technische universität
dortmund

Diplomarbeit

Anwendungsspezifische Maßschneiderung des Linux-Kerns für virtuelle Maschinen

Robert Neue
31. August 2010

Betreuer:
Prof. Dr.-Ing. Olaf Spinczyk
Dipl.-Inform. Jochen Streicher

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl Informatik 12
Arbeitsgruppe Eingebettete Systemsoftware
http://ess.cs.tu-dortmund.de

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet, sowie Zitate kenntlich gemacht habe.

Dortmund, den 31. August 2010

Robert Neue

Zusammenfassung

Diese Arbeit untersucht quantitativ die Auswirkungen von Optimierungen eines Gast-Betriebssystems innerhalb einer virtuellen Maschine für eine Serveranwendung. Das System wurde mit dem VMWare Server 2.0.1, dem Linux-Kernel 2.6.28.10 und lighttpd als Webserveranwendung realisiert. Der Webserver wurde jeweils über zwanzig Minuten einer zeitlich linear ansteigenden Abfragerate ausgesetzt und dabei die Performance aller wichtigen Systemaufrufe unter den verschiedenen Optimierungsstrategien für den Linux-Kernel bestimmt. Untersucht wurden die Auswirkungen von Konfiguration-Minimierungen, Code-Optimierung durch den GCC, Nutzung von verschiedenen Virtualisierungsstrategien, sowie die Maßschneiderung des Kernels. Außer der Anwendung des GCC, die kontraproduktiv war, wurde — integriert über die verschiedenen Lastszenarien — durch die anderen Maßnahmen zusammen grob eine Verdopplung der Ausführungsgeschwindigkeit erreicht. Der Einsatz von KVM als Virtualisierungslösung brachte eine zusätzliche weitere Beschleunigung von 5% gegenüber dem Einsatz von VMWare Server 2.0.1.

Inhaltsverzeichnis

1 Einleitung

Mit dem Einzug immer komplexerer und für den Programmierer komfortabler Programmiersprachen und umfangreicher Bibliotheken (Libraries) nehmen auf der einen Seite Übersichtlichkeit, Wartbarkeit und Qualität der Software zu, die effektive Nutzung von Daten und Ressourcen wird meist jedoch zugunsten von Generalität vernachlässigt.

Betriebssysteme sollen auf unterschiedlichster Hardware in unterschiedlichsten Skalierungen ausgeführt werden können. So sollen sie auf Server-Cluster ebenso ausgeführt werden, wie auf eingebetteten Systemen im Automotive-Sektor oder dem heimischen Rechner.

Dies führt jedoch zu generalisierten Betriebssystemen, die vor dem von A.M. Lister und R.D. Eager festgestellten Dilemma stehen:

„Clearly, the operating system design must be strongly influenced by the type of use for which the machine is intended. Unfortunately it is often the case with 'general purpose machines' that the type of use cannot be easily identified; a common criticism of many systems is that in attempting to be all things to all men they wind up being totally satisfactory to no-one."[LE88]

Nicht jede Anwendung benötigt ein Mehr-Prozessoren-Betriebssystem, welches fähig ist, Dateizugriffe auf die exotischsten Dateisysteme oder sogar im Netzwerk liegende Speicher erfolgreich auszuführen. Diese Beispiele und das Zitat zeigen deutlich, dass Einsparpotenziale im Betriebssystem vorhanden sind und bei der Maßschneiderung des Betriebssystems auf eine Anwendung hin, zu Performance-Gewinnen führen werden.

Mit Beginn der kostenlosen, kommerziell entwickelten Lösung von VMWare in den Jahren 2005/06 begannen viele Firmen, bestehende Server und deren Betriebssysteme mit Hilfe von Virtualisierung zu konsolidieren. Die ursprünglichen Server führten meist nur einen Dienst oder wenige Dienste, wie etwa einen Mail-Server oder einen Webserver, gleichzeitig aus. Ein häufiger Fall im Bereich der Virtualisierung stellen virtuelle Maschinen, die nur einen Dienst ausführen, dar.

Abschnitt 1.1 legt die der Arbeit zugrundeliegende Motivation dar. Eine Arbeit, die eine Maßschneiderung für wissenschaftliche Anwendungen vornimmt, wird in Abschnitt 1.2 erläutert. Die sich durch die Motivation ergebenden Fragestellungen werden in Abschnitt 1.3 als Ziele der Diplomarbeit defniniert. Abschließend gibt Abschnitt 1.4 einen Überblick über die gesamte Arbeit.

1.1 Motivation

Virtuelle Maschinen werden seit einigen Jahren zunehmend genutzt.

Sie finden sich in vielen Bereichen, wie etwa auf Servern, die im Rechenzentrum stehen, bei Testplätzen von Entwicklern — die virtuelle Maschinen benutzen, um zum Beispiel ihre Software auf die Kompaktibiltät zu verschiedenen Betriebssystemen zu testen —, bei Systemadministratoren, die eine Migration von Rechnern planen oder sogar bei ambitionierten Privatpersonen, die ein neues Betriebssystem ausprobieren wollen. Viele dieser Bereiche wurden durch die Virtualisierung geradezu revolutioniert.

Im Bereich der System-Administration ist es auf diese Weise nicht mehr von Nöten, für Testumgebungen, freie Rechner zum Experimentieren zu finden, diese eventuell mit passend gleicher Hardware auszurüsten und am besten in einem isolierten Netzwerk zu verkabeln. Die Virtualisierung ermöglicht zum Beispiel den Test einer Migration von einer Domäne zu einer anderen, mit mehreren beteiligten Servern und Workstations, auf nur einem leistungsstarken Rechner nachzubilden.

Durch den Einsatz von Virtualisierung in vielen Testumgebungen, ist die Anzahl der virtualisierten Rechner stark gestiegen. Die Skepsis gegenüber der Virtualisierung nimmt seit einigen Jahren immer weiter ab. Viele Unternehmen virtualisieren inzwischen immer mehr Produktiv-Server, auch in unternehmenskritischen Bereichen, wie etwa einem SAP-Datenbankserver. Zwischen 24 % und 33 % der SAP-Anwender verwenden bereits virtualisierte Serveranwendungen (Siehe: Abbildung 1.1). Da besonders größere Firmen mit einer größeren Anzahl von Servern hiervon Gebrauch machen, ist sogar der Prozentsatz bezogen auf die Anzahl der Server noch einmal höher (Siehe: Abbildung 1.2).

Vor der Einführung der kostenlosen Virtualisierungslösung von VMWare, wurden viele Produktiv-Server, isoliert als einzelne Rechner mit wenigen Diensten, idealerweise nur mit einem, installiert. Viele Produktiv-Server können durch entsprechende Virtualisierungs-Werkzeuge direkt in eine virtuelle Maschine verwandelt werden und verringern so zum Beispiel das Hardware-Ausfallrisiko[1], welches bei Produktiv-Servern, die lange Zeit im Dienst sind und stets auf derselben (alten) Hardware ausgeführt werden, ständig steigt. Zusätzlich kann die Anzahl der Server auf diese Weise konsolidiert werden, da mehrere Server auf einer virtuellen Maschine laufen können.

Gerne wird bei Virtualisierungslösungen mit den Synergie-Effekten beim Zusammenlegen von Servern geworben. So können Unternehmen durch die Konsolidierung Stromkosten sparen, vorhandene Hardware-Ressourcen besser auslasten

[1] Das Totalausfallrisiko hingegen steigt, da durch einen Hardwareausfall des Host viele virtuelle Maschinen betroffen sind. Diese können jedoch leichter als sonst auf andere Rechner umgesetzt werden. Um dem Totalausfallrisiko zu begegnen, müssen andere Maßnahmen, wie Aufbau von Redundanzen in Betracht gezogen werden.

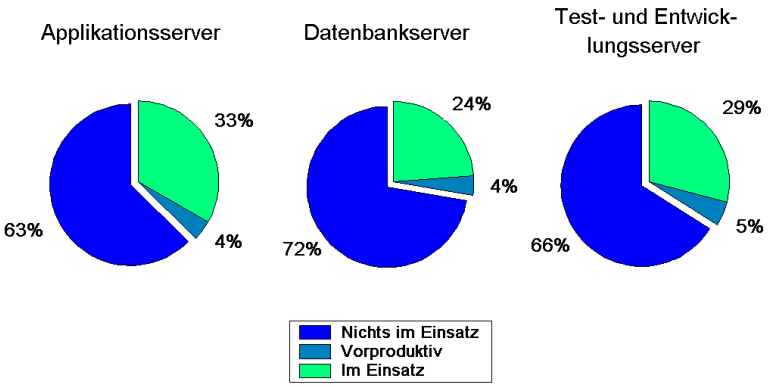

Abbildung 1.1: Prozentualer Anteil der SAP-Anwender mit Virtualisierung auf unternehmenskritischen Servern (Quelle: RAAD)

Abbildung 1.2: Prozentualer Anteil an Servern mit Virtualisierung bei SAP-Anwendern (Quelle: RAAD)

und vieles mehr. Doch durch die Virtualisierung entsteht auch wieder Overhead an vielen Stellen der Virtualisierung. So müssen kritische Befehle vor dem Ausführen auf der CPU abgefangen, Aufrufe an die emulierte Hardware[2] umgeleitet und der Speicher für die virtuellen Maschinen entsprechend verwaltet werden. Alleine der Overhead durch die Verwaltung des zugesicherten RAMs beträgt nach Ahnert 5%-10% bei einer Virtualisierung durch VMWare [Ahn09].

