trotz der erwähnten Einschränkungen und Vereinfachungen, arbeitet. Dafür werden Messungen aus dem vorhandenen Testbett mit Simulationen verglichen, die dieselben Bedingungen widerspiegeln sollen. Es werden dabei vor allem einfache, überschaubare Randfälle betrachtet werden. Die Referenzmessungen wurden mit dem selbst entwickelten Trace-Replay-Tool SPECTRE (siehe Abschnitt 2.2.6, Seite 32) durchgeführt.

Setup, Parameter und Metriken: SPECTRE ist für die Messungen auf der Maschine Leibniz installiert und greift via der IB1 auf die IB5 zu (siehe Tabelle 5.1, Seite 65). Um das Verhalten der BLQ einfacher beobachten zu können, wurde eine sehr kurze BLQ mit einer Länge von 20 Plätzen gewählt. Die Metriken, die für die Validierung verwendet werden, sind die *QoS* (gleichbedeutend mit der Anzahl der Timeouts, genauere Erläuterung siehe Abschnitt 5.6, Seite 76) und die *Anfragedauer* (AD) in Millisekunden. Die *Anfragedauer* umfasst dabei sämtliche Netzwerkzeiten (Verbindungsaufbau etc.) sowie die Verweilzeit der Anfrage. Da die Maschinen im Simulator potenziell auch länger mit einer Anfrage beschäftigt sein können, als dies die *FRT* angibt, wird hier die *Anfragedauer* verwendet und nicht, wie in den Messungen, die *FRT*. Da die *Anfragedauer* allerdings potenziell mehr Zeit umfasst, ist ein direkter Vergleich zwischen *FRT* und *Anfragedauer* schwierig.
Die Messungen werden jeweils 10 Mal wiederholt und der Median der Metriken wird angegeben. Die Simulationen werden jeweils 3 Mal durchgeführt. Es kann durch Rundungsfehlern zu minimal abweichenden Ergebnissen kommen. Während der Simulationen ist allerdings keine Abweichung aufgetreten.

1. Experiment: 1 Knoten, 4 Kerne, 1 Worker pro Kern, konstante Last Dieses Experiment ist sehr einfach gehalten und soll zunächst einer grundlegenden Funktionsüberprüfung dienen. Es werden zwei verschiedene Traces als Arbeitslast verwendet, ein leichtgewichtiger und ein schwergewichtiger. Beide sind 10 Minuten lang und bestehen pro Sekunde aus vier identischen Anfragen. Insgesamt beinhalten beide Traces 2400 Anfragen. Der Unterschied der Traces liegt in der Anfrage, die wiederholt gesendet wird. Der leichtgewichtige Trace verwendet als Anfrage eine mit 0,8 ms sehr schnell zu bearbeitende Anfrage und der schwergewichtige Trace nutzt eine Anfrage mit einer Dauer von 3,6 Sekunden. Es werden ausschließlich die gecachten Geschwindigkeiten verwendet. Für die Messungen wurde dazu die Anfrage vor Messbeginn bereits an das System gesendet und der Simulator verfügt über eine Option (*hotcache*), die bei Aktivierung ausschließlich die gecachten Werte nutzt. Um immer eine gleiche Messumgebung zu gewährleisten, wurde die IB5 zwischen jeder Messung neu gestartet.
Für den leichtgewichtigen Trace wird erwartet, dass alle Anfragen erfolgreich bearbeitet werden. Die Erwartungen für den schwergewichtigen Trace sind, dass nur die ersten 4 Anfragen erfolgreich bearbeitet werden. Da die ersten Anfragen bereits alle Worker 3,6 Sekunden beschäftigen, müssen die 4 nachfolgenden bereits 2,6 Sekunden warten, bis sie bearbeitet werden. Mit ihrer eigenen Bearbeitungszeit und der Wartezeit ergibt sich bereits eine Verweilzeit von 6,2 Sekunden, was einem Timeout entspricht (Timeout gilt ab einer Zeit von größer 5 Sekunden). Alle weiteren Anfragen kommen ebenfalls erst in die BLQ bzw. werden sogar ganz verworfen, da diese nach kurzer Zeit überläuft.
Die Ergebnisse der Messungen spiegeln genau die Erwartungen wider und sind in Tabelle 8.1 zu sehen. Die Abweichung bei der *Anfragedauer* ist durch Messschwankungen zu erklären und immer noch sehr nah an den simulierten Ergebnissen. Es wurde nicht zwischen den beiden verschiedenen Arten der BLQ-Überprüfung unterschieden, da die Ergebnisse hier gleich sind.

2. Experiment: 1 Knoten, 4 Kerne, 1 Worker pro Kern, ansteigende Last Ach dieses Experiment dient noch der Überprüfung der Grundfunktionalität, allerdings in einem erweiterten Szenario.

Art der Anfrage	Metrik	Messung	Simulation
leichtgewichtig	QoS	100 %	100 %
	AD	0,92 ms	0,802 ms
schwergewichtig	QoS	0,166 %	0,166 %
	AD	3,582 s	3,578 s

Tabelle 8.1: Ergebnisse von Experiment 1.

Die Konfiguration der Knoten ist wie in Experiment 1, mit Ausnahme der *hotcache* Funktion. Für dieses Experiment würde die *hotcache* Funktion keinen Unterschied machen, da die real genutzte Anfrage keine Vorteile aus dem Caching ziehen kann und die gecachte und ungecachte Bearbeitungszeit somit identisch ist. Die Länge der verwendeten Traces sind wieder 10 Minuten. Die Menge der Anfragen wird aber in 5 verschiedenen Stufen verändert. Die Stufen sind 4, 8, 12, 16 und 20 Anfragen pro Sekunde. Die Anfragen sind immer identisch und haben eine Länge von 0,36 Sekunden.

Um valide Referenzwerte für dieses Experiment zu ermitteln, wurde ein PHP-Skript geschrieben, welches ein sogenanntes *aktives Warten* (engl. busy waiting) für 0,36 Sekunden durchführt. Das Skript fragt dafür seine Rechenzeit so lange ab, bis sie 0,36 Sekunden erreicht hat. Die Verwendung eines `sleep` Befehls für 0,36 Sekunden liefert hier nicht das gewünschte Resultat, da durch ein `sleep` die CPU nicht blockiert bzw. verwendet wird. Wird die CPU nicht verwendet, müssen die Anfragen auch nicht gegenseitig aufeinander warten um CPU-Zeit zu bekommen und die Messungen würden nicht die reale Begebenheit widerspiegeln.

In diesem Experiment ist vor allem die Stufe mit 12 Anfragen pro Sekunde interessant. Hier wird es zu dem Effekt kommen, dass die Last der Anfragen größer ist als die Leistung der Maschine. Bei 12 Anfragen pro Sekunde muss jeder Kern/Worker 3 Anfragen innerhalb einer Sekunde bearbeiten. Da die 3 Anfragen (a 0,36 Sekunden) aber zusammen 1,08 Sekunden benötigen, werden 0,08 Sekunden Rechenzeit mit in die nächste Sekunde genommen. Die Anfragen in der Sekunde darauf müssen also 0,08 Sekunden warten, bis sie bearbeitet werden können. Dieser Effekt verstärkt sich kaskadierend, bis die BLQ irgendwann überläuft bzw. Anfragen in der BLQ so lange warten müssen, dass sie trotz eines Platzes in der BLQ einem Timeout entsprechen. Die gewählte BLQ-Länge von 20 Plätzen sorgt allerdings dafür, dass Anfragen, die es in die BLQ schaffen, auch schnell genug bearbeitet werden. Da immer 4 Anfragen gleichzeitig bearbeitet werden, ist die schlechteste Situation für eine neue Anfrage (ausgenommen vom Verworfenwerden), wenn sie in der BLQ Platz 17-20 erhält. Bei dieser Platzierung muss sie abwarten, bis 5 Anfragen vor ihr fertig bearbeitet werden. Davon haben 4 die volle Bearbeitungszeit (0,36 Sekunden), da sie vor ihr in der BLQ sind und 1 Anfrage hat noch eine Bearbeitungszeit von 0,04 Sekunden * x (wobei $x \in \{1..9\}$), da diese noch vom Worker fertig bearbeitet werden muss. Im schlechtesten Fall muss eine Anfrage auf Platz 17-20 somit 5 * 0,36 Sekunden = 1,8 Sekunden warten und wird daher kein Timeout werden.

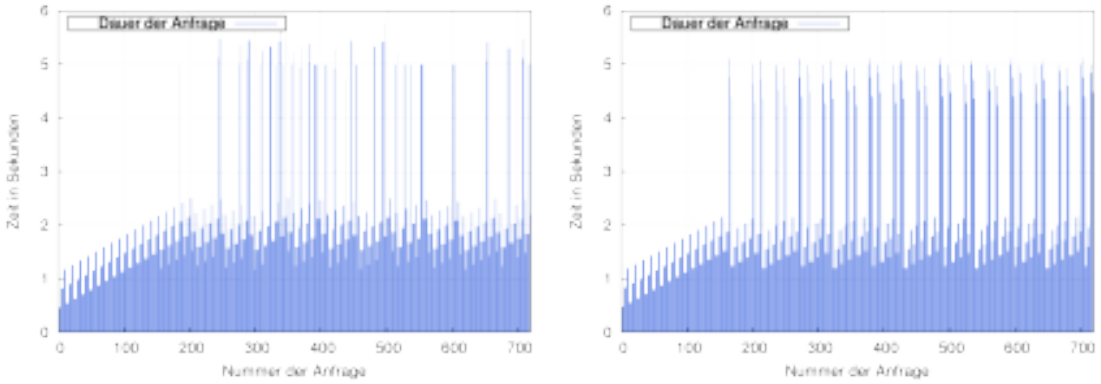
Tabelle 8.2 fasst die Ergebnisse von Experiment 2 zusammen. Wie erwartet entstehen bei weniger als 12 Anfragen pro Sekunde keine Probleme. Die Simulation spiegelt hier die Ergebnisse sehr akkurat wider. Ab 12 Anfragen pro Sekunde kommt es zu Timeouts und damit zu einer verringerten *QoS*. Da die BLQ bei 12 und mehr Anfragen pro Sekunde regelmäßig überläuft, macht sich nun

Anf/s	Metrik	Messung	Simulation (frühe BLQ Überprüfung)	Simulation (späte BLQ Überprüfung)
4	QoS	100 %	100 %	100 %
	AD	0,36 ms	0,36 ms	0,36 ms
8	QoS	100 %	100 %	100 %
	AD	0,64 ms	0,6 ms	0,6 ms
12	QoS	92,34 %	92,92 %	93,01 %
	AD	1,88 ms	1,92 ms	2,88 ms
16	QoS	69,27 %	69,88 %	69,98 %
	AD	1,97 ms	2,16 ms	3,16 ms
20	QoS	55,36 %	55,90 %	56,00 %
	AD	1,97 ms	2,16 ms	3,16 ms

Tabelle 8.2: Ergebnisse des 2. Experiments.

auch der Unterschied zwischen einer frühen oder späten Überprüfung der BLQ bemerkbar. Die Ergebnisse für eine späte Überprüfung sind eine Sekunde langsamer als die mit der frühen Überprüfung. Dies ist naheliegend, da bei der späten Überprüfung die aktuelle Sekunde noch komplett zur Berechnung verwendet werden kann, bevor überschüssige Anfragen verworfen werden. Insgesamt stimmen die Ergebnisse bei Verwendung der späten Überprüfung der BLQ daher nicht besonders gut mit denen der Messung überein. Die frühe Überprüfung dafür aber um so besser. Da die frühe Überprüfung der BLQ erheblich genauer ist, werden die nachfolgenden Simulationen nur noch mit dieser Methode durchgeführt (Experiment 3 und 4).
Abbildung 8.17 zeigt dies noch einmal anhand des Anfragemusters. Zu sehen ist hier die *Anfragedauer* jeder einzelnen Anfrage in der Reihenfolge, wie die Anfragen am System eingetroffen sind. Gezeigt ist hier das 12-Anfragen-pro-Sekunde-Szenario. Dargestellt sind nur die 720 Anfragen der ersten von zehn Minuten. Ab etwa Anfrage Nummer 200 (Abbildung 8.17a, Messung) bzw. 160 (Abbildung 8.17a, Simulation) ist die BLQ des Knotens bzw. des VNodes gefüllt. Da die Anfragen in der Simulation etwas länger dauern als in der Messung, ist die BLQ in der Simulation auch schneller gefüllt. Bevor die BLQ voll ist, sieht man, wie die Anfragen immer zu viert nahezu identisch schnell fertig bearbeitet werden (da 4 CPUs gleichzeitig an ihnen arbeiten). 1 Sekunde im Verlauf des Traces entspricht in der Abbildung 12 blauen Strichen (bzw. 3 gebündelte blauen „Treppen", bestehend aus je 4 Anfragen). Man kann sehr gut erkennen, wie sich von Sekunde zu Sekunde die Überlast langsam weiter aufbaut. Ist die BLQ dann gefüllt, kommt es zu Retransmissions, welche es entweder in die BLQ schaffen und dann auch fertig bearbeitet werden (blaue Striche größer 5 Sekunden) oder die es nicht schaffen und damit zu einem Timeout werden (blaue Striche exakt 5 Sekunden groß). Auch wenn das Muster der Simulation etwas regelmäßiger wirkt, ähnelt es der Messung in einem sehr hohen Maße.

3. Experiment: 1 Knoten, 4 Kerne, 1-2 Worker pro Kern, BLQ 1-80 Nachdem die ersten beiden Experimente vor allem die Korrektheit der Grundfunktionalität feststellen sollten, wird sich Experiment 3 dem Effekt der überlaufenden BLQ widmen. Wie bereits erklärt und gezeigt, kommt es ab einer gewissen Länge der BLQ dazu, dass, wenn diese voll ist, eingereihte Anfragen so lange war-



(a) Anfragedauer einzelner Anfragen in der Messung.

(b) Anfragedauer einzelner Anfragen in der Simulation.

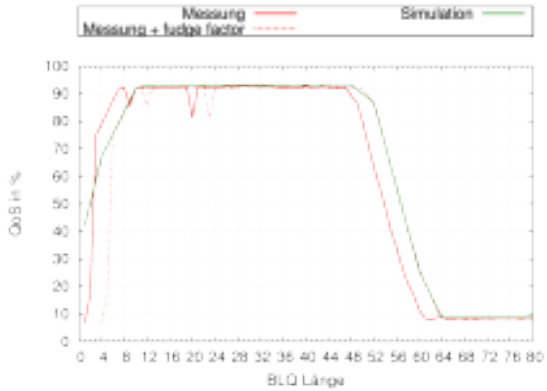
Abbildung 8.17: Vergleich der Anfragedauer für 12 Anfragen pro Sekunde zwischen Messung und Simulation.

ten müssen, dass sie zwar bearbeitet werden, aber durch ihre lange Wartezeit zu Timeouts werden. Der Scheitelpunkt, ab welcher Länge dies geschieht und wie gut der Simulator dies widerspiegelt, soll jetzt ermittelt werden.

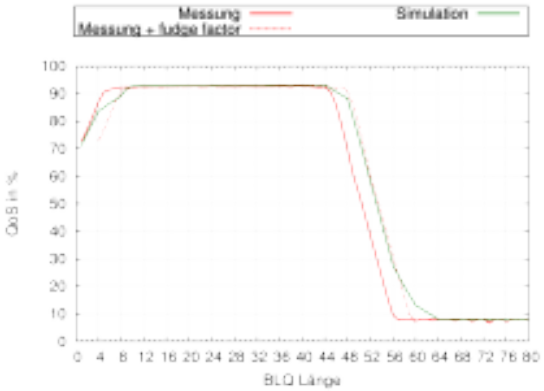
Die Bedingungen für dieses Experiment gleichen dem der vorherigen. Diesmal ist allerdings die Arbeitslast konstant mit 12 Anfragen pro Sekunde. Alle Anfragen haben wieder eine Bearbeitungszeit von 0,36 Sekunden. Die Bearbeitungszeit beeinflusst ebenfalls den Scheitelpunkt. Sie verlagert ihn nach hinten oder vorne. Aus diesem Grund ist der primäre Faktor für dieses Experiment die Länge der BLQ. Sie wird von 1 bis 80 variiert. Um einen Eindruck für das Verhalten von mehreren Workern pro Kern zu bekommen, wird die gesamte Messung auch für 2 Worker pro Kern durchgeführt. Auf Grund der Dauer ($2 * 80 * 10$ Minuten) dieser Messreihe wird keine Wiederholung durchgeführt. Ausreißer sind deshalb nicht auszuschließen.

Abbildung 8.18 stellt die Ergebnisse des Experiments dar. Die Messungen verhalten sich im Vergleich zur Simulation so, als wäre ihre BLQ exakt 3 Plätze kürzer. Der Effekt ist in allen vier Darstellungen zu beobachten und zeigt, dass der Unterschied nicht mit der Anzahl der Worker in Verbindung steht. Der Unterschied scheint vom sogenannten *fudge factor* (engl. Schummel-Faktor) zu kommen. Die Längenangabe des Nutzers wird, ohne dessen Wissen, um diesen Faktor erweitert (siehe [73], Kapitel 4.5). Der Grund für diesen Faktor ist laut [73] nicht mehr bekannt. Der *fudge factor* variiert zwischen verschiedenen Systemen und [73] gibt ihn für `Linux 2.4.7` mit *+3* an. Diese Verschiebung stimmt genau mit den gemessenen Werten überein. Dies spricht dafür, dass in der verwendeten Implementierung (`2.6.18-308.el5`) noch derselbe Faktor Verwendung findet. Wenn in der Messung die BLQ mit z.B. 60 angegeben ist, ist sie also in Wirklichkeit bei 63. Da die Simulation diesen Faktor nicht berücksichtigt, entsteht die beobachtete Verschiebung. In den Abbildungen ist deshalb die Messung noch einmal um drei verschoben abgebildet.

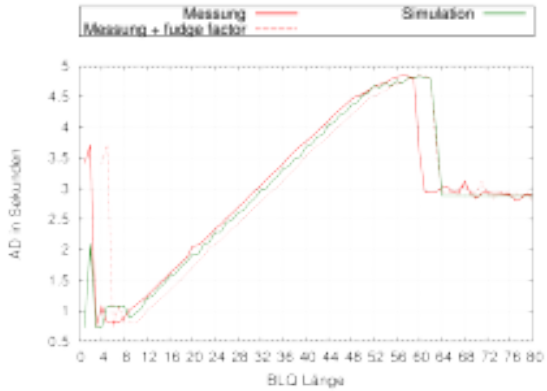
Die Abbildungen 8.18a und 8.18b zeigen das Verhalten der *QoS* für 4 und 8 Worker. Bis auf den Randfall, in dem die BLQ kürzer ist als die Anzahl der Worker, sind die Profile unter Berücksichtigung des *fudge factors* nahezu deckungsgleich. Dasselbe gilt auch für die *Anfragedauer*, zu sehen in Abbildung 8.18c und 8.18d. Kleinere Abweichungen gibt es noch im Bereich um die BLQ Län-



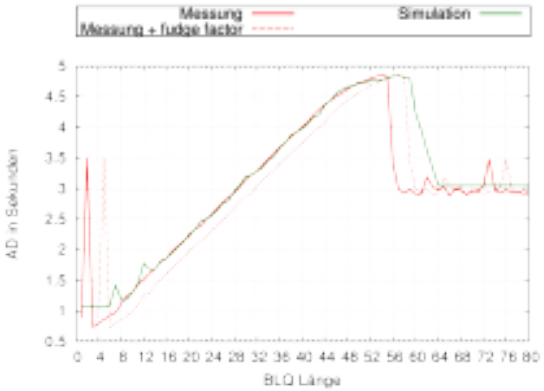
(a) Vergleich der QoS zwischen Simulation und Messung (4 Worker).



(b) Vergleich der QoS zwischen Simulation und Messung (8 Worker).



(c) Vergleich der Anfragedauer zwischen Simulation und Messung (4 Worker).

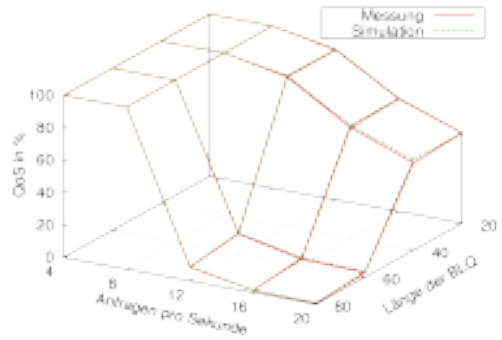


(d) Vergleich der Anfragedauer zwischen Simulation und Messung (8 Worker).

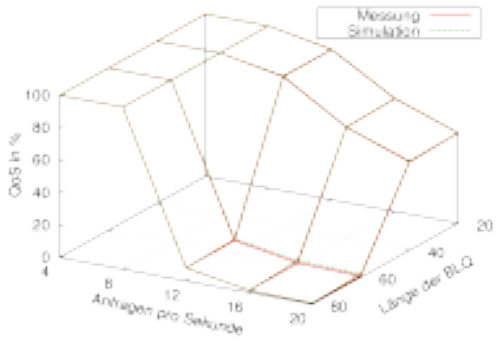
Abbildung 8.18: Unterschied zwischen Simulation und Messung in Abhängigkeit der Länge der BLQ und der Verwendung von 4 bzw. 8 Workern.

ge von 60. Dieser Bereich ist schwierig deckungsgleich zu bekommen, da hier der Scheitelpunkt der BLQ liegt und schon kleinere Abweichungen der *Anfragedauer* (zwischen Simulation und Messung) die Ergebnisse stärker beeinflussen können.
Die Rohdaten dieses Experiments befinden sich in Anhang A: Messergebnisse, Seite 160, Tabelle A.2 bis A.5.

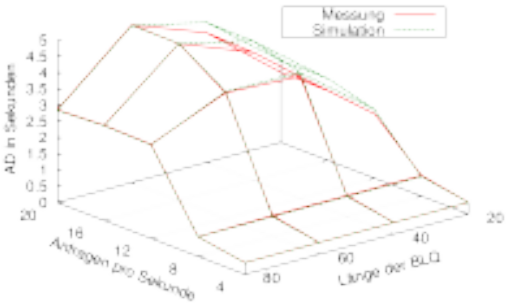
4. Experiment: 1 Knoten, 4 Kerne, 1-2 Worker pro Kern, BLQ 20-80, ansteigende Last Das letzte Experiment zur Validierung ist noch einmal ein Kreuzprodukt aus den bisherigen Vergleichen. Es werden sowohl die Worker zwischen 4 und 8 variiert als auch die Länge der BLQ (20, 40, 60, 80) und die Arbeitslast (4, 8, 12, 16, 20 Anfragen pro Sekunde). Um den beobachteten *fudge factor* auszugleichen, wird die Simulation jeweils mit einer Länge von $x+3$ durchgeführt. Da die Simulation aber, wie bereits gezeigt, ein korrektes Verhalten bezüglich eines BLQ Überlaufes aufweist, wird der *fudge factor* nicht in die Implementation des Simulators übernommen. Diese Simulation wird zusätzlich auch mit der Option zur späten Überprüfung der BLQ durch-



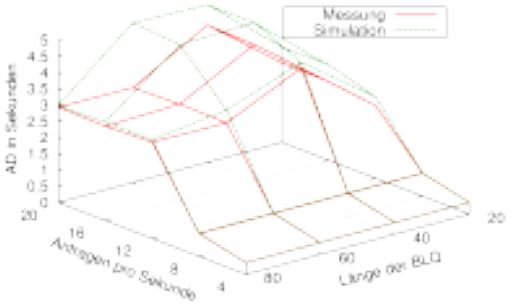
(a) Vergleich des QoS zwischen Messung und Simulation (4 Worker).



(b) Vergleich des QoS zwischen Messung und Simulation (8 Worker).



(c) Vergleich der Anfragedauer zwischen Messung und Simulation (4 Worker).



(d) Vergleich der Anfragedauer zwischen Messung und Simulation (8 Worker).

Abbildung 8.19: Vergleich zwischen Simulation und Messung in Abhängigkeit der Länge der BLQ, der Anfragen pro Sekunde und der Anzahl der Worker.

geführt, aber wie bereits erwartet, waren die Ergebnisse wesentlich ungenauer, weshalb auf die Präsentation der Ergebnisse an dieser Stelle verzichtet wird.
Die Abbildung 8.19 fasst die Ergebnisse für die Nutzung der frühen BLQ-Überprüfung zusammen. In Abbildung 8.19a und 8.19b sind die unterschiedlichen *QoS*-Ergebnisse zu sehen. Diese sind nahezu nicht vorhanden. Die stärkste Abweichung der *QoS* entsteht bei der Verwendung von 4 Workern, 20 Anfragen pro Sekunde sowie einer BLQ von 40 und beträgt 2,24 %. Abbildung 8.19c und 8.19d zeigen die unterschiedliche *Anfragedauer*. Hier gibt es an zwei Stellen mit einer BLQ von 60 besonders große Abweichungen, die beide denselben Ursprung haben. Die beiden Abweichungen entstehen bei der Verwendung von 8 Workern, einer BLQ von 60 sowie 16 bzw. 20 Anfragen pro Sekunde. Die Abweichung beträgt hier 1,83 und 2 Sekunden. Der Grund für die sehr großen Abweichungen ist die sehr geringe *QoS* und vereinzelte Anfragen, die in der Simulation noch fristgerecht bearbeitet werden können, in den Messungen aber nicht mehr. Gezeigt sei dies anhand der 2 Sekunden Abweichung. Die Anzahl der Anfragen, die noch bearbeitet werden konnten, ist hier mit 0,84 % (Messung) sehr gering. Auch die Simulation hat eine sehr geringe

QoS mit 2,21 %. Die Differenz von 1,37 % zwischen Messung und Simulation kommt von Anfragen, die in der Simulation auch mit gefüllter BLQ noch eine *Anfragedauer* knapp unterhalb der 5-Sekunden-Grenze besitzen. Die realen Maschinen schaffen diese wenigen Anfragen nicht mehr zu bearbeiten. Da die Messungen und Simulationen 10 Minuten dauern und die BLQ nach etwa 12 bzw. 6 Sekunden bereits gefüllt ist (für 16 bzw. 20 Anfragen pro Sekunde), überwiegen die Anfragen mit einer sehr hohen *Anfragedauer*. Der Median der *Anfragedauer* wird demzufolge aus einer dieser Anfragen gewählt und liegt damit knapp unter 5 Sekunden (erkennbar in 8.19d). In den Messungen können nach den 12 bzw. 6 Sekunden keine weiteren Anfragen mehr fristgerecht bearbeitet werden, wodurch der Median aus diesen ersten Anfragen besteht und viel geringer ist. Da diese Abweichungen nur in Fällen auftreten, in denen die *QoS* ohnehin in einem inakzeptablen Zustand ist, macht eine selbst so große Abweichung kaum einen Unterschied. Wichtiger ist aber, dass diese Abweichung nicht durch unplausible Vorgänge entsteht. Würden die Anfragen in der Simulation auch nur geringfügig länger dauern, würden auch die knappen Anfragen nicht mehr rechtzeitig bearbeitet werden und die Ergebnisse wären wieder nahezu identisch. Ursachen hierfür können die bereits erwähnten Limitierungen der Simulation sein (z.B. keine Simulation von Systemrauschen).
Die Rohdaten dieses Experiments befinden sich in Anhang A: Messergebnisse, Seite 160, Tabelle A.6 und A.7.

8.9 Zusammenfassung

Zusammenfassend kann gesagt werden, dass die Simulation für das angestrebte Szenario glaubwürdige Ergebnisse liefern wird. Die Architektur erfüllt die Anforderungen an einen solchen Simulator und die vier durchgeführten Experimente konnten zeigen, dass das Verhalten dem einer realen Maschine weitestgehend entspricht. Größere Abweichungen zwischen Messung und Simulation kamen dabei nur in pathologischen Fällen vor und auch nur bei der *Anfragedauer*. Es konnte allerdings begründet werden, dass die Abweichungen nicht im Widerspruch zum Verhalten einer echten Maschine stehen.

9 Skalierung der Energiesparstrategie

Nachdem in Kapitel 8 *ClusterSim* als ideales Werkzeug für eine Simulationsstudie mit *CHERUB* vorgestellt wurde, kann sich dieses Kapitel nun dem Nachweis von Anforderung 2 widmen (Funktionalität des Algorithmus in größeren Setups, siehe Kapitel 7, Seite 96).
Dafür werden die mit *ClusterSim* und *CHERUB* durchgeführten Simulationen vorgestellt. Es handelt sich dabei um eine Kreuzproduktstudie, in der verschiedene Parameterkonfigurationen von *CHERUB* untersucht werden und deren Eignung festgestellt wird.
Die Studie wurden mit der *ClusterSim* Version 1.06.05 auf der IB1 Maschine (siehe Tabelle 5.1, Seite 65) durchgeführt.

9.1 Metriken

Für die Bewertung der Ergebnisse werden fünf verschiedene Metriken Verwendung finden. Zwei um die gesamte Qualität des Dienstes zu bewerten, zwei um die Kosten zu bewerten und eine gewichtete Summe aus allen vier Metriken.

1. **Service Level Agreement (*SLA*) / Quality of Service (*QoS*):** Wie schon in den anderen Messungen dieser Arbeit wird die primäre Bewertung der gesamten Qualität des Dienstes über das *SLA* bzw. die *QoS* stattfinden. Dabei gilt wie zuvor, dass die *SLA*/*QoS* das Verhältnis zwischen Timeouts und erfolgreich bearbeiteten Anfragen darstellt. Es beträgt 100 %, wenn alle Anfragen erfolgreich bearbeitet wurden. Ein Timeout wird gewertet, wenn eine Anfrage länger als 5 Sekunden für eine Antwort benötigt.

2. **Anfragedauer (AD):** Ist die sekundäre Metrik für die gesamte Qualität des Dienstes. Sie ergibt sich aus dem Median der *Anfragedauer* aller erfolgreich bearbeiteten Anfragen. Diese Metrik wird als sekundär angesehen, da sie von der Anzahl der erfolgreich bearbeiteten Anfragen abhängt. Eine höhere Anzahl von bearbeiteten Anfragen kann, wie im 4. Experiment von Abschnitt 8.8 gesehen und erklärt, zu einer höheren Verweilzeit führen. Es wird angenommen, dass eine größere Anzahl an erfolgreich bearbeiteten Anfragen wenigen, schnelleren Anfragen vorzuziehen ist. Wie schon beschrieben, umfasst die *Anfragedauer* sämtliche Netzwerkzeiten, Verweilzeit und die Bearbeitungszeit.

3. **Energieverbrauch (EV) / Eingesparte Energie (EE):** Der Energieverbrauch in Wattstunden (Wh) gibt den Gesamtverbrauch des Clusters an. An ihm können die Kosten abgelesen werden. Die eingesparte Energie ist der korrespondierende Prozentwert der an Energie gegenüber dem Referenzszenario gespart wurde. Der Energieverbrauch/eingesparte Energie stellt die primäre Metrik für die Bewertung der Kosten dar. Wie *ClusterSim* den Energieverbrauch berechnet, wurde in Abschnitt 8.6 (Seite 124) erklärt.

4. **Anzahl der physikalischen Zustandswechsel (PZW):** Als physikalische Zustandsänderung wird das An- oder Abschalten einer Maschine definiert. Dieser Vorgang kann auf Dauer die Lebenszeit der eingesetzten Hardware reduzieren. Da solch ein Ausfall schwierig vorherzusagen ist und von vielen Faktoren abhängt, handelt es sich nicht um direkte Kosten, die sofort entstehen, und sind daher nur als sekundärer Kostenfaktor zu betrachten.

   Beide auf Kosten bezogenen Metriken verursachen nicht nur finanzielle Kosten, sondern auch Kosten in Form von Umweltschäden. Bei einem Ausfall von Hardware muss neue Hardware gebaut und verschifft werden. Beide Prozesse verbrauchen viel Energie. Für den Bau werden zusätzlich Ressourcen wie Metalle der seltenen Erden etc. verwendet, die wiederum gewonnen werden müssen (häufig unter sehr umweltschädlichen Bedingungen [171, 172]). Auch die Energiegewinnung sorgt durch z.B. CO^2-Ausstoß, Fracking, nukleare Abfälle etc. für teils immense Schäden. Diese zusätzlich zu bedenkenden Kosten sind hier aber leider nicht quantifizierbar und unterliegen ebenfalls Schwankungen (z.B. wenn erneuerbare Energien Verwendung finden).

5. **Score:** Die Score stellen eine gewichtete Summe der vier vorgestellten Metriken dar. Sie dient dazu, unter einer gewählten Strategie die beste Konfiguration zu finden. Definiert werden an dieser Stelle vier verschiedene Strategien. Man kann anhand der Gewichtung aber beliebige Strategien betrachten. Die gewählten Strategien haben bestimmte Fokusse, die wie folgt aussehen:

   a) Hochleistungsbetrieb. Diese Strategie verfolgt das Ziel, möglichst viele Anfragen möglichst schnell zu beantworten und priorisiert daher die Metriken Qualität des Dienstes und Anfragedauer.

   b) Preiswerter Betrieb. Diese Strategie versucht die Kosten des Betriebs so gering wie möglich zu halten und priorisiert daher die verbrauchte Energie und die physikalischen Zustandswechsel.

   c) Ausgeglichener Betrieb. Diese Strategie soll eine gute Qualität des Dienstes bei möglichst geringen Kosten gewährleisten. Dabei wird bei der Gewichtung auf die Qualität des Dienstes und die Eingesparte Energie gesetzt.

   d) Alte Hardware. Diese Strategie soll zeigen, dass auch etwas speziellere Fokusse mit dem Score-System gut bewertet werden können. Diese Strategie ist für Betreiber von sehr alter, unzuverlässiger Hardware gedacht. Hier besteht eher die Gefahr, dass bei häufigem An- und Abschalten die Hardware Schaden nimmt. Hier wird die Anzahl der *PZW*s besonders stark gewichtet.

   Die Gewichtungen zu den Strategien ist in Tabelle 9.1 angegeben und spiegelt die beschriebene Priorisierung wider.

Die durchgeführte Kreuzproduktstudie besteht aus 80 Simulationen. Da die vier einzelnen Metriken alle einen anderen Wertebereich besitzen, wird die Metrik *Score* eine ranglistenbasierte Normalisierung innerhalb der 80 ermittelten Ergebnisse durchführen. Dafür wird für jede Metrik m eine Rangliste aufgestellt, bei der dem besten Ergebnis $Rang_m = 0$ zugewiesen wird und dem schlechtesten Ergebnis $Rang_m = 79$. Pro Metrik bekommt eine Konfiguration $(80 - Rang_m)$ Punkte. Die erzielten Gesamtpunkte entsprechen dem *Score* und ergeben sich aus der Summe der

Strategie	Gewichtung			
	QoS	EV	AD	PZW
Hochleistungsbetrieb	0,5	0,1	0,3	0,1
Preiswerter Betrieb	0,1	0,5	0,1	0,3
Ausgeglichener Betrieb	0,4	0,4	0,1	0,1
Alte Hardware	0,1	0,1	0,1	0,7

Tabelle 9.1: Verwendete Gewichtung in Abhängigkeit der gewählten Strategie.

System	Parameter	Wert
ClusterSim	Anzahl VNodes	100
	Initial angeschaltete VNodes	3
	CPUs und Worker eines VNodes	4 CPUs / 4 Worker
	Länge der TCP-Backlog-Queue	80
CHERUB	Intervall	15 s
	wait_for_sign_off	1 min
	wait_for_shutdown	1 min
	wait_for_boot	siehe Faktoren
	wait_for_register	0 min
	Überlast-Schwellwert	25 Anf

Tabelle 9.2: Verwendete Parameter für *ClusterSim* und *CHERUB*.

gewichteten Punkte, die pro Metrik erzielt wurden. Formal ist dies ausgedrückt in Formel 9.2.

$$Metriken = \{QoS, AD, EV, PZW\} \tag{9.1}$$

$$Score = \sum_{m \in Metriken} (80 - Rang_m) * Gewicht_m \tag{9.2}$$

9.2 Parameter

Die von *ClusterSim* verwendeten Parameter, um das virtuelle Cluster und die virtuellen Knoten zu konfigurieren, sowie die Parameter von *CHERUB* sind in Tabelle 9.2 aufgeführt.

Parameter, ClusterSim: Wie eingangs in Kapitel 7 beschrieben, ist bei einem größeren Cluster von einer komplexeren Architektur auszugehen. Wenn die Strategie von *CHERUB* für 100 VNodes funktioniert, gibt es außerdem keinen Grund für die Annahme, dass die verwendete Strategie nicht auch bei mehr als 100 Maschinen funktioniert. Daher wurde für den Skalierbarkeitsnachweis ein 100 VNodes großes Setup gewählt.
Initial sind 3 VNodes angeschaltet. Dabei handelt es sich um die ideale Anzahl von VNodes, um die initiale *Niedrig-Last-Phase* der Arbeitslast (beschrieben in Abschnitt 9.4, Seite 138) zu bewältigen.

System	Faktor	Werte
ClusterSim	Zeit zum Hochfahren eines VNodes	5, 30, 60, 120, 180 s
CHERUB	sequential_shutdown	automatisch (0), 1
	wait_for_boot	0, 1 min
	boot_duration	15, 30, 60, 120, 180 s
	Backup	0, 5, 10, 25 %

Tabelle 9.3: Verwendete Faktoren für *ClusterSim* und *CHERUB*.

4 Worker und 4 CPUs werden verwendet, da die Ergebnisse der Validierung am genausten waren, wenn die Anzahl der Worker gleich der Anzahl der CPUs war. Zusätzlich wurden in den bisherigen Messungen ebenfalls Maschinen mit 4 Kernen verwendet.
Da der Großteil der Anfragen in der Arbeitslast wesentlich schneller ist als die in der Validierung verwendeten Anfragen kann die BLQ ohne Weiteres auf 80 Plätze gestellt werden. Der beobachtete Effekt aus den Validierungsexperimenten, bezüglich einer zu langen Wartezeit bei einer zu langen BLQ, ist nicht zu erwarten. Hier könnte eventuell sogar mit einem größeren Parameter gearbeitet werden.

Parameter, CHERUB: Die wait_for_x-Parameter wurden hier identisch zu den Einstellungen der vorherigen Messungen gelassen. Die Ausnahme ist wait_for_boot und wird bei den Faktoren erläutert. Die gewählte Einstellung ist damit eher der Performance des Systems zuträglich und erhöht den Energieverbrauch.
Auch das Intervall hat sich mit 15 Sekunden nicht geändert und ist weiterhin geeignet, um eine schnelle Reaktionszeit zu gewährleisten.
Verändert hat sich der Überlast-Schwellwert. Er ist nun mit 25 Anfragen so hoch gewählt, wie die Performance des Systems ist. Der Parameter besitzt somit kein implizites Backup mehr und riskiert damit, der *QoS* zu schaden.

9.3 Faktoren

Die von *ClusterSim* und *CHERUB* verwendeten Faktoren, welche über die Simulationen hinweg variieren, sind in Tabelle 9.3 zu finden.

Faktoren, ClusterSim: Natürlich kann die Zeit zum Hochfahren einer Maschine in der Realität nicht einfach verändert werden, wie ein einstellbarer Wert in einer Konfigurationsdatei. Aber in diesem Experiment soll ermittelt werden, wie stark sich die Dauer des Bootens auf die Effektivität des Algorithmus auswirkt. Dadurch kann zum einen die beste Konfiguration für eine bestimmte Bootzeit gewählt werden, und zum anderen kann es eventuell die Kaufentscheidung bei einer neuen Hardwarebeschaffung beeinflussen. Die Bootzeit von 5 Sekunden soll hier die Verwendung von STR-fähigen Maschinen (siehe Abschnitt 2.1.1, Seite 9) darstellen.

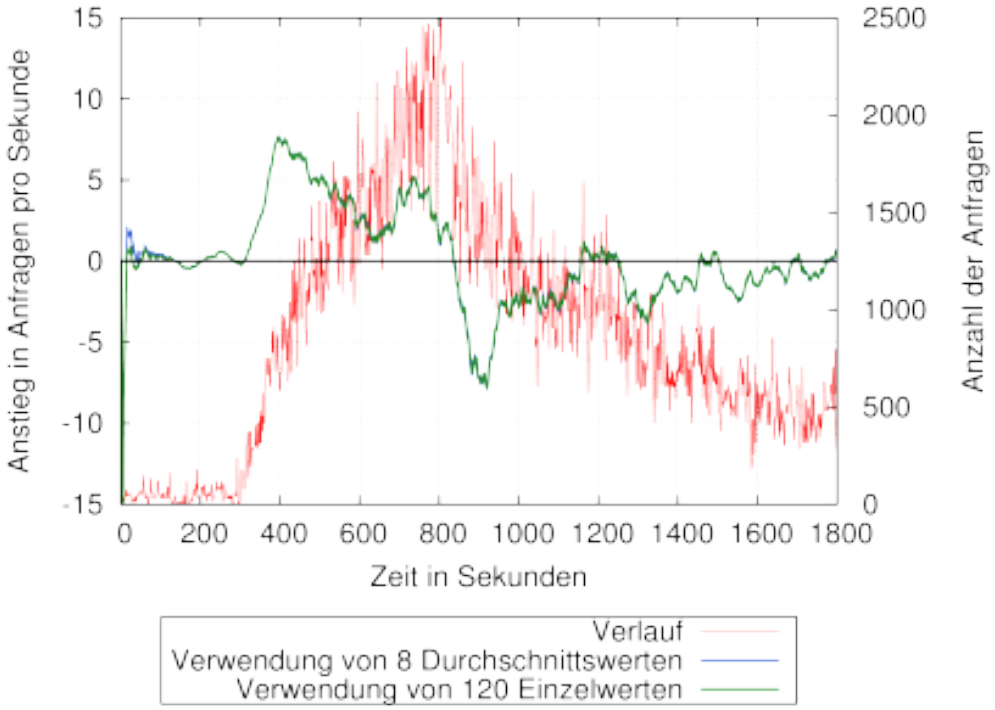
Faktoren, CHERUB: Da nun 100 Maschinen und nicht mehr nur 2 Maschinen verwaltet werden, soll auch verglichen werden, was passiert, wenn *CHERUB* von alleine ermittelt, wie viele Kno-

ten gleichzeitig abgeschaltet werden können. Dieses Vorgehen ist sehr aggressiv, im Gegensatz zu einem sehr langsamen sequenziellen Abschalten. Es birgt allerdings das Potenzial, viel Energie einzusparen. Ist `sequential_shutdown` weiterhin auf 1 gestellt, kann es mitunter sehr lange dauern, bis eine große Zahl überschüssiger Maschinen wieder abgeschaltet wurde. Im Gegenzug können aber erneut auftretende Lastspitzen einfacher abgefangen werden.
Der wait_for_boot-Faktor wurde vor allem verwendet, um das Problem des *State Flappings* (siehe Abschnitt 4.2, Seite 59) zu reduzieren. Der Vergleich mit 0 Minuten (kein Warten) soll feststellen, ob dies wirklich notwendig ist, oder ob die Lastvorhersage dies bereits ausreichend verhindert.
Die boot_duration muss *CHERUB* weiterhin kennen, um rechtzeitig Maschinen zu starten. Der Wert entspricht deshalb immer dem Wert der Bootzeit, die bei den VNodes eingestellt ist. An dieser Stelle wird das Limit erneut ausgereizt, da kein ε (siehe Abschnitt 6.3.1, Seite 86) auf die Bootzeit geschlagen wird. Hier wird wieder *QoS* für Energieeinsparungen riskiert. Einzig bei 5 Sekunden wird für *CHERUB* ein größerer Wert eingestellt. Dieser entspricht dem genutzten Intervall. Eine Vorhersage auf Basis eines Zeitfensters, welches kürzer ist als das Intervall, in dem Daten über die Vorhersage gesammelt werden, könnte zu Seiteneffekten führen und wird deshalb vermieden.
Bei dem Backup-Faktor handelt es sich um eine Neuerung von *CHERUB*. Er ermöglicht zu konfigurieren, wieviel Prozent des gesamten Clusters *CHERUB* als Backup immer zusätzlich zu den aktuell verwendeten Maschinen angeschaltet lassen soll. Dieser Faktor tauscht verbrauchte Energie direkt gegen *QoS*.