Die Betriebssysteme in den virtuellen Maschinen sind im Wesentlichen sogenannte Vielzweck-Betriebssysteme (*General Purpose Operating Systems*). Die bekannten Vertreter dieser Betriebssysteme, wie beispielsweise Linux oder Windows, haben ihre Popularität daher, dass sie für fast sämtliche Hardware Treiber bereitstellen, und so generisch geschrieben sind, dass sie mit jeglicher Hardware kommunizieren können.

Betriebssysteme müssen jedoch nicht nur mit der Hardware kommunizieren können, sondern auch Systemaufrufe den verschiedensten Anwendungen anbieten. Diese Vielzweck-Betriebssysteme vertrauen jedoch nicht auf die Umgebung in der sie laufen, sondern wenden verschiedene Konzepte an, um sich beispielsweise gegen die folgenden Problemfelder abzusichern:

- Fehlerhafte oder bösartige Anwendungen: Präemptives Scheduling verhindert, dass eine Anwendung die CPU dauerhaft nutzen kann, ohne diese wieder freigeben zu müssen; Adressraumtrennung schützt normale Anwendungen vor dem Zugriff durch bösartige Anwendungen.
- Mehrbenutzerbetrieb: Dieser wird durch die Authentifizierung, sowie den Zugriffschutz auf Dateien und Geräte sichergestellt.
- Vielfädige Anwendungen, miteinander interagierende Prozesse: Kritische Bereiche, wie gemeinsam genutzte Datenstrukturen, werden zum Beispiel mit Semaphoren vor unerlaubten Änderungen durch andere Prozesse geschützt.
- Große oder viele Programme: Verschiedene Konzepte wie virtueller Speicher oder Swapping ermöglichen die Nutzung von Ressourcen wie RAM auch noch bei vielen, speicherhungrigen Programmen.[Spi08a]

Als Betriebssystem für alle Eventualitäten gewappnet zu sein, heißt für die Anwendungen, dass diese auch für nicht benutzte Funktionalitäten des Betriebssystems, durch eine erhöhte Ausführungszeit des System-Aufrufs, bezahlen. Auf den ersten Blick mag dies noch fair erscheinen, da alle Anwendungen gleich viel bezahlen müssen. Benötigt eine Anwendung aber nur einen kleinen Teil der Funktionalität, so ist der Overhead für die Anwendung recht groß und es ergeben sich Maßschneiderungspotenziale.

[2] Bei der Virtualisierung werden nur Teilbereiche der Hardware emuliert. CPU oder Speicher, sind perfomance-kritisch und werden nicht emuliert. Eventuell ist jedoch eine Ausnahmebehandlung vor der Nutzung der Hardware erforderlich.

Um herauszufinden, welche Funktionalitäten die Anwendung vom Betriebssystem wirklich benötigt, können verschiedene Profiler eingesetzt werden. Bei der Auswahl des Profilers ist darauf zu achten, dass der Overhead durch die Messung so gering wie möglich bleibt [KMPP07].

Mit den Ergebnissen aus dem Profiling kann anschließend das Betriebssystem entsprechend untersucht und maßgeschneidert werden.

Um herauszufinden, ob die Anwendung vollständig von der Maßschneiderung profitiert und die Maßschneiderung zu einer Steigerung der Performance führt, oder andere Barrieren die Ergebnisse der Maßschneiderung widerum beeinflussen, können anwendungsbezogene Benchmarks benutzt werden.

Das Anliegen dieser Diplomarbeit ist es, die Möglichkeiten zur Einsparung von CPU-Zyklen in einer bekannten Umgebung, der virtuellen Hardware eines VMWare Server 2.0 und eines Debian Kernels, auszuloten.

Vielzweck-Betriebssysteme definieren oftmals einen abstrakten Normalfall für alle Anwendungen, den jedoch keine Anwendung vollständig ausnutzt. Eine maßgeschneiderte Systemsoftware erhöht die Effizienz nicht nur bei der Ein- und Ausgabe oder der Rechenzeit, sondern ebenso durch weniger Platzbedarf im Speicher und eine bessere Energieeffizienz. Die Steigerung der Energieeffizienz und die Verringerung des Speicherbedarfs, können zum Teil recht einfach durch das Entfernen von grafischen Oberflächen und das Deinstallieren nicht benötigter Dienste, die einen entsprechend großen Overhead erzeugen, erreicht werden. Ebenso erscheint das Ausloten des Einsparpotenzials durch die Anpassung des Vielzweck-Betriebssystems an die Umgebung der virtuellen Maschine einige interessante Fragestellungen zu bieten.

Wie stark die Anwendung von einer Steigerung, der von ihr genutzten System-Aufrufe, profitiert und ob Einsparungen im Betriebssystem auch in gleichem Maße die Performance der Anwendung erhöhen, ist eine weitere interessante Fragestellung.

Einige Artikel im Internet werfen die interessante Frage auf, wie gut inzwischen andere Virtualisierungssoftware, wie beispielsweise KVM, im Vergleich zu VMWare geworden ist. KVM wird mit immer größer werdendem Interesse auch im Unternehmensbereich eingesetzt. So hat RedHat im April 2010 Xen als Virtuelisierungslösung aus der aktuellen Version von RedHat Enterprise Linux (RHEL) zugunsten von KVM entfernt[3]. Bereits 2008 verkündete RedHat mit Hilfe von KVM als Virtualisierungslösung für je drei virtuelle Maschinen die unter VMWare ESX Server betrieben wurden, fünf virtuelle Maschinen unter KVM auf derselben Hardware laufen lassen zu können. So konnte KVM 52 virtuelle Maschinen,

[3]Siehe: http://www.computerworld.com/s/article/9175883/Red_Hat_drops_Xen_from_RHEL

im Gegensatz zum VMWare Server mit 35 virtuellen Maschinen auf demselben Rechner betreiben. [4]

1.2 Verwandte Arbeiten

Während es viele Arbeiten zu Maßschneiderungs-Techniken gibt, so gibt es wenige Arbeiten, die eine Maßschneiderung des Linux-Kernels auf eine Anwendung hin untersucht haben.

Eine dieser Arbeiten ist von M.Youseff et al. [MWK05]. Sie untersucht, welche Maßschneiderungspotenziale, beim Linux-Kernel im Gebiet der wissenschaftlichen Anwendungen, auftreten. M.Youseff et al. stellen fest, dass durch die Nutzung von Linux sowohl für lokal installierte PCs als auch für batch-kontrollierte High-End-Cluster bei den wissenschaftlichen Anwendern zu andauernden Bemühungen um Optimierungen geführt hat.

Linux versucht, allen möglichen Anwendungsanforderungen gerecht zu werden und besitzt daher viele Merkmale, die von wissenschaftlichen Hoch-Performance-Anwendungen (*high-end scientific applications*) nicht benötigt werden. Diese Anwendungen werden in performanten Clustern ausgeführt und sind größtenteils lange rechnende, ressourcenintensive Anwendungen, die exklusiven Zugriff auf die ihnen zugeteilte Maschine, durch ein Batch-System erlangen. Deshalb gibt es keine Prozesse, die um die Ressourcen wie I/O und CPU konkurrieren müssen. Dennoch widmet Linux einen großen Teil der Funktionalität den Bereichen Sicherheit und Fairness bei nebenläufigen Programmen. Diese Merkmale stehen im Gegensatz zu dem für wissenschaftliche Anwendungen benötigten „Batch“-artigen Anforderungen. Obwohl die Einsparpotenziale sich hierdurch bereits schon abzeichnen, hat sich Linux als de facto Standard bei den Betriebssystemen für Cluster-Architekturen durchgesetzt.

Trotz des Aufwands, der durch Debugging und Optimierung der Programme auf das Cluster betrieben wurde, zeigten diese Ansätze jedoch nur wenig Wirkung. Der Gedanke, Linux auf die wissenschaftlichen Anwendungen direkt Maß zu schneidern und überflüssige Funktionalität zu entfernen, lag nahe. M.Youseff et al. stellen bei ihrer Untersuchung fest, dass das I/O Subsystem für ihre Anwendung „Performance-kritisch“ ist. Während andere Arbeiten mehr das Verhalten von Lese-Anforderungen untersuchen, konzentrieren M.Youseff et al. sich auf die Performance des `write()`-Systemaufrufs, der ihre wissenschaftlichen Anwendungen stark beeinflusst.

M.Youseff et al. stellen fest, dass in ihrer Anwendungsdomäne zwei Strategien angewendet werden können. Zum einen können überflüssige Überprüfungen

[4] Siehe: http://www.channelregister.co.uk/2008/09/10/run_more_vms_with_red_hat_than_vmware/

entfernt werden, zum anderen bietet es sich für die Anwendung an, nicht sofort alles per `write()` zu schreiben, sondern einen Schreib-Puffer einzusetzen. Getestet wurden ein 2.4er-Kernel, ein 2.5er-Kernel, sowie ein 2.6er Kernel. Die Ergebnisse fielen sehr unterschiedlich aus. Während der 2.4er und 2.5er Kernel von der Verschlankung des Kernels am meisten profitierten und die Optimierung des `write()`-Systemaufrufs kaum Zeitersparnis erbrachte, so verhielt es sich bei dem 2.6er Kernel mit den Ergebnissen für Verschlankung und Optimierung genau umgekehrt. 2005 erreichten M.Youseff et al. durch die angewandten Strategien einen Performance-Gewinn von 24% gegenüber einem nicht modifizierten System.

Im Gegensatz zu der Arbeit von M.Youseff et al. beleuchtet diese Diplom-Arbeit die Auswirkungen der Maßschneiderung eines Betriebssystems in einem anderen, stets wachsenden, Anforderungsgebiet von Betriebssystemen: den virtuelle Maschinen. Hier sind typische Anwendungen Serverdienste, die von bestehenden Produktivrechner virtualisiert wurden, fertig vorkonfigurierte Virtual Appliances[5] oder Testrechner, die zu speziellen Zwecken eingerichtet wurden. Ebenso wie bei M.Youseff et al., gibt es auch in virtuellen Maschinen häufig nur eine Anwendung, so dass das Betriebssystem gut maßgeschneidert werden kann.

1.3 Ziele

Die Anforderungen an Betriebssysteme sind vielfältig. Betriebssysteme können je nach Anforderung verschiedenen Bereichen zugeordnet werden. Achilles [Ach06] nennt:

- Batchbetrieb,
- Time-Sharing,
- Transaktionsorientiertheit,
- Echtzeit-Verhalten,

als Anforderungsrichtungen für Betriebssysteme. Wesentliche andere Bereiche sind jedoch noch der stets wachsende Bereich der eingebetteten System, die zunehmend mehr auch im Automotive-Bereich zum Einsatz kommen, sowie sichere Systeme, etwa im Bankenumfeld, oder das High-Performance-Computing, bei dem minimale Kommunikationslatenzen gefordert sind[Spi08a].

So vielfältig die Anforderungsbereiche sind, so unterschiedlich sind auch die Ansätze zu Optimierungen im Bereich der Systemsoftware.

Zwei Trends, die sich in den letzten Jahren etabliert haben, sind die Virtualisierung und die Maßschneiderung.

[5] Ein vorkonfiguriertes Image einer virtuellen Maschine. Es beinhaltet das Gast-Betriebssystem, sowie eine Anwendung

Während der Trend, immer mehr Server zu konsolidieren und als virtuelle Maschinen zu betreiben, nicht mehr so rasant wie noch vor wenigen Jahren betrieben wird, so liegt der Fokus auf der Optimierung der Leistung virtueller Maschinen. Die Marketing-Versprechen, mit Virtualisierung Hardware-Ressourcen effizienter nutzen zu können, sind nicht notwendigerweise richtig, denn durch die Virtualisierung entsteht neuer Overhead.

Die Maßschneiderung von Betriebssystemen gewinnt auf dem Hintergrund der enormen Fülle an virtualisierten Rechnern ebenso an Bedeutung.

Diese Diplomarbeit untersucht, inwieweit eine Performance-Verbesserung durch Maßschneiderung des Betriebssystemkerns auf die Bedürfnisse einer speziellen Anwendung hin, erreicht werden kann. Die Ziele lassen sich wie folgt zusammenfassen: Die Einsparungspotenziale der Maßschneiderung sollen

- aufgezeigt,
- exemplarisch umgesetzt und
- evaluiert werden.
- Anschließend werden die Maßschneiderungsmöglichkeiten auf ihren praktischen Nutzen für die Anwendung untersucht.

Aus Motivation und Zielsetzung ergeben sich folgende Fragestellungen:

- Welche Geschwindigkeitsvorteile können durch die Anpassung des Betriebssystems an die Anwendung als „Common Case“ realisiert werden?
- Welche Möglichkeiten der Maßschneiderung können genutzt werden?
- Wie viel kann durch eine Minimierung der Konfiguration auf das Wesentliche und Nötige eingespart werden?
- Wie effektiv unterstützen die einzelnen Maßnahmen eine Steigerung der Performance für die Anwendung?
- Profitiert die Anwendung von der Maßschneiderung oder gibt es Grenzen, die eine Nutzung der Einsparungen behindern?
- Ist bereits die Wahl der Virtualisierungssoftware entscheidend für die Performance einer Anwendung?

1.4 Aufbau der Arbeit

Die Begriffe Virtualisierung und Maßschneiderung sind in dieser Arbeit zentral und werden im Kapitel 2 Grundlagen erläutert. Zusätzlich werden die in der Arbeit benutzten Werkzeuge eingeführt.

Bei der Server-Virtualisierung stellt VMWare mit einem Marktanteil von ca. 90%[Wie10] „die“ Virtualisierungslösung dar. Als zu untersuchende Virtualisierungslösung wurde der kostenlose „VMWare Server für Linux“ vom Marktführer der kommerziellen Virtualisierungslösungen, VMWare, ausgewählt. Debian wurde wegen seiner stabilen und gut getesteten Versionen sowie dem sehr gut strukturierten und fein granularen Paketmanagement als Test-Betriebssystem gewählt. Unter den konsolidierten Servern, die in vielen virtuellen Maschinen ausgeführt werden, finden sich eine große Anzahl von Webservern. Als zu analysierende Anwendung wurde daher der leichtgewichtige Webserver lighttpd ausgesucht. Die Ergebnisse der Analyse des lighttpd Servers — der unter Debian in einer von VMWare ausgeführten virtuellen Maschine installiert ist — sind im Kapitel 3 dargestellt.

Die in der Analyse gefundenen Optimierungsmöglichkeiten werden im Kapitel 4 dokumentiert.

KVM wurde im Oktober 2006 veröffentlicht. Bereits in Kernel-Version 2.6.20, schaffte es KVM in den Mainstream-Linux-Kernel. Seit September 2008 ist Qumranet eine Tochtergesellschaft von RedHat. Trotz dieser relativ jungen Entwicklungsgeschichte ist KVM sehr leistungsstark und bildet eine interessante Alternative zu VMWare.

Hierdurch motiviert behandelt die Evaluation im Kapitel 5 nicht nur die gefundenen und implementierten Optimierungsmöglichkeiten, sondern auch einen Vergleich der Performance-Unterschiede von VMWare und KVM. Kapitel 6 gibt eine Zusammenfassung und bietet einen Ausblick auf mögliche Erweiterungen.

2 Grundlagen

Das auf die Anwendung anzupassende Betriebssystem läuft nicht direkt auf der Hardware, sondern wird virtualisiert. Abschnitt 2.1 klärt in diesem Zusammenhang die Begrifflichkeiten und gibt eine Einführung in virtuelle Maschinen. Hierin werden die verschiedenen Virtualisierungsarchitekturen erläutert und Virtualisierungslösungen vorgestellt. Der Abschnitt 2.2 stellt verschiedene Kriterien, wie Granularität der Maßschneiderung, eine Klassifizierung der Optimierungsarten sowie Optimierungsziele zur Einteilung der entsprechenden Maßschneiderungs-Techniken vor. Die während der Maßschneiderung benutzten Werkzeuge werden abschließend im Abschnitt 2.3 erläutert.

2.1 Virtuelle Maschinen

Einen generellen Überblick über den Aufbau einer virtuellen Maschine liefert Abbildung 2.1. Je nach Virtualisierungsarchitektur verändern sich einige Schichten.

Definition 2.1 (Virtuelle Maschine)
„A virtual machine is taken to be an efficient, isolated duplicate of the real machine.“[PG74]

Obwohl bereits Popek und Goldberg 1974 virtuelle Maschinen wie in Definition 2.1 angegeben und beschrieben haben, empfiehlt es sich, für den Einstieg eine Praxis orientiertere Definition zu wählen.

Definition 2.2 (Virtuelle Maschine)
„Eine VM [Virtuelle Maschine] ist ein nachgebildeter Rechner, der in einer abgeschotteten Umgebung auf einer realen Maschine läuft. Jede VM verhält sich dabei wie ein vollwertiger Computer mit eigenen Komponenten, wie CPU, RAM, VGA-Adapter, Netzwerkkarten und Festplatten. Auf einige physikalisch vorhandene Bauteile, etwa die CPU oder den RAM darf die VM kontrolliert zugreifen. Andere Geräte wie z.B. Netzwerkkarten können sogar komplett emuliert werden, ohne dass echte Hardware existiert.“ [Ahn09]

In der umfangreichen Definition von Ahnert fehlt jedoch der explizite Zusatz von Popek und Goldberg, das eine virtuelle Maschine effizient sein soll. Dieser Zusatz ermöglicht, im späteren Abschnitt 2.1.1, unter anderem die Unterteilung in Virtualisierung und Emulation.

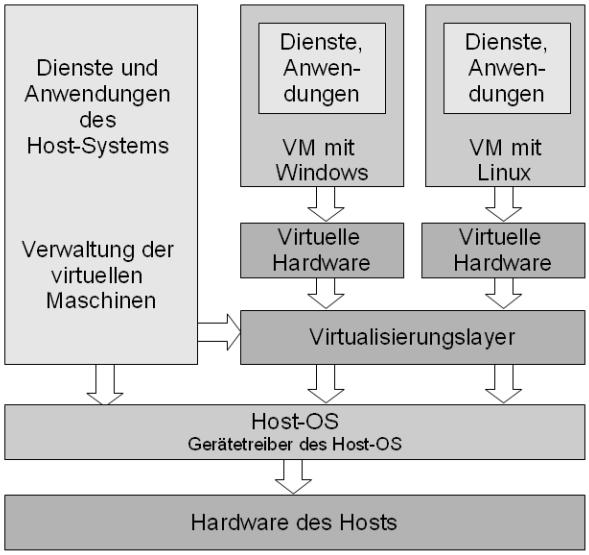

Abbildung 2.1: Virtualisierung - Überblick

Die virtuelle Maschine selbst wird Gast, der reale Rechner, der die virtuelle Maschine ausführt, wird Host oder Wirt genannt. Die virtuelle Maschine kommuniziert mit der vom Virtualisierungslayer bereitgestellten Hardware.

Definition 2.3 (Virtualisierungslayer)
„Die Software, die sich zwischen das Gastbetriebssystem und die reale Hardware schiebt, wird als Virtualisierungslayer, bzw. als Hypervisor oder Virtual Machine Monitor (VMM) bezeichnet. Dieser Virtualisierungslayer emuliert für die Gäste die virtuelle Hardware, teilt ihnen CPU-Zeit zu oder ermöglicht den kontrollierten Zugriff auf bestimmte physische Geräte des Wirts.“ [Ahn09]

Je nachdem, welche Stellung der Virtualisierungslayer in der Hierarchie einnimmt, unterscheidet man zwei Typen:

- Typ 1 Hypervisor, wird privilegiert ausgeführt und verwaltet die Hardware exklusiv.
- Typ 2 Hypervisor, läuft unter einem Host OS, mit dem es sich die Ressourcen teilen muss.