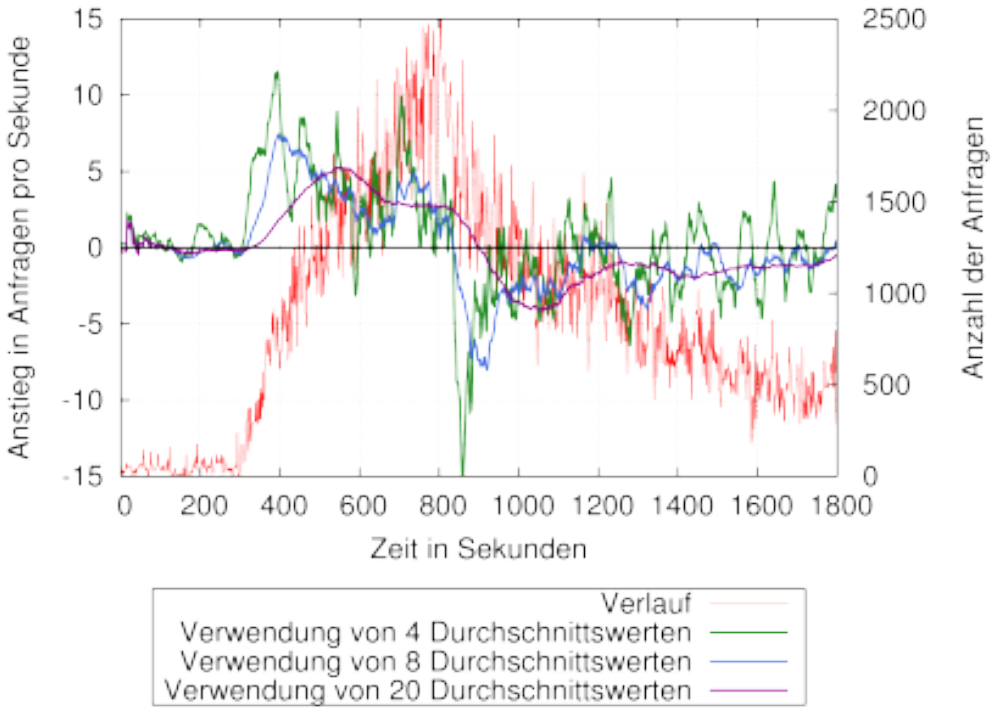
Art der Studie: Es wird eine volle Kreuzproduktstudie durchgeführt. Da der Faktor *Zeit zum Hochfahren eines VNodes* und *boot_duration* bis auf die erwähnte Ausnahme gleich ist, ergeben sich 80 Simulationen, bei denen *CHERUB* das virtuelle Cluster von *ClusterSim* verwaltet.

9.4 Arbeitslast und Wahl der Referenz

Bei der Arbeitslast handelt es sich im Grunde um denselben Trace, der schon in Abschnitt 5.4 (Seite 69) beschrieben wurde. Da jetzt allerdings ein Cluster mit 100 Maschinen verwendet wird, wurde der Trace entsprechend vergrößert. Das Profil ist dasselbe, aber der höchste Punkt des Traces umfasst nun 2.500 Anfragen, nicht mehr nur 50. Außerdem wurden Anfragen, deren ungecachte Dauer größer als 5 Sekunden waren, entfernt, da sich diese durch die Vergrößerung des Traces gehäuft hätten. Daher ergibt sich eine Gesamtanzahl von 1.586.160 Anfragen, von denen 395.441 Stück einmalig sind. Abbildung 9.1 zeigt noch einmal die Arbeitslast.
Zusätzlich ist in Abbildung 9.1 abgetragen, welche Steigung *CHERUB* in welcher Sekunde errechnet, um seine Vorhersage zu treffen. Wie in Abschnitt 7.3 (Seite 100) diskutiert, wurde für eine performantere Funktion auch der Ansatz verworfen, in dem `tcpdump` zur Lasterkennung genutzt wurde. Durch diese Veränderung basiert die errechnete lineare Regression nicht mehr auf 120 Werten, die pro Sekunde gesammelt wurden, sondern auf 8 Werten, die den Durchschnitt von je 15 Sekunden bilden.
Man kann anhand von Abbildung 9.1a sehr gut erkennen, dass es praktisch keinen Unterschied macht, ob 120 Einzelwerte oder 8 Durchschnittswerte verwendet werden. Anhand der Steigung kann außerdem das Verhalten von *CHERUB* in dem Experiment besser nachvollzogen werden. Erkennbar ist, dass um Sekunde 1200 und Sekunde 1450 jeweils noch einmal ein kleiner Anstieg



(a) Vergleich zwischen 120 Einzelwerten und 8 Durchschnittswerten.



(b) Vergleich zwischen der Verwendung von 60, 120 und 300 Sekunden Historie.

Abbildung 9.1: Berechnete Steigung von *CHERUB*, anhand derer die Lastvorhersage getroffen wird.

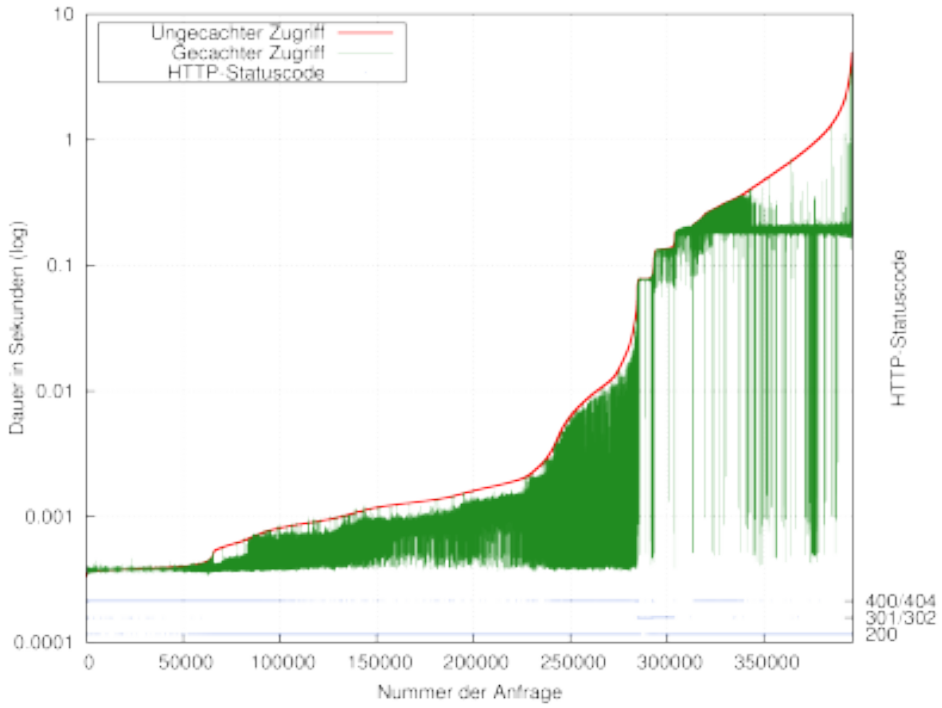


Abbildung 9.2: Zusammensetzung der Anfragen in dem verwendeten Trace.

zu verzeichnen ist. Es ist also wahrscheinlich, dass *CHERUB* bei gewissen Konfigurationen an diesen Stellen noch einmal Maschinen anschalten wird.
Abbildung 9.1b zeigt zusätzlich den Unterschied der berechneten Steigung, wenn für die Berechnung 60, 120 bzw. 300 Sekunden (4, 8 bzw. 20 Durchschnittswerte) Historie verwendet werden. Man erkennt deutlich, dass bei Verwendung von weniger historischen Daten der Anstieg jeweils stärker schwankt. Genauso ist auch zu erkennen, dass bei mehr Daten wesentlich weniger Schwankungen vorhanden sind, was zu einer wesentlich trägeren Reaktionzeit führt. Beide Effekte wurden schon in den bisherigen Messungen beobachtet und werden durch die Darstellung noch einmal unterstützt. Es wird daher weiterhin die 120 Sekunden Historie verwendet.
Abbildung 9.2 zeigt die Bearbeitungsdauer der Anfragen, gecached und ungecached, mit dazugehörigem HTTP-Statuscode (abgetragen auf der 2. Y-Achse) und Tabelle 9.4 zeigt die dazugehörigen Perzentile. Zu sehen ist, dass 2/3 der Anfragen sehr schnell (< 10 Millisekunden) bearbeitet werden können. Es gibt allerdings auch 4 % schwergewichtige (> 1 Sekunde) Anfragen. Vom Caching profitieren durchweg alle Anfragen, wobei Anfragen im oberen Viertel verhältnismäßig mehr vom Caching haben. Die angegebenen Geschwindigkeiten sind auch die, die in der Anfrage-Bibliothek von *ClusterSim* hinterlegt sind (siehe 8.3, Seite 110).
Nach ersten Referenztests mit 100 VNodes, die über den gesamten Trace hinweg angeschaltet waren, hat sich herausgestellt, dass der so erzeugte Trace das Setup nicht im geplanten Maße auslastet. Es wurde ermittelt, dass zum Zeitpunkt der maximalen Last nur eine Auslastung von etwa 35,4 % vorhanden ist. Deshalb wurde eine Gewichtung für *ClusteSim* als Option implementiert. Über diese Option kann eine Gewichtung angegeben werden, mit der jede Anfrage, die am System ankommt, multipliziert wird. Die neu berechnete Dauer der Anfragen wird bei maximal 99 % des

Perzentil (ungecached)	Dauer der Anfrage in s
33.	0,001
67.	0,01
74.	0,1
96.	1

Tabelle 9.4: Verteilung der Dauer der Anfragen.

Gewichtung	Maximale Auslastung in %	QoS in %	EV in Wh
2.6	88,59	99,02	3140,20
2.8	92,49	98,67	3214,50
3.0	93,86	98,39	3283,96
3.2	96,80	97,89	3352,49
3.4	99,31	97,16	3417,8

Tabelle 9.5: Auswirkung der Gewichtung der Anfragen. Verwendet wird für die Studie die Gewichtung 2.8.

eingestellten Timeouts begrenzt.
Tabelle 9.5 zeigt alle getesteten Gewichtungen zusammen mit der daraus resultierenden Auslastung die an der Lastspitze stattfindet. Zusätzlich angegeben ist die jeweils erzielte *QoS* und der erzielte *EV*. Zu erkennen ist, das bei einer kleineren Gewichtung eine zu geringe Auslastung stattfindet und bei einer größeren Gewichtung die *QoS* für das Referenzszenario zu stark sinkt. Eine bessere Auslastung kann für die jeweiligen Gewichtungen nur durch ein besseres Scheduling-Verfahren erreicht werden. Um eine hohe Auslastung bei noch akzeptabler *QoS* für ein Referenzszenario zu erreichen, wurde sich für eine Gewichtung von 2,8 entschieden. Dabei wird das Setup zur Lastspitze zu 92,49 % ausgelastet und erzielt eine *QoS* von 98,67 % bei einem *EV* von 3214,5 Wh. Die so erreichten Werte werden für die beiden Metriken *QoS* und *EV* in den folgenden Simulationen als Referenzwerte, die ohne den Einsatz von *CHERUB* erzielt werden können, verwendet.

9.5 Resultate

Um die zentralen Ergebnisse für die Skalierbarkeit von *CHERUB* zu präsentieren, wird als erstes auf Vortests eingegangen, in denen bereits Erkenntnisse gewonnen wurden, die dann in dem Hauptexperiment Berücksichtigung finden konnten.

9.5.1 Vortests

Im Rahmen der Lehrveranstaltung „Leistungsanalyse: Messen, Modellieren und Simulation” (Sommersemester 2015, Universität Potsdam) wurden von dem Studenten Jörg Kapelle Experimente durchgeführt, die genutzt wurden, um *CHERUB* und *ClusterSim* zu härten und erste Erkenntnisse über eine gute Konfiguration zu gewinnen. Nach zusätzlichen eigenen Tests konnten die folgenden

Bootzeit in s	EV in Wh	EE in %	PZW
180	2862,14	10,96	181
120	2734,15	14,94	215
60	2604,77	18,97	249
30	2477,34	22,93	404
5	2255,02	29,85	1094

Tabelle 9.6: Minimal möglicher Energieverbrauch bei maximaler Qualität des Dienstes in Abhängigkeit der Bootzeit.

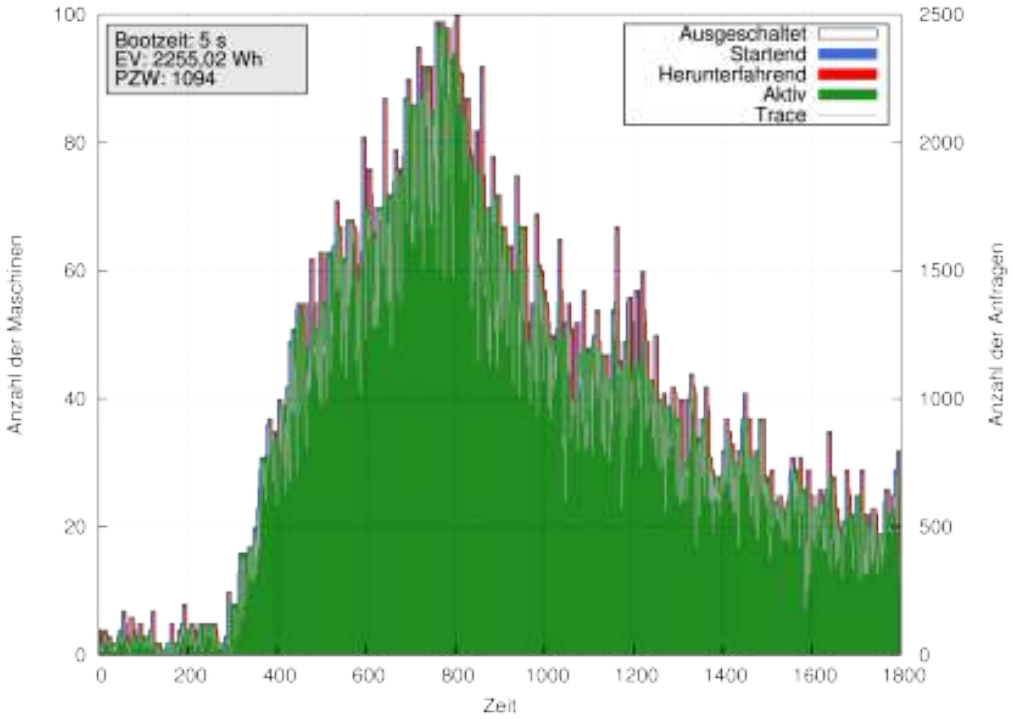
Erkenntnisse bereits in die Studie mit einfließen.

Sequenzielles Abschalten Im Rahmen der von Jörg Kapelle durchgeführten Experimente wurden die Werte 6 und 12 für den Faktor sequenzielles Abschalten getestet. Dabei haben diese Werte zwar gute Ergebnisse erzielt, allerdings waren diese darauf zurückzuführen, dass der verwendete Trace für diese beiden Werte gut passt und ein Trace mit anderen Charakteristika vermutlich nicht so gut abschneidet. Da sich außerdem `automatisch` bereits hier als überlegen gegenüber einem fixen Wert herausgestellt hat, wurde das Hauptexperiment nur mit den Werten 1 und `automatisch` durchgeführt.

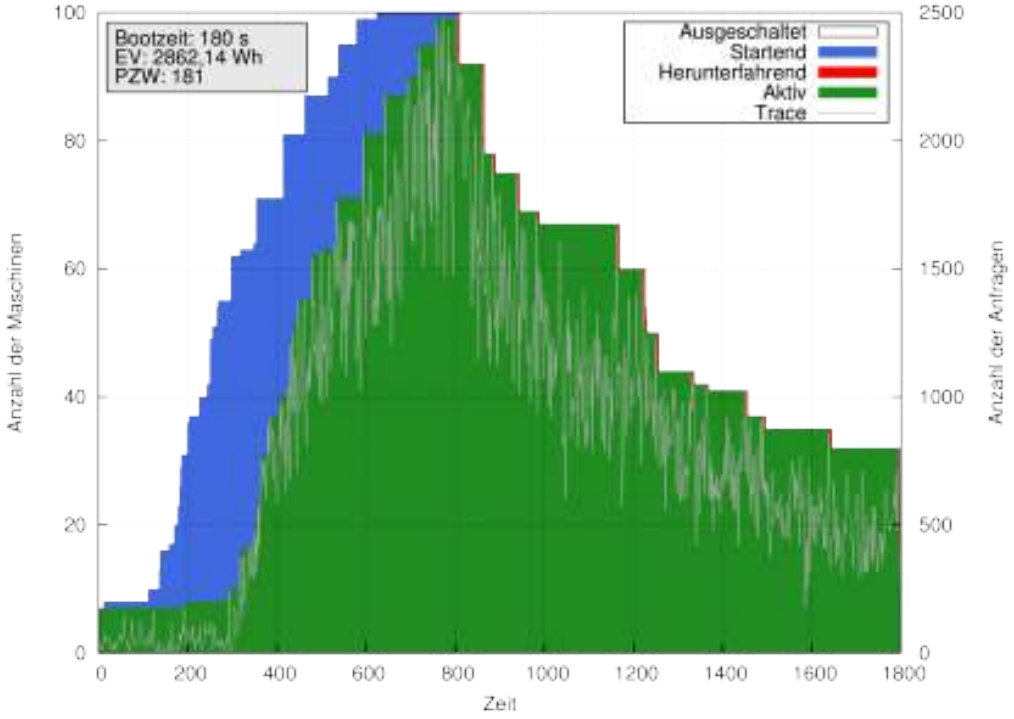
Dynamischer Unterlast-Schwellwert Der in Abschnitt 6.2.3 (Seite 83) eingeführte dynamische Unterlast-Schwellwert wurde ohne seine 80 % Grenze getestet. Dabei hat sich gezeigt, dass die Grenze aus gutem Grund verwendet wird. Die Ergebnisse haben ohne Grenze stark geschwankt und waren nicht verwertbar.

9.5.2 Hauptexperiment

Um die erzielten Ergebnisse des Experiments sinnvoll einordnen zu können, wurden für die einzelnen Bootzeiten der jeweils korrespondierende minimal mögliche Energieverbrauch ermittelt. Für diesen Energieverbrauch wird ein allwissendes Orakel für die An- und Abschaltentscheidungen angenommen, welches die eingesparte Energie, bei maximaler Qualität des Dienstes, optimieren soll. Die daraus resultierenden Einsparungen und physikalischen Zustandswechsel sind in Tabelle 9.6 abgebildet. Da durch die Hilfe des Orakels immer so viele Maschinen wie notwendig an sind, wird eine maximale *QoS* angenommen. Da trotzdem die aktiven Maschinen nicht immer zu 100 % ausgelastet sind, wird für den optimalen Energieverbrauch bei aktiven Maschinen eine Auslastung von 92,49 % angenommen (siehe Tabelle 9.5). Die Spanne der verbrauchten Energie reicht dann von 2862,14 Wh für sehr langsam bootende Hardware, bis zu 2255,02 Wh bei der Verwendung von STR. Die Anzahl der *PZW* sind bei der STR-Variante überproportional hoch. Eine reine Optimierung bezüglich der eingesparten Energie ist daher nicht wünschenswert. Die erzielten Werte aus Tabelle 9.6 werden im Folgenden als optimal erreichbare Werte bezüglich *EV/EE* angesehen. In den Abbildungen 9.3 ist für zwei Bootzeiten illustriert, wie die Maschinen des Clusters im Orakel-Fall verwaltet würden. Dabei ist farblich der Zustand der einzelnen Maschinen gekennzeichnet und der verwendete Trace hellgrau im Hintergrund zu sehen. In Abbildung 9.3a kann



(a) Verlauf bei einer Bootzeit von 5 Sekunden.



(b) Verlauf bei einer Bootzeit von 180 Sekunden.

Abbildung 9.3: Zustände des Clusters in Abhängigkeit von der Bootzeit seiner Knoten und bei minimal möglichem Energieverbrauch.

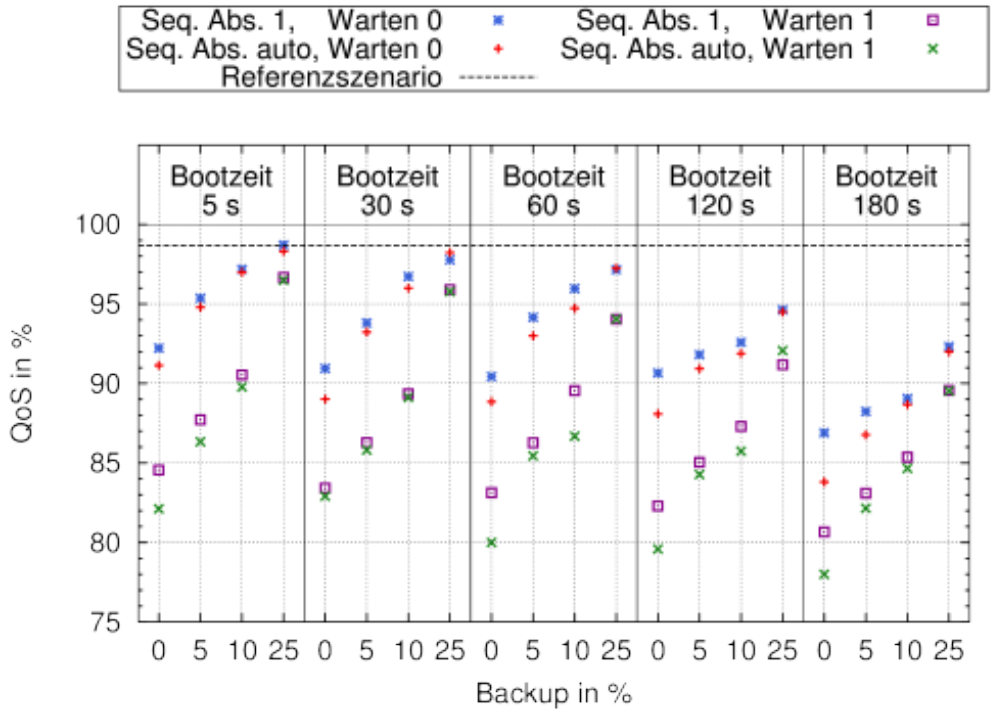


Abbildung 9.4: Zusammenfassung der Ergebnisse bezüglich der Metrik SLA in Prozent.

man gut erkennen, woher die erhöhten *PZW* kommen. Durch das perfekte Wissen können auch kleine Lastschwankungen berücksichtigt werden, für die dann Maschinen gestartet werden, die allerdings anschließend sofort wieder abgeschaltet werden. Im starken Kontrast dazu steht das Verhalten bei einer Bootzeit von 180 Sekunden (Abbildung 9.3b). Dort können auch größere Bereiche mit Energiesparpotenzial (ab Sekunde 1000) nicht ausgenutzt werden, da abgeschaltete Knoten nicht rechtzeitig wieder zur Verfügung stehen würden, um spätere Last abzufangen.
Nachfolgend werden die Ergebnisse des Experiments nach Metrik geordnet vorgestellt. Der gesamte Datensatz findet sich in Anhang A: Messergebnisse, Seite 160, Tabelle A.8 und A.9.