Bei einem Typ 1 Hypervisor (siehe Abbildung 2.2) wird beim Booten sofort die Virtualisierungschicht (der Hypervisor) geladen, und das mit dem Hypervisor

assoziierte Betriebssystem als virtuelle Maschine auf dem Hypervisor gestartet und ausgeführt. Auf diese Art und Weise sind alle Betriebssysteme gleich gut isoliert voneinander und konkurrieren um die gleichen Ressourcen, die jetzt nicht mehr ein Vielzweck-Betriebssystem, sondern der direkt auf der Hardware liegenden Hypervisor[1], verwaltet.

Ein Typ 2 Hypervisor (siehe Abbildung 2.2) ist lediglich eine virtuelle Abstraktionsschicht, die zwischen dem gestarteten Betriebssystem (Host) und dem Gast-Betriebssystem sitzt. Ein Beispiel für diesen Hypervisortyp ist der VMWare Server.

Der VMWare ESX Server hingegen ist ein Beispiel für eine Virtualisierungslösung mit einem Typ 1 Hypervisor. Durch einen direkt auf der Hardware sitzenden Virtualisierungslayer profitieren die virtuellen Maschinen von einer wesentlich höheren Performance. I/O-Instruktionen müssen nicht erst durch das Host-Betriebssystem auf die Hardware zugreifen. Desweiteren sind beim VMWare ESX Server durch die eigenen, auf der Hardware sitzenden Treiber einige Funktionen realisiert, die Administratoren einen deutlichen Mehrwert versprechen:

- *Netzwerkkarten-Teaming* (Bündelung von Adaptern zur Ausfallsicherheit oder Performancesteigerung),
- *Unterstützung von VLans* (virtuelle seperate Netzwerke auf dem gleichen physischen Netzwerk),
- *Multipathing* (mehrere redundante Wege zum externen Speicher zwecks Ausfallsicherheit).

Ein großer Nachteil ist, dass VMWare nicht für alle Hersteller und Komponenten Treiber liefert, sondern nur für zertifizierte. [Ahn09]

Virtualisierungslösungen vom Typ 1 besitzen selber einen kleinen Kernel, der direkt auf der Hardware läuft und alle virtuellen Maschinen bedient. Die direkte Hardwarekontrolle führt zu weiteren Performancevorteilen der VMWare-Treiber.

2.1.1 Virtualisierung und Emulation

Virtualisierung und Emulation sind zwei verschiedene Konzepte, die voneinander abgegrenzt werden müssen.

Definition 2.4 (Emulation)
Als Emulation wird in der Computertechnik das funktionelle Nachbilden eines

[1] Diese Art von Hypervisor besitzt ein eigenes Betriebssystem, welches auf der nackten Hardware (*Bare Metal*) läuft.

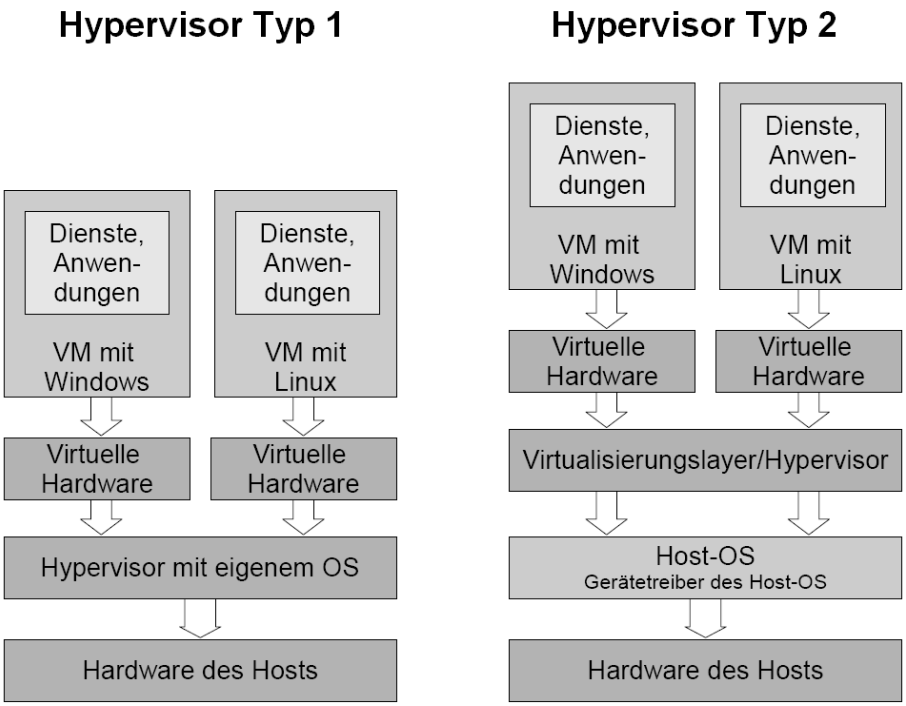

Abbildung 2.2: Überblick über die Hypervisor-Typen

Systems durch ein anderes bezeichnet. Das nachzubildende System erhält die gleichen Daten, führt die gleichen Programme aus und erzielt die gleichen Ergebnisse wie das originale System.[WR10]

Man unterscheidet zwei Arten der Emulation: Hardware-Emulation und API-Emulation.

2.1.1.1 Hardware-Emulation

Mit der Hardware-Emulation wird die komplette Hardware eines Rechners, inklusive seiner CPU und deren Befehlssatz, nachgebildet. Auf diese Weise können Betriebssysteme und Programme installiert und ausgeführt werden, die für eine andere CPU (-Architektur), zum Beispiel dem C64, entworfen wurden. Die Emulation hat zwei große Nachteile:

- geringe Performance (Jede Instruktion, die vom Gast an die CPU übergeben werden soll, muss vom Emulator interpretiert werden.)
- Fehlen einer dynamischen Ressourcen-Verwaltung (Während der Laufzeit können dem Gast die Ressourcen nicht variabel zugeteilt werden.)

Ein Beispiel für einen Emulator ist der Qemu ohne Beschleuniger.[WR10]

2.1.1.2 API-Emulation

Ziel der API-Emulation ist es, Programme anderer Architekturen auf der eigenen laufen zu lassen. Die Schnittstellen (Application Programming Interface - API) und Bibliotheken des anderen Betriebssystems werden nachgebildet. Die Programme laufen nicht notwendigerweise isoliert im eigenen Host-System. Ein einfaches Beispiel ist Cygwin, der die Linux-API unter Windows zur Verfügung stellt. Diese Emulation stellt jedoch keine Virtualisierung dar, sondern ermöglicht lediglich das Ausführen von Programmen, die für andere Bibliotheken entworfen wurden.[WR10]

Der Unterschied zwischen Emulation und Virtualisierung besteht in der Art der virtuellen Hardware. Die Emulation bildet einen kompletten Rechner, inklusive der CPU und ihres Befehlssatzes in Software nach.

Ein Beispiel hierfür ist ein C64-Emulator. In dem Host, der den Emulator ausführt, ist nichts von der C64-Hardware vorhanden, daher muss die komplette Hardware in Software realisiert werden. *„Durch die vollständige Emulation der kompletten Hardware sind solche Lösungen aber sehr langsam, weil für jeden Befehl der nachgebildeten CPU mehrere Befehle des Emulationsprogramms auf der realen CPU ablaufen müssen.“* [Ahn09]

Anders bei der Virtualisierung: Hier wird die physisch in einem Rechner vorhandene Hardware virtualisiert. Dadurch ist beispielsweise die an den Gast durchgereichte CPU dieselbe wie im Host und die Notwendigkeit, den Befehlssatz zu übersetzen, ist nicht gegeben.

Jeden Befehl einzeln zu übersetzen, ist zu zeitaufwendig, daher verfolgt die Virtualisierung einen anderen Ansatz und versucht dem Gast so weit wie möglich, einen kontrollierten Zugriff zur Hardware zu ermöglichen. Ein Beispiel dafür sind die Instruktionen, die direkt auf der CPU des Hosts laufen. [Durch die Emulation eines anderen Befehlssatzes sind sehr rechenaufwendige Transformationen von Instruktionen des einen Befehlssatzes (z.B. C64) in Instruktionen für den zweiten Befehlssatz (z.B. x86) nötig.] Bei der Virtualisierung sind die Befehlssätze für die Instruktionen zwar gleich, dennoch gehen sowohl der Gast als auch der Host davon aus, die volle Gewalt über die Hardware zu besitzen. Dies führt bei einigen Instruktionen, die der Gast ausführen möchte, dazu, dass Konflikte auftreten können, die vor der Durchreichung und Ausführung der Instruktionen auf der zugrunde liegenden CPU abgefangen und geändert werden müssen. Man unterscheidet sensitive und nicht sensitive Befehle.

Definition 2.5 (sensitive Befehle)
Sensitive Befehle können potenziell den Zustand der Maschine verändern. Die virtuelle Maschine darf solche sensitiven Befehle daher nicht direkt auf der CPU ausführen.

Definition 2.6 (nicht-sensitive Befehle)
Die nicht sensitiven Befehle verändern den Zustand der Maschine nicht und haben keinen Zugriff auf die Kontrollregister. Nicht sensitive Befehle können direkt an den Prozessor weitergeleitet werden. Die nicht sensitiven Befehle können weiterhin in privilegierte und nicht privilegierte Befehle, unterschieden werden.
(a) Privilegierte Befehle werden im Kernel Mode,
(b) Nicht privilegierte im User Mode ausgeführt.

Nach Popek und Goldberg [PG74] ist eine Rechenmaschine allgemein nur dann virtualisierbar, wenn jeder sensitive Befehl abgefangen und anschliessend emuliert werden kann. Sollte ein privilegierter Befehl trotzdem im User Modus ausgeführt werden, so wird die Ausführung der Instruktion von der Hardware nicht beachtet und stattdessen zur Fehlerbehandlung an das Betriebssystem weitergegeben. [SGG05]

Sensitive Befehle dürfen nur im Kernel Modus den Zustand der Maschine verändern, sind also automatisch privilegierte Befehle.

Wichtig ist, dass bei der Virtualisierung alle sensitiven Befehle, die ausgeführt werden sollen, abgefangen werden. Werden alle sensitiven Befehle abgefangen, hat ein virtualisiertes Betriebssystem keinen Grund an der Annahme, die volle Kontrolle über die Hardware zu besitzen, zu zweifeln. Der Anteil dieser Instruktionen an der Gesamtzahl aller auszuführenden Instruktionen ist jedoch vergleichsweise klein. An dieser Stelle der Virtualisierung tritt dasselbe Problem wie beim Betriebssystembau auf. Wenn alle Gäste uneingeschränkten Zugriff auf Ressourcen wie RAM oder CPU haben, so ist eine Isolierung nicht mehr gegeben. Eine virtuelle Maschine könnte also sowohl den Host als auch die anderen Gäste mit zum Absturz bringen.

Dieses Problem ist analog zu der Problematik die bei modernen Betriebssystemen auftritt. Ein Betriebssystem muss stets die Kontrolle über alle auf ihm laufenden Anwendungen haben, damit schlecht oder absichtlich bösartig programmierte Anwendungen nicht das Betriebssystem mit zum Absturz bringen. Moderne Betriebssysteme nutzen daher den „Protected Mode“, der verschiedene Prioritätsstufen unterstützt.

Definition 2.7 (Protected Mode)
„Der Protected Virtual Address Mode oder kurz Protected Mode wurde beginnend mit dem 80286 implementiert, um (wie der Name schon sagt) die verschiedenen Tasks unter einem Multitasking-Betriebssystem zu schützen. Zu diesem Zweck prüft die Prozessor-Hardware den Zugriff eines Programms auf Daten und Code und vergibt Berechtigungen für einen solchen Zugriff auf insgesamt vier Schutzebenen. Damit sind Daten und Code geschützt und ein kompletter Systemabsturz des PCs ist normalerweise nicht möglich.“ [Mes00]

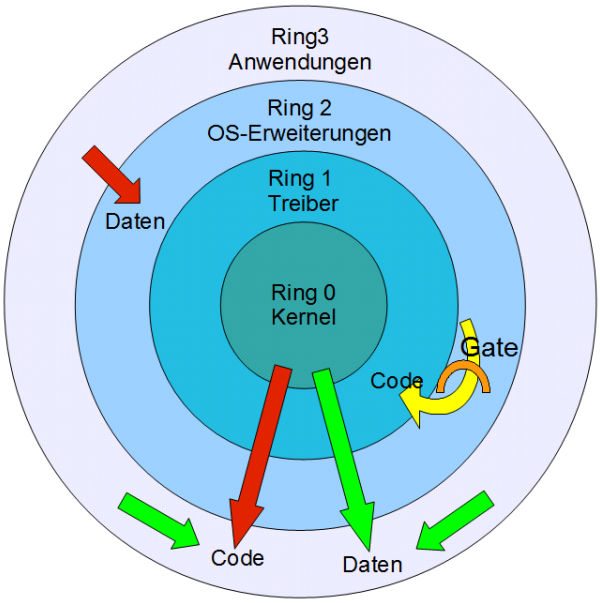

Abbildung 2.3: Protected Mode des i386

Der in Abbildung 2.3 dargestellte Protected Mode erlaubt den Zugriff sowohl auf Code als auch auf Daten derselben Priviliegierungsebene (siehe grüne Pfeile in Ebene 3 - Anwendungen). Für Zugriffe von Ringen mit niedriger Priorität auf Ringe höherer Priorität, zum Beispiel von Ring 3 auf Ring 2, gilt, dass der Zugriff auf Daten untersagt und die Nutzung von Code per Gate (gelber Pfeil) reglementiert ist. Umgekehrt werden Zugriffe von einem Ring mit hoher Priorität auf Daten eines niedrigeren Ringes erlaubt (grüner Pfeil), die Nutzung von Code ist in diesem Fall jedoch untersagt. Obwohl die theoretische Aufteilung des Protected Mode, Treiber und Betriebssystemerweiterungen in den Ringen 1 und 2 unterbringen könnte, nutzen bekannte Betriebssysteme wie Windows oder Linux nur die Ringe 0 und 3.[WR10]

2.1.2 Virtualisierungsarchitekturen

Das Konzept der Virtualisierung ist nicht neu. Bereits 1974 formulierten Popek und Goldberg [PG74] drei Anforderungen für die Virtualisierung:

- Äquivalenz: Virtualisierte Systeme müssen exakt dasselbe Verhalten besitzen, wie dies bei der direkten Ausführung auf der Hardware der Fall ist.
- Kontrolle: Die Ressourcen eines Rechners müssen den virtuellen Maschinen einzeln kontrolliert zugänglich gemacht werden.
- Effizienz: Virtuelle Maschinen dürfen aufgrund der Virtualisierung keinen unangemessen Overhead produzieren, sondern sollen annähernd so schnell

sein, wie auf der direkten Hardware.

Durch die Effizienzanforderung, die eine große Herausforderung darstellt, wurden beim Design von Virtualisierungslösungen eine Vielzahl verschiedener architektonischer Ansätze ausprobiert. Je nach Architektur besitzen die Virtualisierungslösungen leicht unterschiedliche Eigenschaften und sind verschieden performant. Dabei lassen sich einige Ansätze, wie etwa Native Virtualisierung und Paravirtualisierung auch miteinander verbinden, um die Performance weiter zu steigern.

2.1.2.1 Native Virtualisierung

Die Native Virtualisierung stellt dem Gast-System Teilbereiche der physikalischen Hardware des Rechners als virtuelle Hardware zur Verfügung. Daher können nur Gäste benutzt werden, die zum Befehlssatz der Host-CPU kompatibel sind. Zugriffe auf andere Hardwarekomponenten werden bei der Nativen Virtualisierung über emulierte Standardkomponenten geregelt. Beispiele hierfür sind etwa Netzwerkkarten oder USB-Ports.

Virtualisierungssoftware, die diesen Ansatz unterstützt ist z.B. VirtualBox, VMWare Player, VMWare Server, Parallels Desktop/Workstation und QEMU mit dem optionalen Beschleuniger KQEMU. [WR10]

2.1.2.2 Vollvirtualisierung

Die Vollvirtualisierung (auch Betriebssystemneutrale Virtualisierung oder Full Virtualisation genannt) ist eine Virtualisierung vom Hypervisor Typ 1. Da der Hypervisor auf der nackten Hardware läuft und das Gastsystem nicht verändert werden muss, ist dies die „Königsdisziplin“ unter den Virtualisierungsarten. Bei der Vollvirtualisierung sind die Gäste voneinander sicher isoliert. Hierbei hat der Virtualisierungslayer volle Kontrolle über die Ressourcenverteilung der Hardware. Er muss sich nicht mit dem Host die Ressourcen teilen. Die Vollvirtualisierung kommt recht nah an die Effizienz der direkten Ausführung auf der Hardware heran. Verluste gibt es praktisch nur beim Kontextwechsel zwischen den virtuellen Maschinen und bei der Emulation nicht virtualisierbarer Befehle. Bei der x86-Architektur betrifft dies 17 Befehle (zum Beispiel Zugriffe auf den Interrupt-Controller oder die MMU), die abgefangen und emuliert werden müssen. Eine Garantie für Echtzeitverhalten kann hierdurch bei der x86-Architektur nicht gegeben werden. Ein weiterer Nachteil ist, dass Standard-Debugger nicht verwendet werden können, da diese mit den Interna des Hypervisors nicht zurechtkommen. Beispiel für die Vollvirtualisierung ist der ESX-Server von VMWare. [WR10]

2.1.2.3 Vollvirtualisierung mit Hardware-Unterstützung

Vollvirtualisierung mit Hardware-Unterstützung ist identisch mit der oben beschriebenen Vollvirtualisierung mit der Ausnahme, dass der Hypervisor auf Befehlssatzerweiterung (Intel VT oder AMD Pacifica) der CPU zurückgreifen kann, um sensitive Befehle zu bearbeiten. Auf diese Weise wird der Virtualisierungslayer schlanker und robuster, da keine dynamische Übersetzung mehr nötig ist. Auf der anderen Seite ist der Virtualisierungslayer dadurch hardwareabhängig. Xen und KVM nutzen beide die entsprechenden Befehlssatzerweiterungen der CPUs. [WR10]

Bei VMWare beherrscht der kostenpflichtige ESX-Server diese Form der Virtualisierung. Viele der entsprechenden Hardware-Unterstützungen sind jedoch nicht standardmäßig aktiviert.

VMWare bekommt in den letzten Jahren immer Druck von der Konkurrenz[2] und wird in dem Bereich der durch Hardware unterstützten Virtualisierung aktiver, wie die Kooperation mit Intel bei „Virtual Machine Device Queues“ belegen [CH07].