Quality of Service / Service Level Agreement: Die Ergebnisse für die *QoS* sind in Abbildung 9.4 dargestellt. Da die Bootzeit der Faktor ist, welcher dafür verantwortlich ist, wie lange es dauert bis eine Maschine einsatzbereit ist, beeinflusst dieser die *QoS* besonders stark.
Der Anschaltverzögerer (kurz *Warten*) erhöht die Bootzeit de facto um eine Minute und beeinflusst daher die *QoS* mit bis zu 9,02 % weniger bedienten Anfragen. Obwohl der Faktor zum Verhindern von *State-Flapping* gedacht war und er auch tatsächlich in den meisten Fällen zu weniger *PZW* (siehe Abbildung 9.10) führt, ist der Vergleich zu den *PZW*-Werten für den optimalen Fall immer noch in einem sehr akzeptablen Bereich. *Warten* beeinflusst die *QoS* und die *Anfragedauer* so stark negativ, dass die mit *Warten* erzielten Ergebnisse nicht in der Diskussion betrachtet werden.
Bei einer kurzen Bootzeit von 5 Sekunden sind alle erzielten Ergebnisse gut wenn nicht gewartet wird und liegen dann zwischen 98,70 % und 91,14 % (Median: 96,99 %, Referenz: 98,67 %, siehe Tabelle 9.6). Bei einer längeren Bootzeit verpassen die neu gebooteten Maschinen bei einem zu starken und abrupten Lastanstieg den Beginn der Höchststand-Phase. Abbildung 9.5 demonstriert

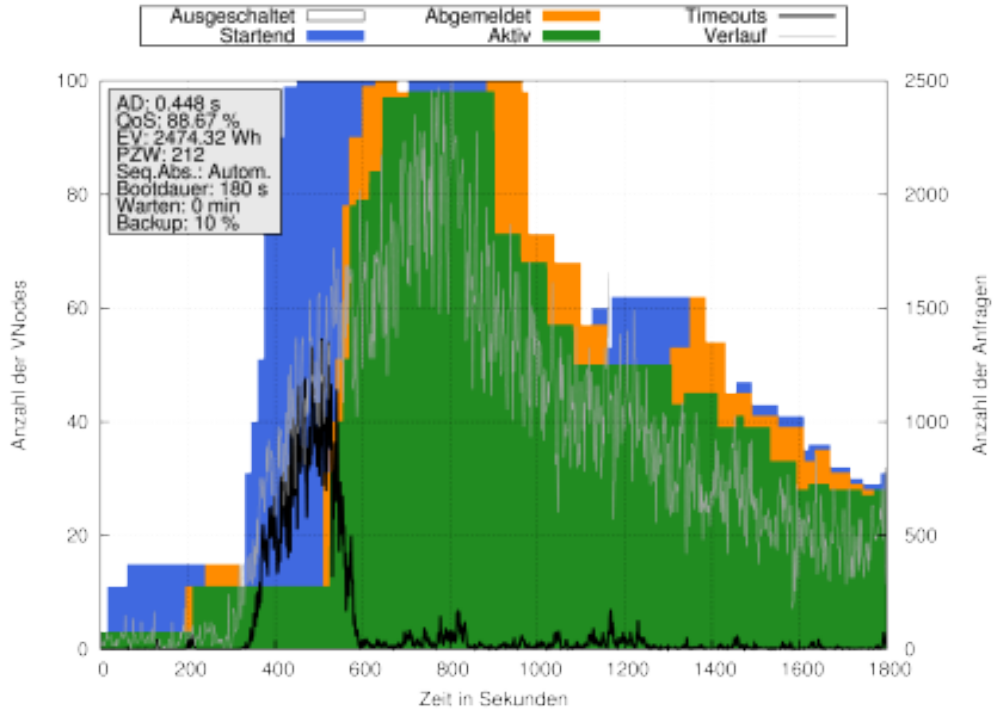


Abbildung 9.5: Die lange Bootzeit sorgt für viele Timeouts bei Beginn der Höchststand-Phase.

dies mit einer Bootzeit von 180 Sekunden. Man sieht, wie *CHERUB* bei dem ersten Anzeichen der kommenden Last die ersten neuen Maschinen startet (Sekunde ~300). Wie beschrieben gehen dann durch die lange Bootzeit zwischen Sekunde 350 und Sekunde 550 die meisten Anfragen verloren (schwarze Linie). Klar zu erkennen ist, dass zur Lastspitze verhältnismäßig wenige Timeouts entstehen und die Spitze damit erfolgreich bearbeitet wurde, selbst bei einem langsamen Setup mit 180 s Bootzeit.
Das Backup wirkt sich ebenfalls sehr stark auf die *QoS* aus. Im 25 %-Fall liegen die *QoS*-Ergebnisse zwischen 98,70 % und 89,53 % und erzielen immer noch mindestens Energieeinsparungen von 10,02 % (2892,39 Wh). Es ist somit möglich, auch langsam bootende Hardware durch ein entsprechend hohes Backup in einem gewissen Maße auszugleichen.
Der Unterschied zwischen dem sequenziellen Abschalten (kurz *Seq.Abs.*) und dem automatischen ist insgesamt klein, wobei das langsame Abschalten in den meisten Fällen leicht bessere Ergebnisse erzielt, da nach einer Lastspitze mehr Maschinen länger angeschaltet bleiben und so auch kleinere Schwankungen besser abgefangen werden können.

Anfragedauer: Die Ergebnisse für die *Anfragedauer* können in Abbildung 9.6 betrachtet werden. Auch hier ist das *Warten* wieder mit starken Einbußen verbunden und diese Ergebnisse werden wieder nicht in der Diskussion betrachtet. Im Grunde verhalten sich die Ergebnisse ähnlich denen der *QoS*. Bei einer hohen *QoS* ist auch eine niedrige *Anfragedauer* zu beobachten. Einen Unterschied macht hier allerdings das sequenzielle Abschalten. Ist es aktiv, sind die Antwortzeiten wesentlich besser. Dieses Verhalten ist damit zu erklären, dass nach der Lastspitze viel mehr Maschinen angeschaltet sind als notwendig und damit die Anfragen in dieser Situation viel schnel-

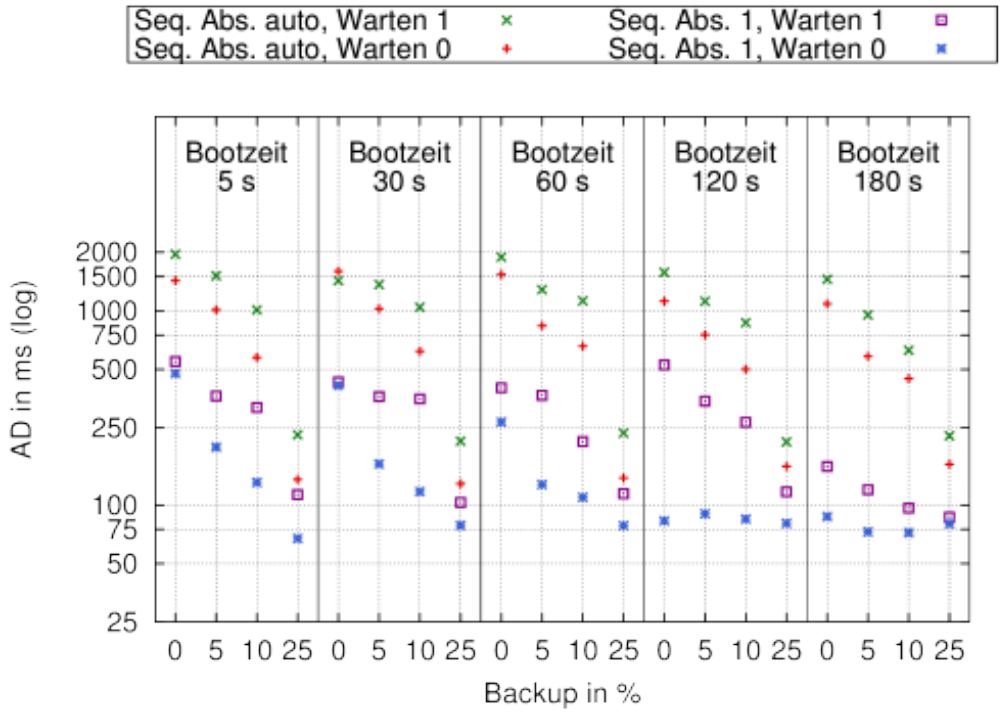


Abbildung 9.6: Zusammenfassung der Ergebnisse bezüglich der Anfragedauer in Millisekunden.

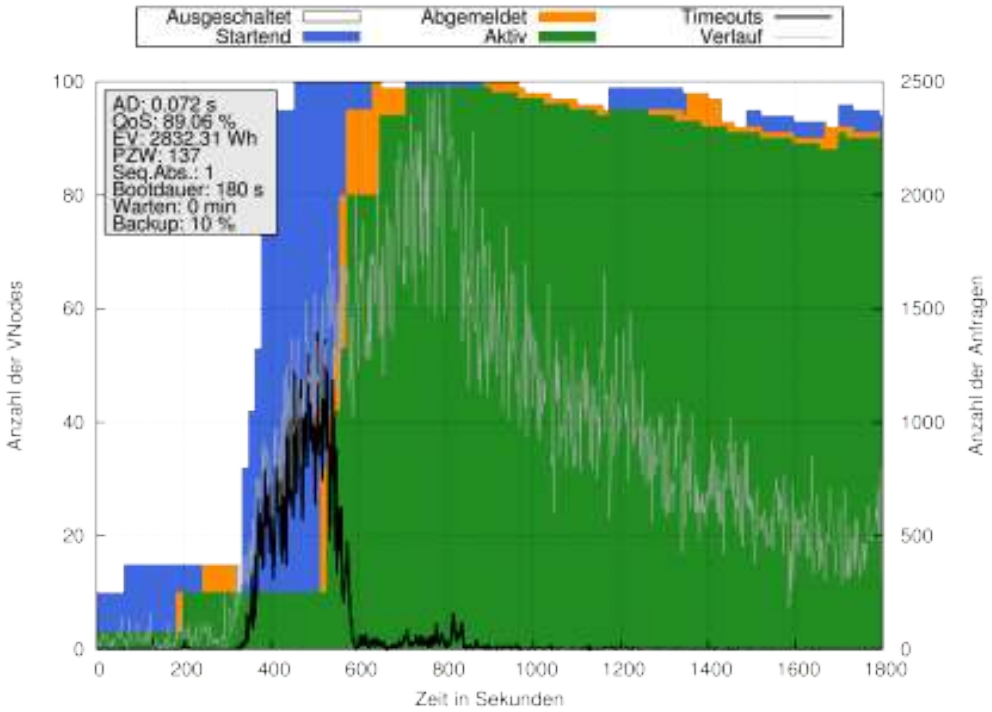


Abbildung 9.7: Durch das langsame Abschalten sinkt die Anfragedauer, vor allem nach einer Lastspitze.

ler beantwortet werden können. Dies ist sehr gut in Abbildung 9.7 zu sehen. Hier sind über die gesamte zweite Hälfte des Verlaufes gut doppelt so viele Maschinen aktiv, als nötig wären.
Cachingeffekte scheinen nicht aufzutreten. Da Maschinen ihren Cache verlieren, wenn sie abgeschaltet werden, hätte man bei erhöhten *PZW*-Werten (siehe Abbildung 9.10) vermuten können, dass dies auch zu einer Verschlechterung der *Anfragedauer* führt. Es ist allerdings keine Korrelation zwischen erhöhtem *PZW* und erhöhter *Anfragedauer* zu erkennen.

Energieverbrauch / Eingesparte Energie: Abbildung 9.8 zeigt die Ergebnisse bezüglich der eingesparten Energie in Wh. Verglichen wird diese mit dem vorher berechneten, minimal möglichen Verbrauch, dargestellt als gestrichelte schwarze Linie. In Abbildung 9.9 ist die daraus resultierende eingesparte Energie in Prozent zu sehen. Als Berechnungsgrundlage wurde der Verbrauch des Referenzszenarios verwendet (3214,50 Wh, siehe Tabelle 9.6).
Dabei fällt auf, dass sich bei steigenden Bootzeiten immer mehr Konfigurationen vom Optimum entfernen. Da die Werte auch kleiner (Abbildung 9.8) bzw. größer (Abbildung 9.9) werden als das Optimum und das Optimum auch von einer optimalen *QoS* ausgeht, können diese zusätzlichen Einsparungen nur auf Kosten der *QoS* gemacht werden (verspätetes Anschalten bzw. langes Booten). Dass dies der Fall ist, konnte schon in Abbildung 9.4 festgestellt werden. Die Bootzeit hat einen Einfluss von bis zu 9,54 % auf die eingesparte Energie, da Maschinen mit einer höheren Bootzeit früher gestartet werden müssen und daher länger an sind. Außerdem ist die Chance bei ihnen höher, dass sie gebootet werden müssen, obwohl sie doch nicht benötigt werden.
Einen großen Einfluss hat das sequenzielle Abschalten. Da bei einer langsamen Konfiguration (siehe Abbildung 9.7) die Maschinen nur Stück für Stück abgeschaltet werden, gehen Einsparungen von bis zu 12,87 % verloren. Dies ist vor allem nach einem starken, kurzen Anstieg wie im verwendeten Trace der Fall. Bei der Verwendung von `automatisch` kann *CHERUB* dieses Potenzial voll ausschöpfen, da nach einem schnellen Abfall der Last die vielen überschüssigen Maschinen auch wieder schnell abgeschaltet werden. Die Konfigurationen mit `automatisch` erzielen in jeder Simulation eine höhere Einsparung als ihre sequenziellen Pendants.
Man kann außerdem sehr gut erkennen, wie das Backup einen fast linearen, negativen Einfluss auf die eingesparte Energie hat. Trotz eines Backups von 25 % wird aber immer zwischen 10,02 % - 25,25 % Energie eingespart.
Im gleichen Maße wie der *Warten*-Faktor einen negativen Einfluss auf die *QoS* hat, hat er einen positiven Einfluss auf die eingesparte Energie, da Maschinen entweder verspätet angeschaltet werden (wodurch sie länger aus sind) oder gar nicht erst angeschaltet werden.

Physikalische Zustandswechsel: Im Vergleich mit den *PZW* des optimalen Szenarios (Tabelle 9.6) verhindert *CHERUB* eine überhöhte Aktivität. Ein *PZW*-Wert von 200 bedeutet für das gewählte Szenario mit 100 Maschinen, dass im Schnitt jede Maschine einmal an- und einmal ausgeschaltet wird. Analysiert man den Trace etwas genauer, stellt man fest, dass wenigstens 185 *PZW* für eine optimale Bearbeitung notwendig sind. Das Cluster beginnt mit 3 gestarteten Maschinen, daher müssen bis zum Maximum des Traces 97 neue Maschinen gestartet werden. Für das Ende des Traces werden 12 Maschinen benötigt, weshalb 88 wieder abgeschaltet werden sollten. Die gemessenen Werte, die in Abbildung 9.10 dargestellt sind, sind somit alle in einem normalen Bereich.
Die Werte für das sequenzielle Abschalten sind alle geringer (besser). Die Ursache hierfür ist, dass

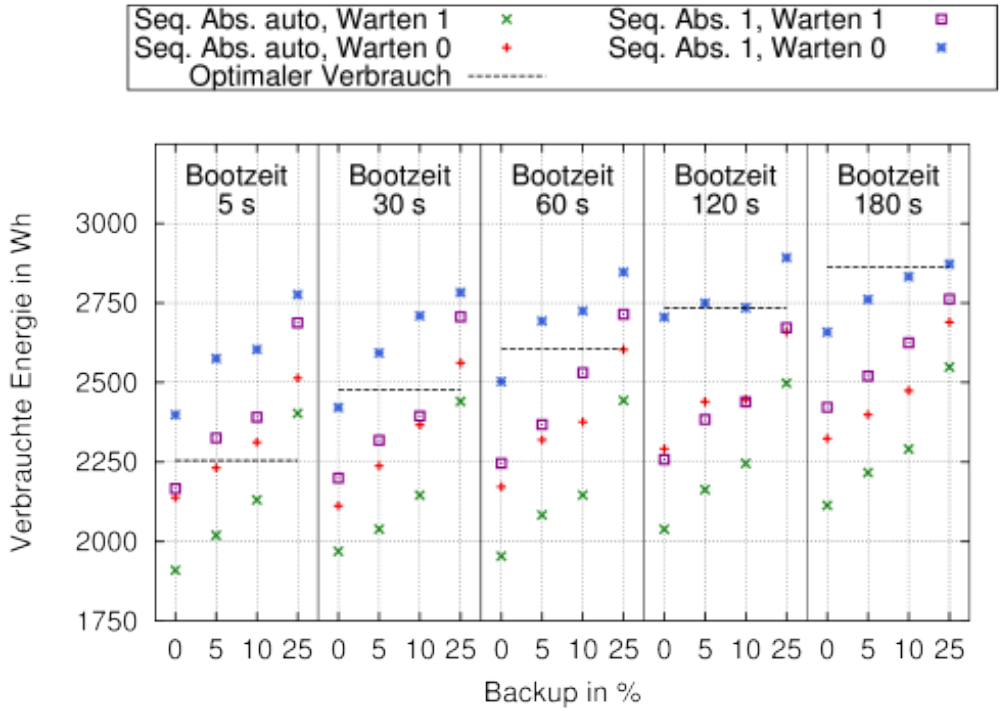


Abbildung 9.8: Zusammenfassung des Energieverbrauchs in Wattstunden.

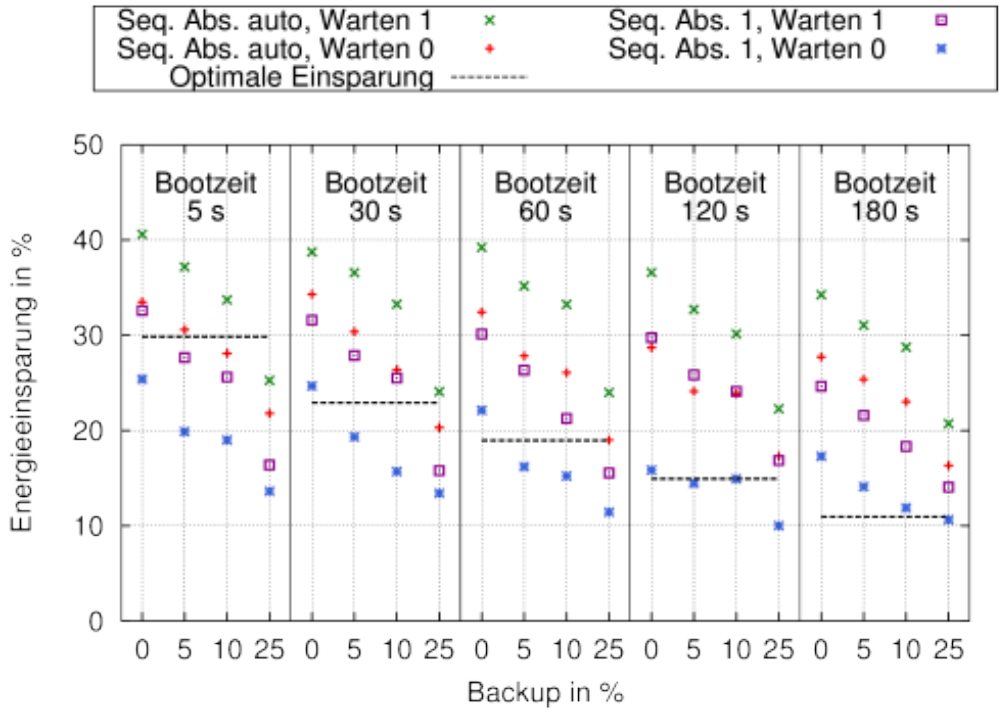


Abbildung 9.9: Zusammenfassung der eingesparten Energie in Prozent.

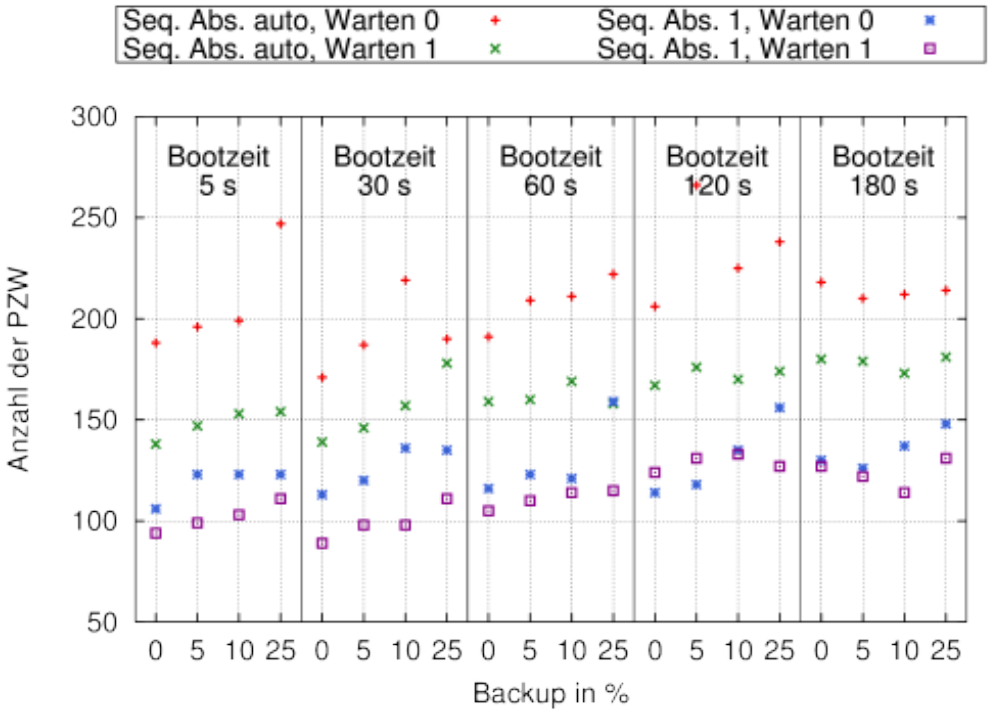


Abbildung 9.10: Zusammenfassung der Ergebnisse der physikalischen Zustandswechsel.

am Ende des Traces nicht genug Zeit bleibt, um alle Knoten wieder nach und nach abzuschalten. Wäre der Trace länger und die Last würde auf dem Niveau bleiben, welche am Ende herrscht, würde der *PZW*-Wert für diese Konfigurationen ebenfalls höher sein. Im Gegensatz dazu kann es passieren, dass beim aggressiveren Abschalten im Laufe des Traces noch einmal Maschinen hochgefahren werden müssen, welche im sequenziellen Fall noch an sind und was zusätzliche *PZW* verursacht.
Dasselbe Verhalten trifft umgekehrt auf den *Warten*-Faktor zu. Wenn nicht mit dem Booten gewartet wird, werden eventuell Maschinen angeschaltet, obwohl diese später überflüssig sind oder nur sehr kurz gebraucht werden. Dies erhöht die *PZW*, verbessert aber womöglich die *QoS*.
Das Backup und die Bootzeit haben nur einen geringen Einfluss. Tendenziell sorgen beide Faktoren dafür, dass bei hohen Werten für Backup und Bootzeit auch die *PZW* steigen.