2.1.2.4 Paravirtualisierung

Bei der Paravirtualisierung wird das Gast-Betriebssystem angepasst, um alle kritischen CPU-Instruktionen (wie oben erwähnt, sind dies die Instruktionen, die im Kernel-Modus auf Ring 0 laufen) selbst abzufangen und umzukodieren, bevor die Instruktionen an den Virtualisierungslayer weitergereicht werden. Durch den Wegfall der Notwendigkeit, die Instruktionen, die im Kernel-Modus ausgeführt werden, im Virtualisierungslayer zu behandeln, ist eine schlanke und performante Implementierung des Virtualisierungslayers möglich. Im Gegensatz zur „Nativen Virtualisierung“ bietet die Paravirtualisierung dem Gast-Betriebssystem eine Schnittstelle, die ähnlich zu der darunter liegenden Hardware ist, an. Ein großer Nachteil dieser Art der Virtualisierung ist, dass das Gast-Betriebssystem angepasst werden muss. Vertreter dieser Virtualisierungslösung sind Parallels Workstation, XEN und VMWare Server. [WR10]

2.1.2.5 Virtualisierung auf Betriebssystemebene

Bei der Virtualisierung auf Betriebssystemebene teilen sich Host-System und Gast-Systeme einen einzigen Kernel. Hier wird nicht die Hardware, sondern das Betriebssystem an sich virtualisiert, indem es Instanzen seiner selbst erzeugt. Durch den resultierenden geringen Overhead ist die Ausführungsgeschwindigkeit

[2] Im Marktsegment sind die größten Rivalen Microsoft und Citrix.

sehr hoch. Im Idealfall muss — im Gegensatz zu Virtualisierungsansätzen mit virtuellen Maschinen — nur ein Betriebssystem aber nicht alle Gast-Betriebssysteme gepflegt und gewartet werden. [WR10]

2.1.3 Virtualisierungslösungen

Dieser Abschnitt stellt Virtualisierungslösungen vor. Im Abschnitt 2.1.3.1 werden Software-Virtualisierungslösungen vorgestellt. Die miteinander verknüpften Pakete qemu, kvm und libvirt werden dargestellt und ihre Beziehung zueinander geklärt. Im Abschnitt 2.1.3.2 Hardware wird als Beispiel einer Hardware-Unterstützung Intels „Vanderpool Technologies“ eingeführt.

2.1.3.1 Software

QEMU

QEMU emuliert die komplette Hardware eines Rechners inklusive der CPU und ihres Befehlssatzes (Hardware-Emulation). QEMU beschränkt sich nicht auf eine Architektur, wie sich zum Beispiel VMware auf die x86-Architektur beschränkt. QEMU nutzt die Dynamic Translation[3] und „[...] erzielt damit eine gute Geschwindigkeit [...]“ [WR10]. Optional konnte bis QEMU 0.11.x der Beschleuniger KQEMU benutzt werden (Native Virtualization). Da die Entwicklung von KQEMU eingestellt wurde, setzt QEMU seit Version 0.12 auf die Kernel-based Virtual Machine (KVM).

Mit QEMU können mehrmalig Snapshots einer virtuellen Maschine erzeugt und gespeichert werden. Ein Snapshot beinhaltet den gespeicherten Zustand der virtuellen Maschine. Wird dieser erneut geladen, so sind die nach dem Snapshot durchgeführten Änderungen unwirksam. QEMU hat einige Besonderheiten, die andere Virtualisierungssoftware nicht bietet. So unterstützt QEMU die Live-Migration[4], das Debuggen des Systems und ist fähig Hardwarefehler zu Testzwecken zu emulieren.

Mit dem Tool qemu-img, welches ebenfalls Teil des QEMU-Paketes ist, können Image-Dateien in unterschiedlichen Formaten angelegt, konvertiert und verschlüsselt werden. Da QEMU virtuelle Maschinen für die Kernel-based Virtual Machine

[3] Hierdurch kann die Ausführung von Byte-Code beschleunigt werden. Die virtuelle Maschine übersetzt kleine Einheiten wie etwa Funktionen in Maschinen-Code und speichert diese Übersetzung in einem Cache. Wird die Funktion nochmal benötigt, wird keine weitere Übersetzung erforderlich, sondern der im Cache gespeicherte Maschinen-Code benutzt.

[4] Live Migration ermöglicht das Verschieben von virtuellen Maschinen auf andere Hosts, während diese ausgeführt werden.

anlegen kann, ist dies somit auch auf Hardware möglich, auf der die „Kernel-based Virtual Machine“ nicht lauffähig ist.

Unter Linux, BSD und Mac OS X unterstützt QEMU auch die Userspace-Emulation[5]. Andere Programme, die nicht für die dynamischen Bibliotheken des Hosts kompiliert wurden, können so im Userspace betrieben werden. Unterstützte Prozessorarchitekturen sind x86, PowerPC, ARM, 32-bit MIPS, Sparc32/64, ColdFire(m68k), ETRAX CRIS und MicroBlaze.

Ein großer Vorteil von QEMU ist, dass es als Open Source-Software entwickelt wird. Viele andere Projekte von Virtualisierungslösungen (KVM, XEN, etc.) nutzen Teile von QEMU und im Gegenzug werden häufig Teile dieser Projekte wieder in QEMU integriert. So entwickelt sich QEMU ständig weiter. [WR10]

KVM

Die Kernel-based Virtual Machine (KVM) ist schnell populär geworden, da sie konsequent auf die Vorteile von Hardware-Virtualisierungstechniken, wie etwa Intels VT oder AMDs Pacifica, setzt. Weitere Hardware-Erweiterungen, wie Nested Paging[6] und IOMMU (I/O Memory Management Unit)[7] werden bereits genutzt. Ein weiterer Grund für den starken Popularitätsanstieg für KVM ist die Unterstützung der Management-Bibliothek libvirt (siehe Abschnitt 2.1.3.1.3).

Die Kernel-based Virtual Machine verwendet die Vollvirtualisierung (Typ-2-Hypervisor) und unterstützt normalerweise nicht die Paravirtualisierung. Dennoch existiert ein Patch, der die Paravirtualisierung von Linux-Gästen ermöglicht, jedoch noch nicht in den Mainstream-Linux-Kernel aufgenommen wurde. Paravirtualisierte Gerätetreiber für Gast-Systeme zu nutzen, ist möglich. Diese Treiber sind performanter und haben einen geringeren Overhead.

KVM ist ein Open-Source-Projekt und läuft unter Linux ab Kernel-Version 2.6.20. Die KVM besteht aus dem Kernel-Module kvm.ko, und den architekturspezifischen Modulen kvm-intel.ko oder kvm-amd.ko, sowie der Gerätedatei /dev/kvm. Sind die Module geladen, arbeitet der Linux-Kernel selbst als Hypervisor. KVM und QEMU werden häufig in einem Atemzug erwähnt, dies liegt teilweise daran, dass KVM keine virtuelle Hardware zur Verfügung stellt. QEMU,

[5] Siehe: 2.1.1.2 API-Emulation

[6] Nested Paging gibt Gästen Zugriff auf ihre eigenen Page Tables ohne den Hypervisor zu benutzen.

[7] Ein Gast-Betriebssystem weiß nicht welche physikalischen Speicheradressen es wirklich adressieren muss. Direkter Hardwarezugriff ist dadurch schwierig. Damit das Gast-System nicht falsch auf den Speicher zugreifen und Daten überschreiben kann, muss der Virtualisierungslayer für eine korrekte Umsetzung der Speicheradressen sorgen. Eine IOMMU löst dieses Problem durch *Remapping* der Hardware-Adressen zu dem selben *Translation Table* den auch die virtuelle Maschine benutzt.

als Emulator, kann diese Aufgabe natürlich erfüllen. Da die beiden Pakete sich gut ergänzen, sollen die Entwicklungen von KVM und QEMU zusammengeführt werden. Es gibt bereits von einigen Linux-Distributionen ein zusammengeführtes Paket (qemu-kvm). KVM läuft wegen der starken Nutzung von hardwarespezifischen Lösungen nur auf x86 Prozessoren und wird daher hauptsächlich zur Server-Konsolidierung eingesetzt.[WR10]

Libvirt

Viele Virtualisierungslösungen sind seit der Veröffentlichung des VMware Servers (Nachfolger des kostenpflichtigen GSX-Servers) inzwischen kostenlos. Das Geld wird nicht mit der Virtualisierung an sich, sondern mit Management-Tools für Virtualisierungslösungen verdient.

Libvirt ist kein eigentliches Management-Tool, sondern eine Abstraktionsebene, die zwischen der Virtualisierungssoftware und den verschiedenen Management-Tools angesiedelt ist. Über dieser Schicht verwalten die eigentlichen Management-Tools (Virtual Machine Manager, virsh) die unterschiedlichen Virtualisierungslösungen. Libvirt unterstützt sämtliche gängigen Virtualisierungslösungen, wie zum Beispiel QEMU, KVM, Xen, VirtualBox, VMware ESX, LXC Linux Container System, OpenVZ Linux Container System, und User Mode Linux. Selbst Windows-Nutzer können die Bibliothek mit Cygwin und Mingw unter Microsoft Windows kompilieren und nutzen. [WR10]

2.1.3.2 Hardware

Hardware-Erweiterungen: Intels Vanderpool Technologies

Bei der Virtualisierung gehen beide Betriebssysteme, das Host-System und das Gast-System, davon aus, die absolute Kontrolle über die Hardware zu haben und in Ring 0 (siehe Abbildung 2.3) zu laufen. Dies führt unweigerlich zu problematischen Instruktionen, die der Virtualisierungslayer abfangen und entsprechend berücksichtigen muss. Intels Vanderpool Technologies (für IA32 VT-x, für Itanium VT-i genannt) und AMDs Pacifica sind beides Hardware-Unterstützungen für die Virtualisierung, die helfen sollen, einen schlanken, kleinen und somit performanten Virtualisierungslayer entwickeln zu können. Die Notwendigkeit für CPU-Paravirtualisierung oder Übersetzungen von Binärcodes sollen so ganz entfallen.