Score: Der erreichte *Score* der einzelnen Strategien können in Tabelle 9.7 betrachtet werden. Dabei ist zu beachten, dass der maximale *Score*, der mathematisch erreicht werden kann, 80 Punkte beträgt. Da die kostenbezogenen Metriken (*EE/EV* und *PZW*) im direkten Gegensatz zu den Service-Metriken (*QoS* und *Anfragedauer*) stehen, wäre das Erreichen einer vollen Punktzahl nur bei einer entsprechenden Gewichtung möglich.
Für jede Strategie wurde für jede der verwendeten Bootzeiten das jeweils beste Ergebnis angegeben. So kann ein Systemadministrator in Abhängigkeit von seinen Maschinen und der gewünschten Strategie, die er verwenden möchte, die notwendige Konfiguration aus der Tabelle ablesen, mit der er die Strategie umsetzen kann.

Strategie	Bootzeit in s	Score	QoS in %	EV in Wh	EE in %	AD in ms	PZW	Seq. Abs.	Warten in min	Backup in %
Hochleistungs-betrieb	5	70,3	98,70	2776,27	13,63	67	123	1	0	25
	30	66,5	97,77	2783,10	13,42	79	135	1	0	25
	60	63,6	97,16	2846,70	11,44	78	159	1	0	25
	120	56,9	94,63	2892,39	10,02	81	156	1	0	25
	180	53,1	92,32	2872,34	10,64	80	148	1	0	25
Preiswerter Betrieb	5	61,0	84,56	2166,53	32,60	548	94	1	1	0
	30	60,5	83,42	2199,23	31,58	430	89	1	1	0
	60	56,3	83,13	2246,20	30,12	401	105	1	1	0
	120	49,2	82,30	2257,95	29,76	527	124	1	1	0
	180	43,1	78,00	2113,57	34,25	1452	180	auto.	1	0
Ausgeglichener Betrieb	5	55,4	96,99	2311,71	28,08	574	199	auto.	0	10
	30	50,8	96,00	2366,73	26,37	619	219	auto.	0	10
	60	47,8	94,73	2376,07	26,08	658	211	auto.	0	10
	120	41,1	92,07	2498,02	22,29	211	174	auto.	1	25
	180	33,9	84,66	2290,76	28,74	626	173	auto.	1	10
Alte Hardware	5	66,2	84,56	2166,53	32,60	548	94	1	1	0
	30	66,9	84,56	2199,23	31,58	430	89	1	1	0
	60	62,3	83,13	2246,20	30,12	402	105	1	1	0
	120	60,7	90,66	2705,04	15,85	83	114	1	0	0
	180	56,9	85,37	2624,92	18,34	97	114	1	1	10

Tabelle 9.7: Erreichter Score der verschiedenen Strategien.

Für die einzelnen Strategien ergeben sich folgende Erkenntnisse:

Hochleistungscluster: Wenn ein Hochleistungscluster betrieben werden soll, sollte immer ausreichend Backup vorhanden sein und es sollte bei Anzeichen eines Lastanstiegs nicht gewartet werden, bis neue Maschinen gestartet werden. Es ist außerdem ratsam, Maschinen nur langsam wieder abzuschalten. Durch dieses Verhalten kann, abhängig von der verwendeten Hardware, eine *QoS* von 92,32 % - 98,7 % und eine *Anfragedauer* von 81 ms - 67 ms gewährleistet werden. Trotz des hohen Backups werden immer noch 10,02 % - 13,63 % Energie eingespart. Anhand des hohen *Scores* des 5-Sekunden-Setups sieht man, dass diese Konfiguration besonders gut für die Aufgabe geeignet ist.

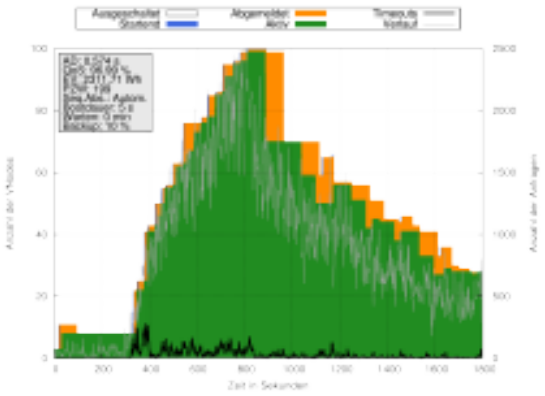
Preiswerter Betrieb: Für einen preiswerten Betrieb sind die beiden Faktoren, die besonders stark zwischen Leistung und Einsparung entscheiden, *Warten* und Backup, praktisch umgekehrt zum Hochleistungscluster einzustellen. Es muss vollständig auf Backup verzichtet werden und gleichzeitig sollte immer erst abgewartet werden, ob zusätzliche Hardware wirklich benötigt wird. Allerdings sollte auch hier (mit einer Ausnahme) die Hardware nur nach und nach wieder abgeschaltet werden. Ein automatisches Abschalten erzielt zwar bessere Werte beim *EV* und den *EE*, aber da das sequenzielle Abschalten wesentlich weniger *PZW* verursacht, wird es in diesem Setup bevorzugt. Mit dieser Konfiguration können, abhängig von der Bootzeit, 29,76 % - 34,25 % Energie eingespart werden. Dabei müssen allerdings Einschränkungen bei der *QoS* hingenommen werden, so dass diese bis auf 78 % sinken kann und maximal noch 84,56 % erreichen kann.

Ausgeglichener Betrieb: Für einen ausgeglichenen Betrieb sollte unabhängig von der Bootzeit immer ein schnelles Abschalten der Knoten vorgenommen werden. Gleichzeitig sollte aber genügend Backup verwendet werden, um Lastspitzen abfangen zu können. Bei Hardware mit erhöhter Bootzeit sollte tendenziell mehr Backup verwendet werden und dafür mit dem Anschalten gewartet werden. Berücksichtigt man diese Einstellungen, können bei Verwendung von schnell bootender Hardware Einsparungen von 28,8 % bei einer gleichzeitigen *QoS* von 96,99 % erzielt werden.

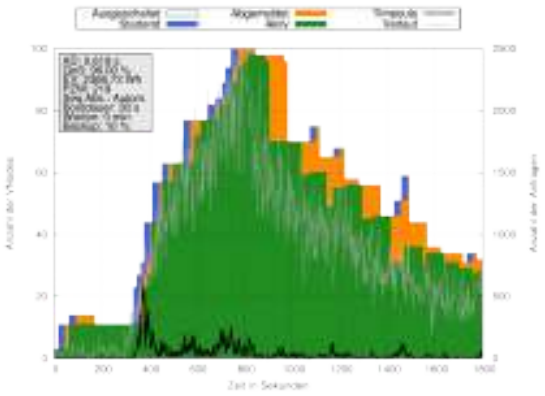
Alte Hardware: Bei alter Hardware empfiehlt es sich, Maschinen nur zögerlich abzuschalten, kein Backup zu verwenden und zu warten, ehe man auf Änderungen reagiert. Ist die Hardware besonders langsam beim Booten, kann es von Vorteil sein, ein kleines Backup zu nutzen oder nicht zu warten. Hier sind die Ergebnisse nicht ganz einheitlich. Unter diesen Einstellungen leidet allerdings die *QoS* und liegt nur zwischen 83,13 % und 90,66 %. Die Einsparungen können dafür aber, vor allem bei schneller bootender Hardware, bei bis zu 32,6 % liegen.

Fazit Zusammenfassend sollte erwähnt werden, dass schnellere Hardware, mit nur einer Ausnahme, immer einen höheren *Score* erzielt als ihre langsameren Pendants. Unabhängig von der Strategie, die man verwenden möchte, sollte also vor allem Wert auf eine möglichst kurze Bootzeit gelegt werden. Die Größe des Backups hat insgesamt einen etwa linearen Einfluss auf das Verhältnis zwischen *EE/EV* und *QoS*. Außerdem kann mit einem genügend großen Backup auch langsamere Hardware noch eine *QoS* von 92,3 % erreichen und dabei 10,6 % Energie einsparen. Das aggressive Abschalten kann im vorliegenden Lastspitzen-Szenario bis zu 12,9 % mehr Energie gegenüber einem sequenziellen Abschalten einsparen. Ein explizites Warten vor einem Bootbefehl hat sich als überflüssig (und schadhaft) herausgestellt, da die Lastvorhersage ein *State-Flapping* bereits verhindert. Insgesamt stellt der ausgewogene Betrieb bei den Strategien einen sehr guten Kompromiss zwischen hoher *QoS* und hohen *EE* dar. Die Abbildungen 9.11a bis 9.11e zeigen die Verläufe für den ausgeglichenen Betrieb.

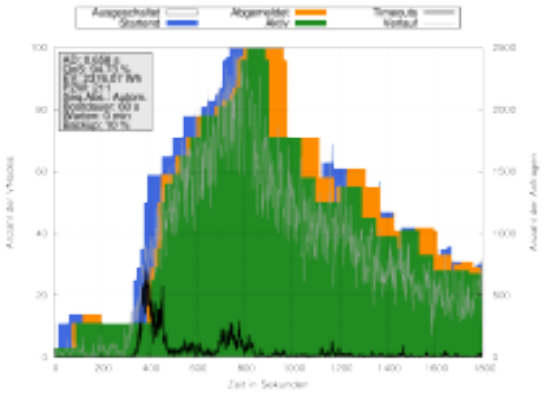
Abgrenzung zu den Messungen mit realen Rechnern Obwohl die Ergebnisse aus Kapitel 6 teils deutlich besser in Bezug auf die *QoS* waren, können diese nur mit Bedacht mit denen der Simulation verglichen werden. Der wichtigste und gravierendste Unterschied ist, dass bei den realen Messungen mit zwei Back-Ends bis zum Anschalten der zweiten Maschine de facto immer 50 % der Gesamtkapazität einsatzbereit war. Dies entspricht in dem 100-Knoten-Setup, bei dem nur drei Knoten zu Beginn angeschaltet waren, einem Backup von 47 %. Es ist zu erwarten, dass bei einem so hohen Backup auch im 100-Knoten-Setup eine genauso hohe *QoS* erreicht werden würde, bzw. zumindest die *QoS* der 100-Knoten-Referenzmessung erreicht werden würde. Die Tendenz dafür ist schon in den Fällen mit einem 25 % Backup deutlich zu erkennen.
Außerdem dürfen, wie bereits erwähnt, *FRT* und *Anfragedauer* nicht eins zu eins mit einander verglichen werden, da die *Anfragedauer* potenziell mehr Zeiten als die *FRT* beinhaltet und deshalb von Natur aus größer ist.
Auch die *Downtime* und der *EV* bzw. die *EE* dürfen nicht eins zu eins verglichen werden. Die *Downtime* berücksichtigt nicht, wie stark Maschinen im Betrieb ausgelastet sind und stellt daher einen weniger komplexen Vergleich mit seiner Referenzmessung dar und entspricht nicht genau der *EE*. Dafür hätte im realen Setup Hardware zum Stromzählen verwendet werden müssen.



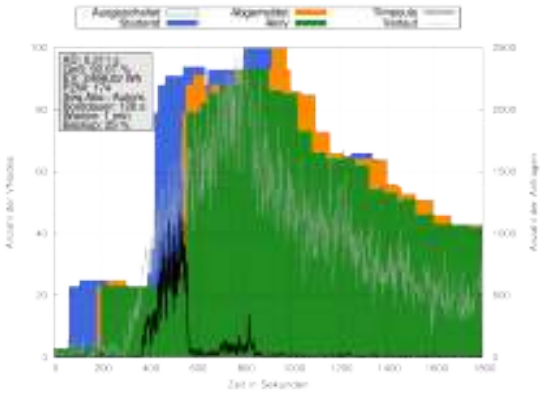
(a) Bootzeit 5 s.



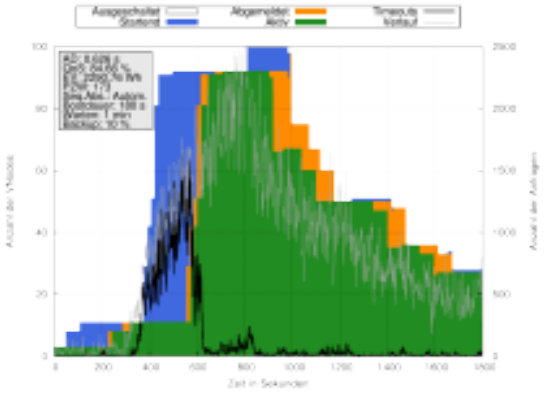
(b) Bootzeit 30 s.



(c) Bootzeit 60 s.



(d) Bootzeit 120 s.



(e) Bootzeit 180 s.

Abbildung 9.11: Verläufe mit dem höchsten *Score* bei einem ausgeglichenen Betrieb, in Abhängigkeit von der verwendeten Hardware.

9.6 Zusammenfassung

In diesem Kapitel wurde die Skalierbarkeit von *CHERUB* mit Fokus auf den Algorithmus und die erzielten Ergebnisse untersucht. Mit Hilfe von *ClusterSim* wurde *CHERUB* mit verschiedensten Konfigurationen gemessen. Resümierend kann gesagt werden, dass für jede untersuchte Strategie und jede getestete Hardware mindestens eine Konfiguration gefunden werden konnte, die die gewählte Strategie umsetzt. Dabei waren die Ergebnisse von schnell bootender Hardware immer besser als die von langsamer bootender Hardware. Es wurde außerdem gezeigt, dass die Schwierigkeit nicht darin besteht, die Lastspitze selbst abzufangen, sondern die Probleme in den ersten Momenten nach dem Beginn eines solchen Lastanstieges entstehen. Dieser Anstieg kann nur mit sehr schnell bootender Hardware oder mit genügend Backup abgefangen werden.
Mit diesem Kapitel wurde die Skalierbarkeit des Verfahrens von *CHERUB* gezeigt. Da während der Experimente *CHERUB* mit nur wenigen Änderungen am Code (siehe Abschnitt 8.4, Seite 113) betrieben wurde und es zu keinen Unstimmigkeiten hinsichtlich der Laufzeit kam, kann dies auch als zusätzlicher Skalierungsnachweis für die Laufzeit von *CHERUB* für ein 100-Maschinen-Cluster angesehen werden.

10 Abschließendes Fazit und weitere Forschungsthemen

Nachdem gezeigt wurde, dass *CHERUB* größere Webserver-Cluster erfolgreich verwalten kann und die zu Beginn von Kapitel 4 (Seite 58) aufgestellten Anforderungen *Energieeinsparung* (siehe Kapitel 6 und 9), *maximale Qualität des Dienstes* (siehe Kapitel 6 und 9), *geringe Belastung der Hardware* (siehe Kapitel 9), *Skalierbarkeit* (siehe Kapitel 7 und 9), *Online-Algorithmus* (siehe Kapitel 4) und *hohe Reaktionsfähigkeit* (siehe Kapitel 7 und 9) erfüllt werden, soll an dieser Stelle noch ein abschließendes Fazit gezogen werden. Außerdem soll noch über andere Aspekte diskutiert werden, die nicht im Fokus der Arbeit lagen. Dazu gehört die Anpassung an heterogene Umgebungen, die Evaluation von weiteren Vorhersage- und Schedulingtechniken sowie der Einsatz im Cloud-Computing.

10.1 Abschließendes Fazit

In dieser Arbeit wurde ein Konzept für ein Energiesparsystem im Server-Load-Balancing-Bereich vorgestellt, das die Maschinen eines Clusters dynamisch nach Bedarf an- und abschaltet. Dafür wurde bestehende verwandte Literatur untersucht und klassifiziert. Um dieses Konzept zu evaluieren, wurde das energieeffiziente Clusterverwaltungssystem *CHERUB* weiterentwickelt. *CHERUB* ist open source und zeichnet sich durch sein modulares Design aus. Dieses Design ermöglicht es ihm, in verschiedene Clustersetups eingesetzt zu werden. Als *Proof of Concept* wurde ein Modul für die Zusammenarbeit von *CHERUB* mit dem Linux-Virtual-Server als Dispatcher und dem Apache-Webserver als Back-End entwickelt und getestet. Diese Kombination wurde, stellvertretend für den gesamten Server-Load-Balancing-Bereich, welcher sich durch viele, schnell zu bearbeitende Anfragen auszeichnet, untersucht. Das verwendete Energiesparverfahren arbeitet dabei online und benötigt als Informationen vorab nur die Zeit zum Hochfahren der Maschinen und ein Maß für deren Rechenleistung. Anhand des *CHERUB*-Prototypen wurden verschiedene Energiesparalgorithmen untersucht und verglichen, teils durch Messungen in einem realen Clustersetup und teils in einer simulierten Clusterumgebung.
Da das reale Setup nur aus zwei Webservern bestand, war es notwendig nachzuweisen, dass sowohl die Strategie als auch die Software skaliert und in größeren Setups eingesetzt werden kann. Für diesen Nachweis wurden zuerst Funktionstests und danach Simulationen durchgeführt. Im Rahmen der Funktionstests wurde zusätzlich die Performance von IMPI-1.5 und IMPI-2.0, sowie die Performance von verschiedenen Prozesserzeugungsfunktionen, welche in Python verfügbar sind, verglichen. Dabei stellte sich heraus, dass IPMI-1.5 etwa 30 % schneller ist, was vermutlich an den zusätzlichen Funktionen von IPMI-2.0 liegt. Die Performance der verschiedenen Prozesserzeugungsfunktionen unter Python waren alle nahezu identisch. Die schnellste Variante von ihnen war eine *fork+execl* Kombination. Es wurde ebenfalls das Analyse-Werkzeug `tcpdump` auf seine

Leistung hin untersucht, da es als dienstunabhängige Lösung zum Protokollieren der Last eingesetzt werden kann. Die Last kann von `tcpdump` durch das Zählen von HTTP *GET-Requests* ermittelt werden. Mit Messungen wurde gezeigt, dass `tcpdump` dabei nur 0,004 % der eingehenden Anfragen verwirft, welche an einen unter Volllast arbeitenden `nginx` Webserver gerichtet waren. Die Einsatzfähigkeit wurde damit nachgewiesen.
Um die Skalierbarkeit der Strategien nachzuweisen, wurde der Simulator *ClusterSim* für beliebig große Apache-Webserver-Cluster entwickelt und validiert. *ClusterSim* arbeitet spurdatenbasiert und ermittelt den Energieverbrauch der simulierten Maschinen abhängig von ihrer Auslastung und auf Basis realer Daten. Um eine realistische Abarbeitung der HTTP-Anfragen zu simulieren, wurde in *ClusterSim* sowohl das Linux-Scheduling als auch die Arbeitsweise des Apache Multi-Processing-Modules modelliert. Die Validierung von *ClusterSim* besteht aus mehreren Experimenten, in denen die Simulationsergebnisse mit den korrespondierenden Messergebnissen verglichen wurden. Dabei wurde festgestellt, dass *ClusterSim* das reale Setup sehr gut nachstellt. Durch eine Schnittstelle zu *CHERUB* ist es mit *ClusterSim* möglich, *CHERUB* in einem simulierten Setup zu testen ohne *CHERUB* selbst simulieren zu müssen.
Mit Hilfe von *ClusterSim* wurde eine Parameterstudie durchgeführt. Diese Studie sollte zum einen die Skalierbarkeit von Strategie und Software nachweisen und zum anderen den Einfluss der Parameter *Länge der Bootzeit*, *Größe des Backups*, *Aggressivität des Abschaltens* und *explizites Warten vor einem Bootbefehl* untersuchen. Als Arbeitslast wurde ein Lastspitzen-Szenario gewählt, welches eine *Worst-Case*-Situation darstellt. Zur Bewertung der Energiesparstrategie und -Parameter wurde die geleistete Dienstqualität, der dafür aufgewandte Energieverbrauch und die Anzahl der Ein/Ausschaltvorgänge herangezogen. Im Gegensatz zu anderen Untersuchungen auf diesem Gebiet, wurden die Ergebnisse auch mit einer optimalen Strategie, die ein allwissendes „Orakel" nutzt, verglichen. Die Bewertung der Ergebnisse wurde aus der Sicht verschiedener Clusterbetreiber-Strategien durchgeführt, welche die Metriken jeweils unterschiedlich stark gewichten. Die Strategien waren ein Hochleistungsbetrieb, ein energiesparsamer Betrieb, ein ausgewogener Betrieb (priorisiert Qualität des Dienstes und Energieverbrauch) und ein Betrieb mit alter Hardware (priorisiert minimale Anzahl von Ein/Ausschaltvorgängen).
Dabei wurde festgestellt, dass vor allem die Zeit für das Hochfahren einer Maschine einen großen Einfluss auf die Ergebnisse hat und schneller bootende Hardware immer bevorzugt eingesetzt werden sollte. Bei den untersuchten Parameterkonfigurationen lieferte, mit Ausnahme von einer einzigen Konfiguration, immer die schneller bootende Hardware das bessere Ergebnis (höheren *Score*). Hardwarehersteller sollten zukünftig Wert darauf legen, ihre Produkte so zu gestalten, dass eine möglichst schnelle Transition von einem sehr tiefen Schlafzustand (bzw. dem Abgeschaltetsein) in einen aktiven Zustand möglich ist. Auf diese Art können alle verwendeten Strategien deutlich optimiert werden.
Es wurde gezeigt, dass mit einem genügend großen Backup auch langsamere Hardware noch 10,6 % Energie einsparen und dabei eine Qualität des Dienstes von 92,3 % halten kann. Die Größe des Backups hat insgesamt einen etwa linearen Einfluss auf das Verhältnis zwischen Energieeinsparung und der Qualität des Dienstes. Weiterhin wurde nachgewiesen, dass ein aggressives Abschalten im vorliegenden Lastspitzen-Szenario bis zu 12,9 % mehr Energie gegenüber einem langsamen (Stück-für-Stück) Abschalten einsparen kann. Es wurde auch gezeigt, dass ein explizites Warten vor einem Bootbefehl überflüssig ist, da die Lastvorhersage ein *State-Flapping*, bereits verhindert.
Im Rahmen der Parameterstudie wurde ein Ranking für die beste Parametrisierung des Systems

aufgestellt. Das Ranking wurde jeweils für die vier vorgestellten Strategien erstellt. Es hat sich gezeigt, dass *CHERUB* in der Lage ist, bei einem Hochleistungsbetrieb Energieeinsparungen von 13,6 % (schnellste bootende Hardware) bei gleichzeitiger Qualität des Dienstes von 98,7 %, zu erzielen. Die Qualität des Dienstes entspricht dabei dem Niveau der optimalen Strategie (98,7 %) ohne den Einsatz von *CHERUB* und zeigt, dass mit guter Hardware eine energieeffiziente Clusterverwaltung ohne Qualitätseinbußen möglich ist. Bei einem besonders preiswerten Betrieb wurden Energieeinsparungen von bis zu 32,6 % bei gleichzeitiger Qualität des Dienstes von 84,6 % erzielt. In einem orakelgetriebenen Szenario mit vollständigem Wissen über die Last könnte bei maximaler Qualität des Dienstes 29,9 % Energie eingespart werden. Bei einem ausgewogenen Betrieb wurden Einsparungen von 28,1 % bei einer Qualität des Dienstes von 97 % erreicht und bei der Verwendung von älterer Hardware Einsparungen von 32,6 % bei einer Qualität des Dienstes von 84,6 %. Der ausgewogene Betrieb stellte bei den Strategien damit einen guten Kompromiss zwischen hoher Qualität des Dienstes und hohen Einsparungen dar.
Insgesamt zeigte die Studie, dass eine energieeffiziente Clusterverwaltung auch im Server-Load-Balancing-Bereich mit nur geringen Einbußen bei der Qualität des Dienstes möglich ist und dass die Verwendung von Lastvorhersage mit Hilfe von linearer Regression dabei klassischen, ausschließlich schwellwertbasierten Verfahren überlegen ist.

10.2 Weitere Forschungsthemen

Abschließend werden nun noch Themenfelder vorgestellt, die von zukünftigen Arbeiten betrachtet werden könnten.

10.2.1 Heterogene Umgebungen

Ein wichtiger Punkt, der nicht im Fokus dieser Arbeit lag, ist die Verwaltung von heterogenen Umgebungen. Rechenzentren besitzen häufig heterogene Cluster. Dies ist meist bedingt durch Finanzierungen, die die Kapazität eines Clusters durch neue Beschaffungen aufstockt. Dabei sind die beschafften Generationen häufig in sich homogen. Durch dieses Vorgehen entstehen in Clustern größere Gruppen von homogenen Maschinen.
Es gibt zwar schon Arbeiten, die heterogene Umgebungen berücksichtigen, z.B. [18, 113, 136], aber es wäre interessant zu ermitteln, wie gut die in dieser Arbeit vorgestellte Strategie eine solche Umgebung handhaben würde, nachdem sie dafür angepasst worden ist. Damit die Strategie dazu in der Lage ist, heterogene Cluster möglichst energieeffizient zu verwalten, wären die folgenden Maßnahmen notwendig.

Metriken Wie bereits erklärt, lassen sich Maschinen in größeren heterogenen Clustern innerhalb des Clusters in homogene Gruppen einordnen. Für diese Gruppen von homogenen Maschinen müsste jeweils bekannt sein, wie ihre Leistung (z.B. in Anf/s) und wie lang ihre Bootzeit ist. Um die Heterogenität auszunutzen, wäre es außerdem sinnvoll, die Leistung/Watt zu ermitteln.

Starten neuer Ressourcen Für das Starten neuer Ressourcen kann über die bekannten Metriken ermittelt werden, ob Maschinen gestartet werden müssen, und falls dem so ist, welche gestartet werden sollen. Grundsätzlich sollten zuerst Maschinen mit einer hohen Leistung/Watt gestartet

werden. Es wäre außerdem möglich, dass die Strategie abhängig von der Stärke der Steigung der aktuellen Last Maschinen startet, die eine passende Bootzeit aufweisen. Bei einem sehr starken Anstieg könnte dann z.B. eine sehr schnell bootende Maschine gewählt werden.

Herunterfahren von Ressourcen Umgekehrt kann über die Leistung/Watt in einer Unterlast-Situation entschieden werden, welche Maschine den größten Energiespareffekt erzielt, wenn sie abgeschaltet wird. Es könnte zusätzlich die Stärke der sinkenden Last in die Entscheidung einfließen, um zum Beispiel kurz zu warten, damit dann eine Maschine abgeschaltet werden kann, die zwar eine größere Leistung besitzt, aber eine schlechtere Leistung/Watt aufweist.

Zusammenfassend kann gesagt werden, dass für den heterogenen Fall vor allem das Zusammenspiel zwischen Leistung, Bootzeit und Leistung/Watt zwischen den verschiedenen homogenen Gruppen genau untersucht werden müsste. Von diesen drei Metriken hängt eine effiziente An- bzw. Abschaltentscheidung wesentlich ab.

10.2.2 Vorhersagetechniken

Obwohl die lineare Regression sich als sehr effektiv erwiesen hat, wäre es sinnvoll, noch weitere Verfahren für die Vorhersage der Last zu untersuchen. Resultate älterer Arbeiten deuten zwar eher darauf hin, dass mit einfachen Verfahren bessere Ergebnisse erzielt werden können (z.B. [13]), ein Versuch könnte trotzdem unternommen werden, ausgewählte Techniken auf ihre Vorhersagequalität hin zu untersuchen. Hier sollte auch immer auf die Menge der zur Vorhersage verwendeten Daten geachtet (z.B. Länge der verwendeten Historie) und deren Einfluss ermittelt werden.

10.2.3 Cloud Computing und Grid Computing

Nachdem das HPC-Feld und in dieser Arbeit das SLB-Feld betrachtet wurden, sind nun beide Bereiche des Cluster Computings umfangreich untersucht. Für ein vollständiges Bild fehlt die Analyse, ob *CHERUB* auch in den Bereichen Grid und/oder Cloud Computing eingesetzt werden könnte.

Grid Computing Da Grids sehr individuell ausfallen können, würde die einfachste und vermutlich effizienteste Lösung eine separate *CHERUB*-Installation bei jedem Partner des Grids darstellen. Die Partner eines Grids betreiben ihrerseits lokal Cluster Computing, welches, wie gezeigt, mit *CHERUB* gut handhabbar ist.

Cloud Computing Zum Cloud Computing wurden bereits in Kapitel 3 (Seite 37) Arbeiten vorgestellt. Diese zeigten aber eine Reihe von Nachteilen, wie beispielsweise den Einsatz einfacher Schwellwerte. Die Problematik eines Cloud-Anbieters, möglichst wenige physikalische Maschinen betreiben zu müssen, ähnelt der Problematik des SLB-Feldes stark. In ersten Untersuchungen könnte *CHERUB* verwendet werden, indem als Lastmetrik nicht mehr *Anfragen*, sondern z.B. *gleichzeitig laufende virtuelle Maschinen-Instanzen* verwendet wird. Handelt es sich in der Cloud um wenige große virtuelle Maschinen, wäre diese Metrik aber eher ungeeignet. In diesem Fall sollte auf Metriken wie die CPU- oder Speicher-Auslastung zurückgegriffen werden. Hier hängt vieles von der konkreten Umsetzung der Cloud ab (z.B. erhält jede VM ihre eigene CPU, ist der

Dienst leicht oder schwergewichtig etc.). Genauer untersucht werden müsste beim Cloud Computing auch, wann und in welchem Maße Migrationen von virtuellen Maschinen stattfinden müssten.

10.2.4 Scheduling

Konkrete Pläne für eine weitere Studie mit Hilfe von *ClusterSim* bestehen bereits. Diese Studie soll Scheduling-Verfahren vergleichen, die Jörg Jung in seiner Dissertation [173] entwickelt und untersucht hat. Die Studie soll dazu dienen, die Skalierung der Verfahren nachzuweisen und die in der Dissertation durchgeführten Messungen zu validieren.
Jörg Jung hat zwei Lastverteilungs-Verfahren entwickelt, welche neben den bestehenden Verfahren (Siehe Abschnitt 2.2.1.2, Seite 18) im Rahmen seiner Dissertation ebenfalls in LVS implementiert wurden. Dabei handelt es sich um die beiden Verfahren **Highest Credits (HC)** und **Next Credits (NC)**. Um diese beiden Verfahren nutzen zu können, müssen die *Real Server* angepasst werden. Nach der Anpassung senden diese in regelmäßigen Abständen sogenannte *Credits* an den *Dispatcher*. Diese spiegeln ihre Bereitschaft wider, neue Anfragen entgegennehmen zu können. Dabei gibt es *Soft Credits* und *Hard Credits*. *Hard Credits* spiegeln eins-zu-eins die freien Plätze in der TCP-Backlog-Queue (siehe Abschnitt 2.2.5, Seite 30) des entsprechenden *Real Servers* wider. *Soft Credits* entsprechen dem Median der letzten 16 gemeldeten *Hard Credits*. HC wählt bei einer eingehenden Anfrage den *Real Server* mit den meisten *Soft Credits* aus. Gibt es keinen *Real Server* mit *Soft Credits*, wird der *Real Server* mit den meisten *Hard Credits* ausgewählt. Hat kein *Real Server* Credits übrig, wird die Anfrage direkt am *Dispatcher* verworfen. NC wählt, ähnlich wie RR, reihum den nächsten *Real Server* mit übrigen *Soft Credits* aus. Ist kein *Real Server* mit *Soft Credits* übrig, verfährt er weiter reihum mit denen, die *Hard Credits* übrig haben. Sind keine Credits mehr übrig, wird die Anfrage ebenfalls verworfen.
ClusterSim, welcher in Abschnitt 8 (Seite 107) vorgestellt wurde, wurde um diese beiden Algorithmen bereits erweitert.

Anhang

Anhang A: Messergebnisse

Dieser Anhang beinhaltet die ausführlichen Messergebnisse für alle größeren Messungen, die innerhalb der Arbeit auf Grund der Übersichtlichkeit und der besseren Erfassbarkeit nur als Grafik dargestellt wurden. Tabelle A.1 fasst zusammen, welche Tabellen im Anhang zu welchen Abschnitten in der Arbeit gehören.

Abschnitt in Arbeit	Seite	Tabelle	Seite
8.8	129	A.2	161
3. Experiment		A.3	162
		A.4	163
		A.5	164
8.8	131 ff	A.6	165
4. Experiment		A.7	166
9.5	141 ff	A.8	167
		A.9	168

Tabelle A.1: Übersicht des Anhangs.

Länge der BLQ	QoS in %		AD in Sekunden	
	Messung	Simulation	Messung	Simulation
1	6,63	41,87	3,405	0,720
2	16,52	50,25	3,719	2,100
3	75,06	58,62	0,722	0,720
4	78,84	67,0	1,079	0,720
5	83,19	71,15	0,802	1,080
6	87,51	75,31	0,803	1,080
7	91,66	79,47	0,804	1,080
8	92,33	83,63	0,887	1,080
9	85,40	87,69	1,024	0,880
10	91,95	91,84	1,087	0,960
11	92,18	92,75	1,169	1,040
12	92,29	92,80	1,243	1,200
13	92,18	92,81	1,341	1,240
14	92,11	92,83	1,433	1,360
15	92,19	92,84	1,521	1,440
16	92,16	92,86	1,616	1,560
17	92,08	92,87	1,699	1,600
18	92,30	92,88	1,793	1,720
19	92,26	92,90	1,875	1,800
20	81,04	92,91	2,054	1,920
21	92,27	92,93	2,075	1,920
22	92,40	92,94	2,147	2,080
23	91,79	92,95	2,238	2,120
24	92,12	92,97	2,339	2,280
25	92,16	92,98	2,412	2,280
26	92,65	93,0	2,513	2,440
27	92,37	93,01	2,596	2,480
28	92,51	93,02	2,688	2,600
29	92,72	93,04	2,767	2,640
30	92,48	93,05	2,875	2,760
31	92,48	93,06	2,962	2,840
32	92,31	93,08	3,040	2,960
33	92,44	93,09	3,143	3,000
34	92,33	93,11	3,235	3,120
35	92,22	93,12	3,313	3,200
36	91,79	93,13	3,402	3,320
37	92,08	93,15	3,514	3,360
38	92,0	93,16	3,60	3,480
39	92,22	93,18	3,695	3,520
40	92,77	93,19	3,763	3,680

Tabelle A.2: Erste Hälfte der Ergebnisse der BLQ Messung. Vergleich zwischen Messung und Simulation mit 4 Workern.

Länge der BLQ	QoS in %		AD in Sekunden	
	Messung	Simulation	Messung	Simulation
41	91,95	93,20	3,844	3,720
42	92,25	93,22	3,92	3,840
43	92,31	93,23	4,020	3,880
44	92,29	93,25	4,10	4,040
45	91,93	93,26	4,209	4,080
46	91,84	93,27	4,289	4,200
47	92,18	93,29	4,370	4,240
48	89,16	93,30	4,454	4,360
49	86,80	91,61	4,514	4,400
50	79,76	89,91	4,544	4,520
51	71,38	88,20	4,601	4,560
52	63,30	86,5	4,66	4,680
53	56,27	78,84	4,705	4,640
54	48,75	71,19	4,72	4,720
55	40,13	63,54	4,788	4,680
56	34,16	55,88	4,820	4,760
57	26,62	48,29	4,845	4,720
58	21,13	40,69	4,837	4,800
59	16,95	33,09	4,801	4,800
60	11,23	25,5	3,61	4,840
61	8,47	21,30	2,931	4,800
62	7,80	17,11	2,950	4,800
63	8,5	12,91	2,94	3,920
64	8,77	8,72	3,008	2,880
65	8,05	8,72	3,027	2,880
66	8,05	8,72	2,935	2,880
67	8,43	8,72	2,922	2,880
68	8,05	8,72	3,122	2,880
69	8,41	8,72	2,924	2,880
70	8,0	8,72	2,82	2,880
71	8,41	8,72	2,928	2,880
72	8,37	8,72	2,935	2,880
73	8,38	8,72	2,918	2,880
74	8,40	8,72	2,920	2,880
75	8,38	8,72	2,911	2,880
76	8,38	8,72	2,903	2,880
77	8,38	8,72	2,792	2,880
78	8,11	8,72	2,836	2,880
79	8,43	8,72	2,919	2,880
80	8,27	8,72	2,883	2,880

Tabelle A.3: Zweite Hälfte der Ergebnisse der BLQ Messung. Vergleich zwischen Messung und Simulation mit 4 Workern.

Länge der BLQ	QoS in %		AD in Sekunden	
	Messung	Simulation	Messung	Simulation
1	72,66	71,13	0,883	1,080
2	77,01	75,30	3,503	1,080
3	82,0	79,44	0,73	1,080
4	86,91	83,61	0,795	1,080
5	90,66	84,94	0,861	1,080
6	91,47	86,58	0,919	1,080
7	91,97	87,77	0,965	1,420
8	92,36	89,16	1,113	1,180
9	92,40	91,81	1,252	1,180
10	92,36	92,75	1,29	1,300
11	92,48	92,77	1,426	1,460
12	92,37	92,88	1,514	1,780
13	92,44	92,83	1,605	1,680
14	92,48	92,80	1,685	1,680
15	92,38	92,84	1,799	1,820
16	92,51	92,83	1,854	1,860
17	92,45	92,84	1,975	1,940
18	92,62	92,86	2,059	2,040
19	92,55	92,87	2,143	2,120
20	92,59	92,88	2,212	2,260
21	92,70	92,90	2,337	2,320
22	92,40	92,91	2,377	2,460
23	92,44	92,93	2,487	2,500
24	92,68	92,94	2,584	2,560
25	92,54	92,95	2,681	2,640
26	92,41	92,97	2,779	2,760
27	92,54	92,98	2,863	2,800
28	92,40	93,0	2,92	2,960
29	92,56	93,01	3,032	3,060
30	92,54	93,02	3,135	3,200
31	92,54	93,04	3,226	3,220
32	92,45	93,05	3,280	3,280
33	92,44	93,06	3,391	3,340
34	92,51	93,08	3,483	3,460
35	92,54	93,09	3,584	3,520
36	92,56	93,11	3,677	3,660
37	92,62	93,12	3,752	3,740
38	92,36	93,13	3,838	3,880
39	92,83	93,15	3,924	3,920
40	92,59	93,16	4,005	3,980

Tabelle A.4: Erste Hälfte der Ergebnisse der BLQ Messung. Vergleich zwischen Messung und Simulation mit 8 Workern.

Länge der BLQ	QoS in %		AD in Sekunden	
	Messung	Simulation	Messung	Simulation
41	92,63	93,18	4,093	4,060
42	92,41	93,19	4,205	4,160
43	92,19	93,20	4,291	4,200
44	92,38	93,22	4,376	4,360
45	90,56	91,94	4,433	4,440
46	85,75	90,66	4,50	4,560
47	78,08	89,38	4,560	4,620
48	69,70	88,11	4,622	4,660
49	60,37	80,86	4,679	4,680
50	53,79	73,61	4,712	4,740
51	46,37	65,93	4,759	4,720
52	38,69	58,25	4,79	4,760
53	31,13	50,59	4,829	4,760
54	23,75	42,95	4,85	4,780
55	16,25	35,30	4,83	4,800
56	9,68	27,66	3,415	4,840
57	7,69	23,83	2,985	4,840
58	8,0	20,02	2,92	4,820
59	8,05	16,59	2,983	4,800
60	7,37	13,16	2,907	4,240
61	8,06	11,90	2,903	3,960
62	8,12	10,63	3,186	3,620
63	8,04	9,37	3,032	3,340
64	7,94	8,11	2,972	3,050
65	7,97	8,11	3,026	3,050
66	7,97	8,11	2,875	3,050
67	7,47	8,11	2,994	3,050
68	8,01	8,11	2,965	3,050
69	7,98	8,11	2,905	3,050
70	7,27	8,11	2,944	3,050
71	7,88	8,11	2,947	3,050
72	8,19	8,11	3,038	3,050
73	6,52	8,11	3,481	3,050
74	8,06	8,11	2,982	3,050
75	7,88	8,11	2,920	3,050
76	8,05	8,11	2,981	3,050
77	7,95	8,11	2,920	3,050
78	8,0	8,11	2,94	3,050
79	8,0	8,11	2,90	3,050
80	7,93	8,11	2,920	3,050

Tabelle A.5: Zweite Hälfte der Ergebnisse der BLQ Messung. Vergleich zwischen Messung und Simulation mit 8 Workern.

Länge der BLQ	Anf/s	QoS in %		AD in Sekunden	
		Messung	Simulation	Messung	Simulation
20	4	100	100	0,362	0,36
20	8	100	100	0,641	0,60
20	12	92,32	92,96	1,979	2,12
20	16	69,36	69,91	2,146	2,52
20	20	55,52	55,92	2,162	2,52
40	4	100	100	0,362	0,36
40	8	100	100	0,641	0,60
40	12	92,40	93,24	3,754	3,88
40	16	68,29	70,00	3,947	4,30
40	20	53,76	56,00	4,001	4,32
60	4	100	100	0,362	0,36
60	8	100	100	0,642	0,60
60	12	12,24	12,92	3,872	3,92
60	16	3,83	5,00	4,832	4,84
60	20	2,76	3,63	4,861	4,84
80	4	100	100	0,362	0,36
80	8	100	100	0,603	0,60
80	12	8,44	8,72	2,901	2,88
80	16	1,58	1,58	2,938	2,92
80	20	0,87	0,87	2,863	2,84

Tabelle A.6: Vergleich zwischen Messung und Simulation mit verschiedenen Parametereinstellungen und bei 4 Workern. Verwendet wurde nur die frühe Überprüfung der BLQ.

Länge der BLQ	Anf/s	QoS in %		AD in Sekunden	
		Messung	Simulation	Messung	Simulation
20	4	100	100	0,362	0,36
20	8	100	100	0,702	0,72
20	12	92,53	92,93	2,242	2,50
20	16	69,48	69,91	2,424	2,94
20	20	55,65	55,92	2,452	2,94
40	4	100	100	0,362	0,36
40	8	100	100	0,702	0,72
40	12	92,51	93,21	4,013	4,20
40	16	68,60	68,92	4,189	4,86
40	20	54,29	55,11	4,237	4,86
60	4	100	100	0,362	0,36
60	8	100	100	0,702	0,72
60	12	8,09	9,38	2,944	3,34
60	16	1,54	3,21	2,970	4,80
60	20	0,84	2,21	2,920	4,92
80	4	100	100	0,362	0,36
80	8	100	100	0,701	0,72
80	12	8,03	8,11	2,967	3,05
80	16	1,5	1,50	2,906	3,04
80	20	0,84	0,83	2,946	3,00

Tabelle A.7: Vergleich zwischen Messung und Simulation mit verschiedenen Parametereinstellungen und bei 8 Workern. Verwendet wurde nur die frühe Überprüfung der BLQ.