Ein VMM, der auf einem IA-32 oder Itanium Kern läuft, muss „Ring-Deprivileging“ betreiben.

Definition 2.8 (Ring-Deprivileging)
Ring-Deprivileging bezeichnet eine Technik, mit der alle Gast-Betriebssysteme mit

einer geringeren Priorität (1-3) als 0 (höchste Priorität, reserviert für Betriebssystemkern) ausgeführt werden. Man unterscheidet zwei Modelle:

- *0/1/3, der Gast-Kernel wird in Ring 1 ausgeführt,*
- *0/3/3, der Gast-Kernel wird in Ring 3 ausgeführt und hat die gleiche Priorität wie eine normale Anwendung.[NSL+06]*

Definition 2.9 (Ring Aliasing)
Ring Aliasing tritt auf, wenn Software auf einer anderen Privilegierungsebene ausgeführt wird, als derjenigen, für die sie entwickelt wurde.[NSL+06]

Bei der IA32 Architektur speichert eine PUSH Instruktion, die mit dem CS-Register ausgeführt wird, den Inhalt des Code-Segment-Registers auf dem Stack. Teil hiervon ist jedoch auch die aktuelle Privilegierungsebene (Current-Privilege-Level - CPL). Auf diese Weise kann ein Gast-Betriebssystem leicht feststellen, dass die Privilegierungsebene, in der es läuft, ungleich 0 ist. [NSL+06]

Mit VT-x (und VT-i) wurden VMX root und VMX non root Operationen eingeführt. VMX root Operationen sind für den Virtualisierungslayer gedacht und verhalten sich so wie die Befehle unter nicht VT erweiterten Prozessoren zuvor. VMX non-root Operationen stellen eine alternative IA32 Umgebung dar, die durch einen Virtualisierungslayer kontrolliert wird und so die virtuellen Maschinen unterstützen soll. Beide Umgebungen unterstützen jeweils die in Abbildung 2.3 gezeigten vier Schutzebenen. Auf diese Weise hat ein Gast-System die von ihm vorausgesetzen vier Privilegierungsebenen, wie auch die VMM. VT-x definiert zwei neue Transitionen:

- *„VM-entry“*, Übergang von VMX root nach VMX non-root; lädt automatisch den Prozessorstatus aus der „guest-state area“ und
- *„VM-exit“*, Übergang von VMX non-root nach VMX root; speichert den aktuellen Status in der guest-state area und lädt automatisch den Prozessorstatus aus der „host-state area“.

Überwacht werden können VM entries und VM exits durch den sogenannten „Virtual-machine Control State“ (VMCS), der die „host-state area“ und „guest-state area“ kontrolliert. [NSL+06]

2.2 Maßschneiderung

Das Dilemma für Entwickler von Betriebssystemen besteht in den sich widersprechenden Zielen Korrektheit einer Operation für alle Anwendungen zu gewährleisten und gleichzeitig eine hohe Performance für die einzelne Anwendung zu liefern.

Dieses wird verschärft, durch die Vielzweck-Betriebssysteme, die wie in der Einleitung erwähnt, *allen* Anwendungen in den unterschiedlichsten Anforderungsbereichen, von Clustern bis zum heimischen PC, gerecht werden wollen.

Der Standardansatz, der von Betriebssystementwicklern gewählt wird, ist einen generalisierten Code mit einigen festen Optimierungsvarianten für den allgemeinen Fall (*common case*) zu konstruieren.[MWP+01]

Verschiedene alternative Ansätze werden in der Forschung untersucht, um den verschiedensten Anforderungen von Anwendungen möglichst exakt zu entsprechen.

Ein etwas älterer, dennoch vielversprechender Ansatz ist die Konfiguration (*Customization*) der Systemstruktur.

Obwohl der Linux-Kernel bereits über eine Vielzahl von Konfigurierungsmöglichkeiten verfügt, so ist die Anzahl der Konfigrationsmöglichkeiten von Betriebssystem-Eigenschaften beim Linux-Kernel stark begrenzt. Auf Seiten der Programmierung mit unzähligen IFDEF-Guards zu arbeiten, erhöht nicht die Lesbarkeit und Wartbarkeit des Linux-Kernels.

Gerade im Bereich eingebetteter Systeme, ein Bereich, in dem es aufgrund der beschränkten Hardwareressourcen und Anforderungen viele Betriebssysteme gibt, lassen sich gut Produktlinien von Betriebssystemen entwickeln. Hier werden bereits Techniken, wie etwa „Aspektorientierte Programmierung“, „Generische Programmierung“ und „Objektorientierte Programmierung“ erfolgreich miteinander kombiniert, um querschneidende Belange[8] konfigurierbar zu machen. [Spi08b]

Bei der Maßschneiderung kann man die Verfahren in manuelle und automatisierte Verfahren unterteilen.

Bei der Maßschneiderung auf Ebene des Quellcodes herrschen maunelle Verfahren vor. Hier wird explizites Fachwissen über die zu maßschneidernde Anwendung benötigt. Einen direkten Überblick über sämtliche Maßschneiderungstechniken zu geben, ist daher nicht möglich. Bekannte Techniken sind jedoch das Einfügen von Puffern, zum Beipiel bei dem Schreiben auf die Festplatte oder das Entfernen von unerwünschtem und überflüssigen Code oder das Entfernen von nicht benötigtem Code.

Als automatisiertes Verfahren auf der Quellcodeebene kann der GNU C Compiler gelten, der bei der Erstellung des Maschinencodes, viele technisch geprägte Optimierungen, wie das Inlining von Funktionen vornehmen kann.

[8] Belang bezeichnet einen Bereich speziellen Interesses.
Querschneidender Belang ist ein nicht modularisierbarer Belang (er schneidet quer durch sämtliche Module und Funktionen — beispielsweise Energiemanagement, Debugging, Zugriffsrechte-Verwaltung).

Ein weiteres Unterscheidungsmerkmal ist das Optimierungsziel nach dem man Maßschneiderungslösungen unterscheiden kann. Gerade im Embedded-Bereich interessiert man sich nicht nur für die Leistung, sondern auch für Minimierung der Codegröße (siehe zum Beispiel [CSB+03, CSB+05]) und ebenso für den Energieverbrauch von Software.

Die oben angedeuteten Ebenen der Maßschneiderung werden in Abschnitt 2.2.1 vertieft.

2.2.1 Granularitäten

Bei der systematischen Vorgehensweise, ein Betriebssystem maßzuschneidern, bemerkt man schnell, dass es verschiedene Ebenen der Maßschneiderung (Granularitäten) gibt. Auf der Paketebene entscheidet man, ob installierte Software benötigt wird oder nicht. Es gibt eine Menge Programme, die diese Arbeit erleichtern und automatisert erledigen können. Hat man unnötig installierte Programme und Dienste entfernt, so erhält man eine Umgebung, in der nur noch die zu benutzende Anwendung, sowie direkte Hilfsprogramme, DNS Dienste oder DHCP Client Software benötigt wird. Hierdurch schafft man eine Umgebung, die nur noch die Anforderungen der maßzuschneidernden Anwendung und damit weniger Komplexität besitzt.

Die Maßschneiderung auf der Kernelkonfigurationsebene dient dazu, das Betriebssystem genau auf die Umgebung anzupassen. Im Falle von Virtual Applications, die meist wenige oder sogar nur eine Anwendung beinhalten und zu einem ganz bestimmten Zweck erstellt werden, sind bestimmte Rahmenbedingungen vorher bekannt. Zu diesen Rahmenbedingungen gehören zum Beispiel Hardwareanforderung, wie etwa wie viele CPUs oder wieviel RAM zugewiesen wird, beziehungsweise welche Hardware von der virtuellen Hardware für den gewünschten Einsatz gestellt werden muss. Konfigurationen auf dieser Ebene entfernen unter Umständen weitere Komplexität (Entfernen von Multi-Prozessor Code per Konfiguration[9]) aus dem Betriebssystemkernel. Jede Zeile Code, die gespart werden kann, erhöht die Ressourcen, die der Anwendung zur Verfügung stehen.

Die Konfigurierungsmöglichkeiten des Linux Kernels sind schon recht beachtlich. Große Subsysteme wie TCP/IP V6 Support oder Mehrprozessor Unterstützung, können so recht komfortabel aus dem Kernel entfernt werden. Automatisierte Unterstützungslösungen, die automatisch die richtigen Einstellungen setzen, sind nicht vorhanden. Das Fachwissen des Administrators, der die Anforderungen seiner Anwendung kennt, ist hier unerlässlich.

Trotzdem ist diese Form der Maßschneiderung immer noch recht grob und nicht direkt auf den Anwendungsfall der Anwendung, sondern nur auf dessen

[9]Nur ein Prozessor ist der VM zugewiesen.

Umgebung zugeschnitten. Aggressiveres Optimieren ist mit Maßschneiderungen auf Code-Ebene möglich. Je stärker die Optimierung betrieben werden soll, desto detailiertere Informationen werden benötigt. Um die betreffenden Informationen zu sammeln, können verschiedene Profiler benutzt werden. Durch diese Form der Optimierung sollten gute Performance-Gewinne zu erzielen sein.

Da die Anwendung sich innerhalb einer virtuellen Maschine befindet, muss diese Virtualisierungsebene mit betrachtet werden.