Seq.Abs.	Warten	Bootdauer	Backup in %	EV in Wh	EE in %	AD in s	QoS in %	PZW
0	0	5	0	2137,84	33,49	1,430	91,14	188
0	0	5	5	2231,33	30,59	1,012	94,79	196
0	0	5	10	2311,71	28,08	0,574	96,99	199
0	0	5	25	2513,62	21,80	0,135	98,30	247
0	0	30	0	2112,10	34,29	1,595	89,02	171
0	0	30	5	2237,74	30.39	1,025	93,23	187
0	0	30	10	2366,73	26,37	0,619	96,00	219
0	0	30	25	2561,59	20,31	0,128	98,22	190
0	0	60	0	2172,41	32,42	1,537	88,85	191
0	0	60	5	2319,91	27,83	0,837	93,00	209
0	0	60	10	2376,07	26,08	0,657	94,73	211
0	0	60	25	2603,49	19,01	0,137	97,24	222
0	0	120	0	2291,30	28,72	1,120	88,08	206
0	0	120	5	2439,36	24,11	0,751	90,93	266
0	0	120	10	2446,37	23,90	0,500	91,88	225
0	0	120	25	2657,98	17,31	0,158	94,50	238
0	0	180	0	2323,54	27,72	1,084	83,81	218
0	0	180	5	2399,03	25.37	0,583	86,77	210
0	0	180	10	2474,32	23,03	0,448	88,67	212
0	0	180	25	2689,39	16,34	0,162	92,00	214
0	1	5	0	1909,78	40,59	1,952	82,12	138
0	1	5	5	2019,81	37,17	1,513	86,33	147
0	1	5	10	2130,79	33,71	1,009	89,78	153
0	1	5	25	2402,69	25,25	0,229	96,49	154
0	1	30	0	1969,44	38,73	1,428	82,91	139
0	1	30	5	2038,92	36,57	1,365	85,82	146
0	1	30	10	2145,51	33,26	1,042	89,12	157
0	1	30	25	2440,45	24,08	0,213	95,77	178
0	1	60	0	1953,90	39,22	1,887	80,00	159
0	1	60	5	2083,74	35,18	1,284	85,45	160
0	1	60	10	2145,88	33,24	1,124	86,68	169
0	1	60	25	2443,31	24,00	0,234	94,05	158
0	1	120	0	2038,37	36,59	1,577	79,59	167
0	1	120	5	2163,23	32,70	1,118	84,28	176
0	1	120	10	2245,66	30,14	0,867	85,74	170
0	1	120	25	2498,02	22,29	0,211	92,07	174
0	1	180	0	2113,57	34,25	1,451	78,00	180
0	1	180	5	2216,56	31,04	0,951	82,15	179
0	1	180	10	2290,76	28,74	0,626	84,66	173
0	1	180	25	2548,52	20,72	0,227	89,53	181

Tabelle A.8: Erste Hälfte aller Daten der Simulation aus Abschnitt 9.5. Die einsparte Energie wurde auf Basis des in Tabelle 9.5 (Seite 141) markierten Energieverbrauchs von 3214,5 Wh berechnet.

Seq.Abs.	Warten	Bootdauer	Backup in %	EV in Wh	EE in %	AD in s	QoS in %	PZW
1	0	5	0	2398,38	25,39	0,475	92,22	106
1	0	5	5	2575,26	19,89	0,198	95,36	123
1	0	5	10	2603,61	19,00	0,130	97,18	123
1	0	5	25	2776,27	13,63	0,067	98,70	123
1	0	30	0	2421,13	24,68	0,413	90,94	113
1	0	30	5	2592,76	19,34	0,162	93,81	120
1	0	30	10	2709,61	15,71	0,117	96,72	136
1	0	30	25	2783,10	13,42	0,078	97,77	135
1	0	60	0	2503,11	22,13	0,267	90,43	116
1	0	60	5	2693,26	16,22	0,127	94,16	123
1	0	60	10	2725,06	15,23	0,109	95,96	121
1	0	60	25	2846,70	11,44	0,078	97,16	159
1	0	120	0	2705,04	15,85	0,082	90,66	114
1	0	120	5	2749,33	14,47	0,090	91,81	118
1	0	120	10	2734,72	14,93	0,084	92,59	135
1	0	120	25	2892,39	10,02	0,080	94,63	156
1	0	180	0	2658,51	17,30	0,087	86,90	130
1	0	180	5	2761,10	14,10	0,072	88,24	126
1	0	180	10	2832,31	11,89	0,072	89,06	137
1	0	180	25	2872,34	10,64	0,080	92,32	148
1	1	5	0	2166,53	32,60	0,548	84,56	94
1	1	5	5	2325,59	27,65	0,363	87,72	99
1	1	5	10	2390,84	25,62	0,317	90,54	103
1	1	5	25	2687,88	16,38	0,113	96,67	111
1	1	30	0	2199,23	31,58	0,430	83,42	89
1	1	30	5	2318,39	27,88	0,361	86,28	98
1	1	30	10	2394,58	25,51	0,351	89,34	98
1	1	30	25	2706,62	15,80	0,103	95,90	111
1	1	60	0	2246,20	30,12	0,401	83,13	105
1	1	60	5	2367,62	26,35	0,366	86,27	110
1	1	60	10	2530,78	21,27	0,212	89,55	114
1	1	60	25	2714,67	15,55	0,114	94,04	115
1	1	120	0	2257,95	29,76	0,526	82,30	124
1	1	120	5	2383,83	25,84	0,342	85,05	131
1	1	120	10	2439,11	24,12	0,266	87,30	133
1	1	120	25	2672,30	16,87	0,117	91,17	127
1	1	180	0	2422,15	24.65	0,158	80,66	127
1	1	180	5	2520,44	21,59	0,119	83,10	122
1	1	180	10	2624,92	18,34	0,096	85,37	114
1	1	180	25	2762,27	14,07	0,086	89,56	131

Tabelle A.9: Zweite Hälfte aller Daten der Simulation aus Abschnitt 9.5. Die einsparte Energie wurde auf Basis des in Tabelle 9.5 (Seite 141) markierten Energieverbrauchs von 3214,5 Wh berechnet.

Glossar

ACPI - Advanced Configuration & Power Interface Offener Industriestandard, siehe Abschnitt 2.1.1, Seite 9.

AD - Anfragedauer Die Anfragedauer umfasst, im Gegensatz zur FRT, die gesamte Zeit des Lebenszykluses einer Anfrage. Die umfasst damit die Netzwerkzeit, die Verweilzeit und die Bearbeitungszeit.

Anf - Anfragen In Anfragen wird die aktuelle Last gemessen und Schwellwerte werden in dieser Einheit Angegeben.

Anf/s - Anfragen pro Sekunde Anfragen pro Sekunde ist die Einheit, in der z.B. die Steigung der Last (resultierend aus Formel 6.7, Seite 87), die Leistung der Maschinen oder die durchschnittliche Last eines Traces vorliegen.

Arbeitslast Die für ein Experiment/Messung verwendete Last. Im Rahmen dieser Arbeit handelt es sich immer um ein Webserver-Log (siehe auch *Trace*), in dem Anfragen protokolliert sind, die an einem Webserver eingegangen sind.

BLQ - TCP-Backlog-Queue Die Warteschlange, in der TCP-Pakete vor Abholung durch die Applikation zwischengelagert werden. Für mehr Details siehe Abschnitt 2.2.5, Seite 30.

BPF - Berkeley Packet Filter Bestandteil des Kernels, welcher eingehende Netzwerk-Pakete nach verschiedensten Kriterien filtern kann. Siehe Abschnitt 2.2.4, Seite 30.

DVFS - Dynamic Voltage and Frequence Scaling Bezeichnet die Fähigkeit, Spannung und Takt einer Komponente zu verändern. Siehe Abschnitt 2.1.4, Seite 12.

ESS - Energiesparsystem Ein Energiesparsystem ist eine Software, die im Clusterumfeld, eingesetzt wird, um Energie zu sparen. Das ESS, welches in dieser Arbeit verwendet wird, heißt *CHERUB* und wird in Abschnitt 2.2.3 auf Seite 24 beschrieben.

FRT - First Response Time Englisch für Antwortzeit. Unter der FRT wird die Zeitspanne verstanden, die es benötigt, um das erste Bit einer Antwort eines angefragten Servers zu empfangen, nachdem eine Anfrage beim Client an den entsprechenden Server gesendet wurde. Die FRT entspricht somit nicht der Anfragedauer (AD), da die FRT nicht die volle Bearbeitungszeit einer Anfrage berücksichtigt.

IPMI - Intelligent Platform Management Interface Industriestandard, siehe Abschnitt 2.1.1, Seite 9.

QoS - Quality of Service Englisch für Qualität des Dienstes. Wird als Synonym für das Service Level Agreement verwendet und ist die primäre Metrik für die wichtigen Messungen dieser Arbeit. Bei einer QoS von 90 % wurden 90 % aller Anfragen innerhalb von 5 Sekunden bearbeitet.

RMS - Ressourcen-Management-System Bei einem Ressourcen-Management-System handelt es sich um die Software, die ein Cluster erst einsatzbereit macht. Sie kümmert sich normalerweise um die Verteilung der anfallenden Last und überwacht die einzelnen Maschinen. Populäre Beispiele sind u.a. das Portable Batch System (PBS), TORQUE/MAUI, IBM LoadLeveler oder Linux Virtual Machine (LVS).

Setup Unter diesem Begriff wird die verwendete Hardware und Software, die für ein Experiment genutzt wird, zusammengefasst.

SLA - Service Level Agreement Englisch für Leistungsvertrag. Wird als Synonym für die Quality of Service verwendet. Siehe QoS - Quality of Service.

STR - Suspend-to-RAM Als Suspend-to-RAM wird der ACPI (siehe Abschnitt 2.1.1, Seite 9) Schlafzustand S3 bezeichnet. In diesem Zustand ist die Maschine abgeschaltet und hat sämtlichen Systemkontext im RAM gespeichert und hält diesen dort. Bei einem Neustart wird der Zustand nur aus dem RAM geladen, was einen erheblichen Geschwindigkeitsvorteil bringt.

Szenario Unter diesem Begriff ist in dieser Arbeit die verwendete Arbeitslast und das verwendete Setup eines Experimentes zusammengefasst.

Trace Als Trace wird ein Verlaufsprotokoll bezeichnet. In dieser Arbeit bezieht sich der Begriff Trace immer auf das Verlaufsprotokoll eines Webservers. Das Log (engl. Protokoll) des Webservers wird hier als Trace bezeichnet.

WOL - Wake on Lan Ein Standard, der das *Magic Packet* definiert, welches zum entfernten Anschalten von Rechnern genutzt werden kann. Siehe Abschnitt 2.1.3, Seite 11.

ZZNE - Zeit zum nächsten Ereignis Die ZZNE ist in *ClusterSim* die Zeit die es dauert, bis das nächste Ereignis bearbeitet werden muss. *ClusterSim* schreitet immer um die ZZNE voran, Arbeitet das entsprechende Ereignis ab und passt die Zeiten aller anderen Ereignisse an. Bei der ZZNE handelt es sich immer um das Minimum von entweder dem kleinsten noch verfügbare Quantum der aktiven Worker oder um die kürzeste Zeit, die eine aktive Anfrage noch zur Fertigstellung benötigt..

Index

Literaturverzeichnis

[1] Top500 Supercomputers. http://www.top500.org/, 2016. letzter Zugriff März 2016.

[2] Rich Miller. Who Has the Most Web Servers? http://www.datacenterknowledge.com/archives/2009/05/14/whos-got-the-most-web-servers/. letzter Zugriff März 2016.

[3] Jonathan G Koomey. Worldwide electricity used in data centers. *Environmental Research Letters*, 3(3):034008, 2008.

[4] Chandrakant D. Patel, Cullen E. Bash, Ratnesh Sharma, Monem Beitelmal, and Rich Friedrich. Smart Cooling Of Data Centers. Patent US 6,574,104 B2, https://www.lens.org/lens/patent/US_6574104_B2/fulltext, 06 2003.

[5] Ali Pahlavan, Mahmoud Momtazpour, and Maziar Goudarzi. Power reduction in HPC data centers: a joint server placement and chassis consolidation approach. *The Journal of Supercomputing*, 70(2):845–879, 2014.

[6] Jonathan Koomey and Jon Taylor. New data supports finding that 30 percent of servers are 'Comatose', indicating that nearly a third of capital in enterprise data centers is wasted. Technical report, University of Standord, Anthesisgroup, 2015.

[7] James M. Kaplan, William Forrest, and Noah Kindle. Revolutionizing Data Center Energy Efficiency. Technical report, McKinsey & Company, July 2008.

[8] International Atomic Energy Agency. Power Reactor Information System. https://www.iaea.org/PRIS/CountryStatistics/CountryDetails.aspx?current=FR. letzter Zugriff Mai 2016.

[9] Luiz Andre Barroso and Urs Hölzle. The Case for Energy-Proportional Computing. *Computer*, 40(12):33–37, Dec 2007.

[10] David Meisner, Brian T. Gold, and Thomas F. Wenisch. PowerNap: Eliminating Server Idle Power. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIV, pages 205–216, New York, NY, USA, 2009. ACM.

[11] Die Europäische Kommission. Verordnung (EU) Nr. 617/2013 der Kommission. http://eur-lex.europa.eu/legal-content/DE/TXT/?qid=1464617832266&uri=CELEX:32013R0617. letzter Zugriff Mai 2016.

[12] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for Reduced CPU Energy. In *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, OSDI '94, Berkeley, CA, USA, 1994. USENIX Association.

[13] Kinshuk Govil, Edwin Chan, and Hal Wasserman. Comparing Algorithm for Dynamic Speed-setting of a Low-power CPU. In *Proceedings of the 1st Annual International Conference on Mobile Computing and Networking*, MobiCom '95, pages 13–25, New York, NY, USA, 1995. ACM.

[14] Frances Yao, Alan Demers, and Scott Shenker. A Scheduling Model for Reduced CPU Energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, FOCS '95, pages 374–382, Washington, DC, USA, 1995. IEEE Computer Society.

[15] Josep Ll. Berral, Íñigo Goiri, Ramón Nou, Ferran Julià, Jordi Guitart, Ricard Gavaldà, and Jordi Torres. Towards Energy-aware Scheduling in Data Centers Using Machine Learning. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, e-Energy '10, pages 215–224, New York, NY, USA, 2010. ACM.

[16] Qi Zhang, Mohamed Faten Zhani, Shuo Zhang, Quanyan Zhu, Raouf Boutaba, and Joseph L. Hellerstein. Dynamic Energy-aware Capacity Provisioning for Cloud Computing Environments. In *Proceedings of the 9th International Conference on Autonomic Computing*, ICAC '12, pages 145–154, New York, NY, USA, 2012. ACM.

[17] Wesam Dawoud. *Scalability and Performance Management of Internet Applications in the Cloud*. PhD thesis, Faculty of Mathematics and Natural Sciences, University of Potsdam, 3 2013.

[18] Jörg Lenhardt, Wolfram Schiffmann, Patrick Eitschberger, and Jörg Keller. Power-Efficient Load Distribution in Heterogeneous Computing Environments. In *Proc. 12th IASTED International Conference on Parallel and Distributed Computing and Networks*, Innsbruck, Austria, February 2014.

[19] Akihiko Miyoshi, Charles Lefurgy, Eric Van Hensbergen, Ram Rajamony, and Raj Rajkumar. Critical Power Slope: Understanding the Runtime Effects of Frequency Scaling. In *Proceedings of the 16th International Conference on Supercomputing*, ICS '02, pages 35–44, New York, NY, USA, 2002. ACM.

[20] Simon Kiertscher. Green IT – Energiebewusstes Clustermanagement. Diplomarbeit, Insitut für Informatik, Universität Potsdam, 2010.

[21] Simon Kiertscher, Jörg Zinke, Stefan Gasterstädt, and Bettina Schnor. Cherub: Power consumption aware cluster resource management. In *2010 IEEE/ACM International Conference on Green Computing and Communications & 2010 IEEE/ACM International Conference on Cyber, Physical and Social Computing*, Hangzhou, China, 2010. IEEE Computer Society Conference Publications Operations Committee. ISBN: 978-0-7695-4331-4, BMS Number: CFP10GCC-CDR.

[22] Simon Kiertscher, Jörg Zinke, and Bettina Schnor. Cherub: power consumption aware cluster resource management. *Cluster Computing*, pages 1–9, 2011. 10.1007/s10586-011-0176-5.

[23] Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., and Toshiba Corporation. Advanced Configuration and Power Interface Specification (Version 6). http://www.uefi.org/sites/default/files/resources/ACPI_6.0.pdf, April 2015. letzter Zugriff März 2016.

[24] Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., and Toshiba Corporation. Advanced Configuration and Power Interface Specification (Version 5a). http://www.acpi.info/spec50a.htm, November 2013. letzter Zugriff März 2016.

[25] Intel, Hewlett-Packard, NEC, and Dell. Intelligent Platform Management Interface Specification. http://www.intel.com/design/servers/ipmi/spec.htm. letzter Zugriff März 2016.

[26] Duncan Laurie. IPMItool. http://ipmitool.sourceforge.net/. letzter Zugriff Juli 2011.

[27] AMD und Hewlett-Packard. Magic Packet Technology. http://support.amd.com/TechDocs/20213.pdf, 1995. letzter Zugriff März 2016.

[28] Etienne Le Sueur and Gernot Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, HotPower'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

[29] Intel Corporation. Enhanced Intel® SpeedStep® Technology for the Intel® Pentium® M Processor . http://download.intel.com/design/network/papers/30117401.pdf, 2004. White-Paper, letzter Zugriff März 2016.

[30] AMD. AMD PowerNow!™ Technology. http://support.amd.com/TechDocs/24404a.pdf, 2000. White-Paper, letzter Zugriff März 2016.

[31] AMD. Cool 'n' Quiet™ Technology Installation Guide for AMD Athlon™ 64 Processor Based Systems. http://www.amd.com/Documents/Cool_N_Quiet_Installation_Guide3.pdf, 2004. Installation Guide, letzter Zugriff März 2016.

[32] AMD. AMD POWERTUNE TECHNOLOGY. https://www.amd.com/Documents/amd_powertune_whitepaper.pdf, 2012. White-Paper, letzter Zugriff März 2016.

[33] AMD. AMD PowerTune Technology. http://www.amd.com/Documents/PowerTune_Technology_Whitepaper.pdf, 2010. White-Paper, letzter Zugriff März 2016.

[34] Intel. Intel® Turbo Boost Technology in Intel® Core™ Microarchitecture (Nehalem) Based Processors . http://files.shareholder.com/downloads/INTC/0x0x348508/C9259E98-BE06-42C8-A433-E28F64CB8EF2/TurboBoostWhitePaper.pdf, 2008. White-Paper, letzter Zugriff März 2016.

[35] AMD. ADVANCED POWER MANAGEMENT HELPS BRING IMPROVED PERFORMANCE TO HIGHLY INTEGRATED X86 PROCESSORS. https://www.amd.com/

Documents/White_paper_layout_and_design.pdf, 2014. White-Paper, letzter Zugriff März 2016.

[36] Murali. ASIC-System on Chip-VLSI Design - Clock Gating. http://asic-soc.blogspot.de/2008/04/clock-gating.html, 2008. Tech-Blog, letzter Zugriff April 2016.

[37] Murali. ASIC-System on Chip-VLSI Design - Power Gating. http://asic-soc.blogspot.de/2008/04/power-gating.html, 2008. Tech-Blog, letzter Zugriff April 2016.

[38] Dominik Brodowski and Nico Golde. Linux CPUFreq - CPUFreq Governors. https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt. letzter Zugriff Mai 2016.

[39] Ubuntu Wiki. Prozessortaktung. https://wiki.ubuntuusers.de/Prozessortaktung/. letzter Zugriff Mai 2016.

[40] Wensong Zhang et al. Linux Virtual Server for Scalable Network Services. *Ottawa Linux Symposium*, 2000.

[41] Wensong Zhang and Wenzhuo Zhang. Linux Virtual Server Clusters: Build highly-scalable and highly-available network services at low cost. *Linux Magazine*, November 2003.

[42] The Linux Virtual Server Project. Linux Server Cluster for Load Balancing. http://www.linuxvirtualserver.org. letzter Zugriff April 2016.

[43] Joseph Mack. LVS: What is an LVS? Can I use an LVS? http://www.austintek.com/LVS/LVS-HOWTO/HOWTO/index.html. letzter Zugriff April 2016.

[44] Red Hat Inc. Piranha - Load-balanced generic service clustering enviroment. https://sourceware.org/piranha/. letzter Zugriff April 2016.

[45] Keepalived - Loadbalancing & High-Availability. http://www.keepalived.org/. letzter Zugriff April 2016.

[46] Simon Horman. Ultra Monkey - Load Balancing and High Availability Solution. http://www.ultramonkey.org/. letzter Zugriff April 2016.

[47] Florian Haas. Heartbeat. http://www.linux-ha.org/wiki/Heartbeat. letzter Zugriff April 2016.

[48] Jim Trocki. mon. https://www.kernel.org/pub/software/admin/mon/html/man/mon.html. letzter Zugriff April 2016.

[49] Systems group of M. Satyanarayanan. Coda File System. http://www.coda.cs.cmu.edu/. letzter Zugriff April 2016.

[50] Simon Horman and Jacob Rief. ldirectord. http://horms.net/projects/ldirectord/. letzter Zugriff April 2016.

[51] The Linux Virtual Server Project. High Availability. http://www.linuxvirtualserver.org/HighAvailability.html. letzter Zugriff April 2016.

[52] The Linux Virtual Server Project. Virtual Server via NAT. http://www.linuxvirtualserver.org/VS-NAT.html. letzter Zugriff Februar 2016.

[53] The Linux Virtual Server Project. Virtual Server via IP Tunneling. http://www.linuxvirtualserver.org/VS-IPTunneling.html. letzter Zugriff Februar 2016.

[54] The Linux Virtual Server Project. ARP problem in VS/TUN and VS/DR. http://www.linuxvirtualserver.org/docs/arp.html. letzter Zugriff April 2016.

[55] The Linux Virtual Server Project. Virtual Server via Direct Routing. http://www.linuxvirtualserver.org/VS-DRouting.html. letzter Zugriff Februar 2016.

[56] Squid: Optimising Web Delivery. http://www.squid-cache.org/. letzter Zugriff April 2016.

[57] The Apache Software Foundation. Apache http server project. https://httpd.apache.org/, 1995.

[58] Q-Success. W3techs - world wide web technology surveys. http://w3techs.com/technologies/overview/web_server/all, 2016.

[59] Netcraft. March 2016 web server survey. http://news.netcraft.com/archives/category/web-server-survey/, 2016.

[60] The Apache Software Foundation. Apache mpm prefork. http://httpd.apache.org/docs/2.2/mod/prefork.html, 2016.

[61] The Apache Software Foundation. Apache mpm worker. http://httpd.apache.org/docs/2.2/mod/worker.html, 2016.

[62] The Apache Software Foundation. Apache mpm event. http://httpd.apache.org/docs/2.4/mod/event.html, 2016.

[63] Sijing You. Analyse der Lastverteilungsverfahren des Apache HTTP-Servers. Bachelorarbeit, Insitut für Informatik und Compuational Science, Universität Potsdam, 2014. http://www.cs.uni-potsdam.de/bs/teaching/docs/thesis/2014/you.pdf.

[64] tcpdump.org. TCPDUMP & LIBPCAP. http://www.tcpdump.org. letzter Zugriff April 2016.

[65] Van Jacobson, Craig Leres and Steven McCanne, all of the Lawrence Berkeley National Laboratory. Manpage of TCPDump. http://www.tcpdump.org/tcpdump_man.html, September 2015. letzter Zugriff April 2016.

[66] Wireshark Foundation. WIRESHARK. https://www.wireshark.org/. letzter Zugriff April 2016.

[67] Van Jacobson, Craig Leres and Steven McCanne, all of the Lawrence Berkeley National Laboratory. Manpage of PCAP. http://www.tcpdump.org/manpages/pcap.3pcap.html, März 2015. letzter Zugriff April 2016.

[68] Viano, Piero and Pai, Navin and Alberelli, Gianpiero and Magliano, Fabio and Lantieri, Antonio and Meo, Giovanni. WinPcap. http://www.winpcap.org/. letzter Zugriff April 2016.

[69] Steven McCanne and Van Jacobson. The bsd packet filter: A new architecture for user-level packet capture. In *Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings*, USENIX'93, pages 2–2, Berkeley, CA, USA, 1993. USENIX Association.

[70] Schulist, Jay and Borkmann, Daniel and Starovoitov, Alexei. Linux Socket Filtering aka Berkeley Packet Filter (BPF). https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/Documentation/networking/filter.txt. letzter Zugriff April 2016.

[71] W. Richard Stevens and Gary R. Wright. *TCP/IP Illustrated, Volume 2, The Implementation*. Addison-Wesley, 17 edition, 2003.

[72] W. Richard Stevens. *TCP/IP Illustrated, Volume 1, The Protokols*. Addison-Wesley, 23 edition, 2003.

[73] W. Richard Stevens, Bill Fenner, and Andrew M. Rudoff. *UNIX Network Programming, Vol. 1*. Pearson Education, 3 edition, 2003.

[74] HP Labs. httperf. http://www.hpl.hp.com/research/linux/httperf/. letzter Zugriff November 2015.

[75] The Apache Software Foundation. Appache benchmark. http://httpd.apache.org/docs/2.2/programs/ab.html. letzter Zugriff November 2015.

[76] Jörg Zinke, Jan Habenschuß, and Bettina Schnor. servload: Generating representative workloads for web server benchmarking. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECT)*, Genoa, 2012.

[77] w3.org. The common logfile format. http://www.w3.org/Daemon/User/Config/Logging.html#common-logfile-format, 1995.

[78] The Apache Software Foundation. Log files. https://httpd.apache.org/docs/2.2/logs.html#accesslog.

[79] Jörg Zinke. salbnet. http://www.salbnet.org/, 2008.

[80] ACME Labs. http_load. http://acme.com/software/http_load/. letzter Zugriff November 2015.

[81] Jörg Jung. servload: Service benchmark for HTTP and DNS. http://www.salbnet.org/. letzter Zugriff November 2015.

[82] Ming Yang Kao. *Encyclopedia of Algorithms*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[83] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, New York, NY, USA, 1998.

[84] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Dynamic Speed Scaling to Manage Energy and Temperature. In *IEEE Syposium on Foundations of Computer Science*, pages 520–529, 2004.

[85] John Augustine, Sandy Irani, and Chaitanya Swamy. Optimal Power-Down Strategies. *SIAM J. Comput.*, 37(5):1499–1516, 2008.

[86] Ho-Leung Chan, Jeff Edmonds, Tak-Wah Lam, Lap-Kei Lee, Alberto Marchetti Spaccamela, and Kirk Pruhs. Nonclairvoyant Speed Scaling for Flow and Energy. In *26th International Symposium on Theoretical Aspects of Computer Science (STACS 2009)*, 2009.

[87] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Online strategies for dynamic power management in systems with multiple power-saving states. *ACM Trans. Embed. Comput. Syst.*, 2:325–346, August 2003.

[88] John Wilkes and Charles Reiss. Google trace 2011. `https://code.google.com/p/googleclusterdata/wiki/ClusterData2011_1`, 2011.

[89] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 7:1–7:13, New York, NY, USA, 2012. ACM.

[90] Amazon Web Services. Amazon EC2. `https://aws.amazon.com/de/ec2/`. letzter Zugriff April 2016.

[91] The Cloud Computing and University of Melbourne Distributed Systems Laboratory. Cloudsim: A framework for modeling and simulation of cloud computing infrastructures and services. `http://www.cloudbus.org/cloudsim/`. letzter Zugriff 23.11.2015.

[92] Martin Arlitt. Worldcup98. `ftp://ita.ee.lbl.gov/html/contrib/WorldCup.html`, 1998.

[93] Gong Chen, Wenbo He, Jie Liu, Suman Nath, Leonidas Rigas, Lin Xiao, and Feng Zhao. Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, pages 337–350, Berkeley, CA, USA, 2008. USENIX Association.

[94] Ecova. Plug Load Solutions (80 PLUS). `http://www.plugloadsolutions.com/`. letzter Zugriff Mai 2016.

[95] Christoph Möbius, Waltenegus Dargie, and Alexander Schill. Power Consumption Estimation Models for Processors, Virtual Machines, and Servers. *IEEE Transactions on Parallel and Distributed Systems*, 25(6):1600–1614, June 2014.

[96] Waltenegus Dargie and Alexander Schill. Analysis of the Power and Hardware Resource Consumption of Servers under Different Load Balancing Policies. In Rong Chang, editor, *IEEE CLOUD*, pages 772–778. IEEE, 2012.

[97] Pat Bohrer, Elmootazbellah N. Elnozahy, Tom Keller, Michael Kistler, Charles Lefurgy, Chandler McDowell, and Ram Rajamony. The Case for Power Management in Web Servers. In Robert Graybill and Rami Melhem, editors, *Power Aware Computing*, pages 261–289. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[98] Karthick Rajamani and Charles Lefurgy. On Evaluating Request-Distribution Schemes for Saving Energy in Server Clusters. In *Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software*, ISPASS '03, pages 111–122, Washington, DC, USA, 2003. IEEE Computer Society.

[99] Charles Lefurgy, Karthick Rajamani, Freeman Rawson, Wes Felter, Michael Kistler, and Tom W. Keller. Energy Management for Commercial Servers. *Computer*, 36(12):39–48, December 2003.

[100] Rajarshi Das, Jeffrey O. Kephart, Charles Lefurgy, Gerald Tesauro, David W. Levine, and Hoi Chan. Autonomic Multi-agent Management of Power and Performance in Data Centers. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Industrial Track*, AAMAS '08, pages 107–114, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[101] Xiaorui Wang, Ming Chen, C. Lefurgy, and T.W. Keller. SHIP: Scalable Hierarchical Power Control for Large-Scale Data Centers. In *Parallel Architectures and Compilation Techniques, 2009. PACT '09. 18th International Conference on*, pages 91–100, Sept 2009.

[102] Xiaorui Wang, Ming Chen, C. Lefurgy, and T.W. Keller. SHIP: A Scalable Hierarchical Power Control Architecture for Large-Scale Data Centers. *Parallel and Distributed Systems, IEEE Transactions on*, 23(1):168–176, Jan 2012.

[103] Charles Lefurgy, Xiaorui Wang, and Malcolm Ware. Power Capping: a Prelude to Power Shifting. *Cluster Computing*, 11(2):183–195, 2007.

[104] Eduardo Pinheiro, Ricardo Bianchini, Enrique V. Carrera, and Taliver Heath. Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLP'01)*, September 2001.

[105] Eduardo Pinheiro, Ricardo Bianchini, Enrique V. Carrera, and Taliver Heath. Dynamic cluster reconfiguration for power and performance. In Luca Benini, Mahmut Kandemir, and J. Ramanujam, editors, *Compilers and operating systems for low power*, pages 75–93. Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[106] Enrique V. Carrera and Ricardo Bianchini. Efficiency vs. Portability in Cluster-based Network Servers. In *Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming*, PPoPP '01, pages 113–122, New York, NY, USA, 2001. ACM.

[107] Enrique V. Carrera and Ricardo Bianchini. Efficiency vs. Portability in Cluster-based Network Servers. *SIGPLAN Not.*, 36(7):113–122, June 2001.

[108] Eduardo Pinheiro and Ricardo Bianchini. The Nomad Project. `http://www.cs.rutgers.edu/~ricardob/nomad.html`. letzter Zugriff April 2016.

[109] Eduardo Pinheiro and Ricardo Bianchini. Nomad: A Scalable Operating System for Clusters of Uni- and Multiprocessors. In *Cluster Computing, 1999. Proceedings. 1st IEEE Computer Society International Workshop on*, pages 247–254, 1999.

[110] T. Heath, E. Pinheiro, J. Hom, U. Kremer, and R. Bianchini. Application transformations for energy and performance-aware device management. In *Parallel Architectures and Compilation Techniques, 2002. Proceedings. 2002 International Conference on*, pages 121–130, 2002.

[111] Ricardo Bianchini. Leveraging Renewable Energy in Data Centers: Present and Future. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '12, pages 135–136, New York, NY, USA, 2012. ACM.

[112] Ricardo Bianchini, Md E. Haque, William Katsak, and Thu D. Nguyen. Parasol: Rutgers' Green μDatacenter. `http://parasol.cs.rutgers.edu/`. letzter Zugriff Mai 2016.

[113] Carlos Santana, Julius C. B. Leite, and Daniel Mossé. Power management by load forecasting in web server clusters. *Cluster Computing*, 14(4), 2011.

[114] Luciano Bertini, Julius C. B. Leite, and Daniel Mossé. Optimal Dynamic Configuration in Web Server Clusters. Technical report, Instituto de Computação – UFF, 2008.

[115] Luciano Bertini, JCB Leite, and Daniel Mossé. SISO PIDF Controller in an Energy-efficient Multi-tier Web Server Cluster for E-commerce. *2nd IEEE Intl. Workshop on Feedback Control Impl. and Design in Computing Sys. and Networks*, 2007.

[116] Luciano Bertini, Julius CB Leite, and Daniel Mosse. Statistical QoS Guarantee and Energy-efficiency in Web Server Clusters. In *Real-Time Systems, 2007. ECRTS'07. 19th Euromicro Conference on*, pages 83–92. IEEE, 2007.

[117] Cosmin Rusu, Alexandre Ferreira, Claudio Scordino, Aaron Watson, Rami Melhem, and Daniel Mossé. Energy-Efficient Real-Time Heterogeneous Server Clusters. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS '06, pages 418–428, Washington, DC, USA, 2006. IEEE Computer Society.

[118] Andreas Merkel and Frank Bellosa. Balancing Power Consumption in Multiprocessor Systems. In *First ACM SIGOPS EuroSys Conference*, Leuven, Belgium, April 18–21 2006.

[119] Nikolas Ioannou, Michael Kauschke, Matthias Gries, and Marcelo Cintra. Phase-Based Application-Driven Hierarchical Power Management on the Single-chip Cloud Computer. In Lawrence Rauchwerger and Vivek Sarkar, editors, *PACT*, pages 131–142. IEEE Computer Society, 2011.

[120] Jason Howard, Saurabh Dighe, Sriram R. Vangal, Gregory Ruhl, Nitin Borkar, Shailendra Jain, Vasantha Erraguntla, Michael Konow, Michael Riepen, Matthias Gries, Guido Droege, Tor Lund-Larsen, Sebastian Steibl, Shekhar Borkar, Vivek K. De, and Rob F. Van der Wijngaart. A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and dvfs for performance and power scaling. *J. Solid-State Circuits*, 46(1):173–183, 2011.

[121] NAS Parallel Benchmarks. `https://www.nas.nasa.gov/Software/NPB/`. letzter Zugriff Mai 2016.

[122] Standard Performance Evaluation Corporation. `https://www.spec.org/mpi2007/`. letzter Zugriff Mai 2016.

[123] Krishna Kandalla, Emilio P. Mancini, Sayantan Sur, and Dhabaleswar K. Panda. Designing Power-Aware Collective Communication Algorithms for InfiniBand Clusters. In *Proceedings of the 2010 39th International Conference on Parallel Processing*, ICPP '10, pages 218–227, Washington, DC, USA, 2010. IEEE Computer Society.

[124] MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE. `http://mvapich.cse.ohio-state.edu/`. letzter Zugriff Mai 2016.

[125] Ervin Toth, Zoltan Hornak, and Gergely Toth. Protection System against Overload and Distributed Denial of Service Attacks. *Dependability of Computer Systems, International Conference on*, 0:195–202, 2008.

[126] Jan Schaffner, Tim Januschowski, Megan Kercher, Tim Kraska, Hasso Plattner, Michael Franklin, and Dean Jacobs. Rtp: Robust tenant placement for elastic in-memory database clusters. In *ACM SIGMOD Conference*, 2013.

[127] Brandon Heller, Srini Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. ElasticTree: saving energy in data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, pages 17–17, Berkeley, CA, USA, 2010. USENIX Association.

[128] Open Networking Foundation. OpenFlow. `https://www.opennetworking.org/sdn-resources/openflow/`. letzter Zugriff Mai 2016.

[129] Priya Mahadevan, Puneet Sharma, Sujata Banerjee, and Parthasarathy Ranganathan. A Power Benchmarking Framework for Network Devices. In *Proceedings of the 8th International IFIP-TC 6 Networking Conference*, NETWORKING '09, pages 795–808, Berlin, Heidelberg, 2009. Springer-Verlag.

[130] Apposite Technologies. Apposite WAN Emulators. `http://www.apposite-tech.com/`. letzter Zugriff Mai 2016.

[131] Waltenegus Dargie. Analysis of the Power Consumption of a Multimedia Server under Different DVFS Policies. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 779–785, June 2012.

[132] Alaa Brihi and Waltenegus Dargie. Dynamic Voltage and Frequency Scaling in Multimedia Servers. In *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, pages 374–380, March 2013.

[133] Kateryna Rybina, Waltenegus Dargie, Anja Strunk, and Alexander Schill. Investigation into the energy cost of live migration of virtual machines. In *Sustainable Internet and ICT for Sustainability (SustainIT), 2013*, pages 1–8, Oct 2013.

[134] Chia-Hung Lien, Ying-Wen Bai, Ming-Bo Lin, and Po-An Chen. The saving of energy in Web server clusters by utilizing dynamic sever management. In *Networks, 2004. (ICON 2004). Proceedings. 12th IEEE International Conference on*, volume 1, pages 253–257 vol.1, Nov 2004.

[135] In Hwan Doh, Young Jin Kim, Jung Soo Park, Eunsam Kim, Jongmoo Choi, Donghee Lee, and Sam H. Noh. Towards Greener Data Centers with Storage Class Memory: Minimizing Idle Power Waste Through Coarse-grain Management in Fine-grain Scale. In *Proceedings of the 7th ACM International Conference on Computing Frontiers*, CF '10, pages 309–318, New York, NY, USA, 2010. ACM.

[136] Andrew Krioukov, Prashanth Mohan, Sara Alspaugh, Laura Keys, David Culler, and Randy H. Katz. NapSAC: Design and Implementation of a Power-proportional Web Cluster. In *Proceedings of the First ACM SIGCOMM Workshop on Green Networking*, Green Networking '10, pages 15–22, New York, NY, USA, 2010. ACM.

[137] BeagleBoard.org Foundation. BeagleBoard. `http://beagleboard.org`. letzter Zugriff Juni 2016.

[138] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power Provisioning for a Warehouse-sized Computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 13–23, New York, NY, USA, 2007. ACM.

[139] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. EnerJ: Approximate Data Types for Safe and General Low-Power Computation. In *Programming Language Design and Implementation (PLDI)*, June 2011.

[140] Junichi Hikita, Akio Hirano, and Hiroshi Nakashima. Saving 200kW and 200 K/year by power-aware job/machine scheduling. In *IPDPS*, pages 1–8. IEEE, 2008.

[141] Rainer Jung. Neues im apache webserver 2.4. `http://www.heise.de/open/artikel/Neues-im-Apache-Webserver-2-4-1437812.html`, 2012.

[142] Haakon Bryhni, Espen Klovning, and Oivind Kure. A comparison of load balancing techniques for scalable Web servers. *IEEE Network*, 14:58–64, 2000.

[143] Yong Meng Teo and Rassul Ayani. Comparison of load balancing strategies on cluster-based web servers. *Simulation*, pages 185–195, 2001.

[144] iconmobile group 2015. Iconmobile. https://www.iconmobile.com/. letzter Zugriff Oktober 2015.

[145] Guido Urdaneta, Guillaume Pierre, and Maarten van Steen. Wikipedia workload analysis for decentralized hosting. *Elsevier Computer Networks*, 53(11):1830–1845, July 2009. http://www.globule.org/publi/WWADH_comnet2009.html.

[146] Simon Kiertscher and Bettina Schnor. Energy Aware Resource Management for Clusters of Web Servers. In *IEEE International Conference on Green Computing and Communications*, pages 148–156, Beijing, China, 2013.

[147] Stefan Pioch. Analyse von performance- und stromverbrauchbeeinflussenden Technologien im High Performance Computing. Bachelorarbeit, Insitut für Informatik, Universität Potsdam, 2014. http://www.cs.uni-potsdam.de/bs/teaching/docs/thesis/2014/pioch.pdf.

[148] Dieter Urban and Jochen Mayerl. *Regressionsanalyse: Theorie, Technik und Anwendung*. Studienskripten zur Soziologie. VS Verlag für Sozialwissenschaften, 2006.

[149] Norman Richard Draper and Harry Smith. *Applied Regression Analysis*. Wiley Series in Probability and Mathematical Statistics. Wiley, 1980.

[150] Langendörfer, Horst and Schnor, Bettina. *Verteilte Systeme*. Anwendungsorientierte Informatik. Hanser, 1994.

[151] Alex Martelli. *Python in a nutshell*, volume 2. O'Reilly, 2006.

[152] Python Software Foundation. Python v2.7.8 documentation. https://docs.python.org/2.7/. letzter Zugriff 03.09.2014.

[153] Python Software Foundation. Python v2.7.10 documentation. https://docs.python.org/2.7/library/subprocess.html#frequently-used-arguments. letzter Zugriff 16.11.2015.

[154] Sebastian Menski. tcpdump Analyse. Studienarbeit, Insitut für Informatik und Compuational Science, Universität Potsdam, 2016. http://www.cs.uni-potsdam.de/bs/teaching/docs/thesis/2016/menski.pdf.

[155] Will Glozer. wrk. https://github.com/wg/wrk. letzter Zugriff Juni 2016.

[156] Amazon Web Services. Hochverfügbare cloud services mit der amazon cloud. https://aws.amazon.com/de/cloud/. letzter Zugriff 18.11.2015.

[157] Microsoft. Microsoft azure. https://azure.microsoft.com/. letzter Zugriff 18.11.2015.

[158] STRATO AG. Strato servercloud. https://www.strato.de/server-cloud/. letzter Zugriff 18.11.2015.

[159] Amazon Web Services. Amazon webservices simple monthly calculator. http://calculator.s3.amazonaws.com/index.html. letzter Zugriff 19.08.2016.

[160] Raj Jain. *The Art of Computer Systems Performance Analysis*, volume 1. Wiley Professional Computing, 1991.

[161] Lars Schneidenbach. *The Benefits of One-Sided Communication Interfaces for Cluster Computing*. Dissertation, University of Potsdam, 2009.

[162] Guilherme Arthur Geronimo, Jorge Werner, Rafael Weingartner, Carlos Becker Westphall, and Carla Merkle Westphall. Provisioning, resource allocation, and dvfs in green clouds. *International Journal on Advances in Networks and Services*, 1&2(7):108–117, 2014.

[163] Nadine Falkmann. *Exploration von Cloudsim*. Studienarbeit, University of Potsdam, 2015.

[164] Zhi Xiong, Shengtao Zeng, and Hongyan Lu. DESDSim: A dynamic energy-saving deployment simulation platform for web server cluster. In *Signal Processing, Communications and Computing (ICSPCC), 2014 IEEE International Conference on*, pages 477–482, Aug 2014.

[165] Ingo Molnar. Completely Fair Scheduler. `https://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt`. letzter Zugriff 23.11.2015.

[166] Daniel P Bovet and Marco Cesati. *Understanding the Linux Kernel; 3rd ed.* O'Reilly, Sebastopol, CA, 2006.

[167] V. Paxson and M. Allman. Computing TCP's Retransmission Timer. RFC 2988 (Proposed Standard), November 2000.

[168] Vern Paxson, Mark Allman, H.K. Jerry Chu, and Matt Sargent. Computing TCP's Retransmission Timer. `http://www.rfc-editor.org/rfc/rfc6298.txt`, 2011.

[169] Standard Performance Evaluation Corporation. SPECpower_ssj2008 Benchmark. `https://www.spec.org/power_ssj2008/`. letzter Zugriff Februar 2016.

[170] I. Grigorik. *High Performance Browser Networking: What every web developer should know about networking and browser performance*. O'Reilly Media, Incorporated, 2013.

[171] Roman Serdar Mendle. Selten Unnachhaltig – Seltene Erden und Umweltverschmutzung in China. `http://www.stimmen-aus-china.de/2011/04/26/selten-unnachhaltig-seltene-erden-und-umweltverschmutzung-in-china/`. letzter Zugriff Juni 2016.

[172] Mineralienatlas. Seltene Erden - Fluch oder Segen? `https://www.mineralienatlas.de/lexikon/index.php/Mineralienportrait/Seltene%20Erden/Seltene%20Erden%20-%20Fluch%20oder%20Segen`. letzter Zugriff Juni 2016.

[173] Jörg Jung. *Efficient credit based server load balancing*. PhD thesis, Universität Potsdam, 2015.