In Abschnitt 2.2.1.1 werden verschiedene Programme vorgestellt, die bei der Optimierung auf der Paketebene helfen. Analog dazu stellt der Abschnitt 2.2.1.2 Kernelkonfigurationsebene, mögliche Werkzeuge zur Optimierung auf dieser Ebene vor. Ein Werkzeug zur Optimierung auf der Codeebene ist der GCC. Dieser erledigt automatisch Maßschneiderungerungen auf der rein technischen Ebene, wie etwa das Abrollen von Schleifen oder das Anpassen des Maschinencodes auf den zugrundeliegenden Prozessor. Auf der Virtualisierungsebene gibt es keine Unterstützung durch Werkzeuge im Sinne einer automatisierten Unterstützung.

2.2.1.1 Paketebene

Die gröbste Ebene der Maßschneiderung ist die Paketebene. Unterstützung bieten hier Programme wie Deborphan, Debfoster und SystemTap.

Mit Debfoster kann man sich eine Menge von bewusst installierter Software und deren Abhängigkeiten anzeigen und automatisch entfernen lassen.

Hat man überflüssige Prozesse gefunden und deinstalliert, so lassen sich mit Deborphan nicht mehr benötigte Pakete finden. Eine Reihe von installierten Prozessen werden auf einer virtuellen Maschine mit einem Webserver nicht benötigt, sind aber seit der Standardinstallation dabei. So kann das Paket exim4[10] genauso entfernt werden, wie ein DHCP-Client[11].

2.2.1.2 Kernelkonfiguration

Der Linux Kernel bietet eine Vielzahl an Konfigurationsmöglichkeiten. Verschiedene Arbeiten und Bücher versuchen die Menge an Konfigurationsmöglichkeiten zu veranschaulichen und zu erklären.

Eine gute Einführung zur geführten statischen Kernelkonfiguration findet sich in [KH06]. Hier werden nicht nur die wichtigsten Gruppen von Kernelkonfigurationen vorgestellt, sondern auch Skripte beschrieben, die zum Beispiel helfen, die aktuell im Kernel geladenen Module zu finden. Auf diese Weise können leicht

[10] E-Mail-Client
[11] Überflüssig, da eine statische Konfiguration möglich ist.

nicht benötigte Module identifiziert und entfernt werden. Desweiteren hat man einen Überblick über die verwendeten Module und kann ferner diese auf Wunsch direkt im Kernel integrieren, so dass keine Module mehr nötig sind.

Dynamische, vollautomatisierte kommerzielle Lösungen gibt es kaum. Einige Programme wie MeasureWare oder SARCHECK[12] versprechen dem Nutzer eine signifikante Performancesteigerung durch die Untersuchung von Kernel-Parametern zur Laufzeit. Eigentlich für HP-UX Systeme entwickelt, verspricht SARCHECK auf der Webseite auch den Einsatz unter Solaris, sowie Linux. SARCHECK sammelt mit einem eigenen Profiler namens „sar“ Daten über die auf dem Server laufenden Anwendungen. Nach dem Erreichen einer entsprechend großen Datenbasis, ca. 1-2 Tage später, können sogenannte Reports erzeugt werden. Diese Reports beinhalten detaillierte Angaben zum gefundenen Problem, dem Vorschlag zur Behebung des Problems und meistens einer Schritt-für-Schritt-Anleitung, wie die entsprechenden Parameter, meist mit Hilfe von procfs zu setzen sind (siehe Abbildung 2.4).

RECOMMENDATIONS SECTION

All recommendations contained in this report are based solely on the conditions which were present when the performance data was collected. It is possible that conditions which were not present at that time may cause some of these recommendations to result in worse performance. To minimize this risk, analyze data from several different days and implement only regularly occurring recommendations.

Consider adjusting process priorities with the nice(C) command, optimizing applications, or improved job scheduling. Occasionally heavy CPU utilization was seen and these recommendations may help to avoid the need for a hardware upgrade.

Change the bdflush parameter 'nfract' from 50 to 55. This is the percentage of dirty buffers allowed in the buffer cache before the kernel flushes some of them.

Change the bdflush parameter 'nfract_sync' from 80 to 85. This is the percentage of dirty buffers in the buffer cache before the kernel agressively flushes them synchronously.

Change the bdflush parameter 'nfract_stop_bdflush' from 50 to 45. This is the percentage of dirty buffers in the buffer cache used to decide when to stop bdflush.

To change the value of the bdflush parameters immediately as described in the above recommendations, use the following command:

```
echo "55 500 8 0 500 3000 85 45 0" > /proc/sys/vm/bdflush
```

With some kernels, this will not work because the file /proc/sys/vm/bdflush is read-only and you may not be able to change its permissions. If you are able to make this change and it improves performance, you can make the change permanent by adding the command to the /etc/rc.d/rc.local file.

Abbildung 2.4: Auszug aus einem SARCHECK Report[13]

Die Entwickler von SARCHECK werben mit den „geringen Kosten“, die 450 US$ pro Server betragen. Eine Testversion zu bekommen, war nicht möglich.

Das Programm soll automatisch die besten Parameter für Kernelkonfigurationen finden, die die ausgewählte Anwendung beschleunigen.

[12]Siehe http://www.sarcheck.com

2.3 Tools

Die in dieser Arbeit benutzten Werkzeuge werden in diesem Abschnitt motiviert und ihre Funktionsweisen erklärt. In Abschnitt 2.3.1 werden die Besonderheiten beim Kernel-Debugging erklärt und das Debugging mit Hilfe von KProbes motiviert. Diese werden in Abschnitt 2.3.2 beschrieben und bilden ihrerseits die Grundlage für das in dieser Arbeit verwendete SystemTap (siehe Abschnitt 2.3.3). Im Abschnitt 2.3.5 werden Vorgehensweisen zur Minimierung der Kernel Konfiguration erläutert. Abschließend wird in Abschnitt 2.3.6 KSplice vorgestellt. Dieses Werkzeug wird später zum Beschleunigen des Workflows beim Testen benutzt. Veränderter Quellcode wird mit KSplice direkt in den Binärdateien des Kernels geändert und erlaubt so beim Messen später, die gefundenen Optimierungen „On-the-Fly“ austauschen zu können. Die Zeit bis zum erneuten Messen kann auf diese Weise gering gehalten werden.

2.3.1 Debugging

Das Debugging des Linux-Kernels unterscheidet sich vom Debugging im User-Space vor allem durch die Schwierigkeiten beim Debugging. Im Vergleich zum User-Space gibt es keine übergeordnete Schicht, die die Ausführung des Kernels kontrolliert, auch läuft der Kernel in einem eigenen geschützten Adressraum. Der Einsatz eines normalen Debuggers, wie etwa dem GNU-Debugger gdb, ist zwar möglich, jedoch ist damit nicht der volle Debugging-Umfang nutzbar. Mit `gdb vmlinux /proc/kcore` kann man den Debugger für einen laufenden Kernel genauso starten, wie für jeden anderen laufenden Prozess. vmlinux bezeichnet hierbei das unkomprimierte Kernel-Image. `/proc/kcore` fungiert als Core-File, um dem gdb Einblick in den Speicher des laufenden Kernels zu ermöglichen.

Es ist möglich, sämtliche gdb-Befehle auszuführen, um Informationen, wie etwa den Wert einer Variablen, auszulesen. Nicht nutzbar sind gdb-Befehle, die Daten im Kernel verändern. Eine Einzelschrittausführung des Kernel-Codes ist genauso wenig möglich, wie das Setzen von Breakpoints. [LK05]

2.3.2 KProbes

KProbes wurden als Debugging- und Monitoring-Mechanismus für den Linux-Kernel von IBM entwickelt. Mit KProbes kann das Auftreten bestimmter Ereignisse (*Events*) protokolliert und das Betreten oder Verlassen von Funktionen und somit der Kontrollfluss verfolgt werden. KProbes eignen sich nicht nur zum Debuggen, sondern ebenso zur Identifizierung von Performance-Engpässen (*Bottlenecks*). IBM entwickelte KProbes als unterliegenden Mechanismus für ihr Tracing-Werkzeug DProbe. KProbes erlauben dem Benutzer dynamisch, an beliebigen

Stellen im Linux-Kernel Sonden (*Probes*) einzuführen. Vor dem Ausführen der untersuchten Instruktion wird ein spezieller Pre-Handler aufgerufen. Optional kann nachher ein Post-Handler aufgerufen werden.

Zurzeit werden vom Linux-Kernel drei Arten von Sonden (*Probes*) unterstützt: KProbes, JProbes und KRetProbes. Die drei Arten von Sonden unterscheiden sich hinsichtlich ihres Verhaltens und der Granularität. Eine KProbe kann an einer beliebigen Instruktion im Kernel eingefügt werden. Eine JProbe wird beim Einstieg (*Entry*) in eine Kernel Funktion eingefügt und liefert so einen einfachen Zugang zu den Parametern einer Funktion. Eine KRetProbe oder auch einfach nur „return probe“ genannt, wird erst beim Verlassen einer Funktion ausgelöst.[KPH10]

2.3.2.1 Funktionsweise

Wenn eine KProbe registriert wird, kopiert diese die ersten Bytes der zu untersuchenden Instruktion und ersetzt die zu untersuchende Instruktion durch eine Breakpoint-Anweisung (archtitekturabhängig, z.B. int3 bei i386, x86_64). Sobald die CPU die Breakpoint-Anweisung erreicht, wird eine Trap ausgelöst, die Register der CPU werden gesichert und die KProbe wird ausgeführt. KProbes führen als erstes ihren Pre-Handler aus, der im Einzelschritt (*Single Stepping*) die Kopie der zu untersuchenden Anweisung ausführt. Bei der Entwicklung der KProbes wurde bewusst darauf geachtet, dass jeder Breakpoint unter allen Umständen erreicht wird. Würde man nicht die Kopie der Anweisung, sondern die originale Anweisung direkt ausführen, so würde temporär der Breakpoint entfernt und somit ein kleines Zeitfenster offen sein, in dem andere CPUs den Breakpoint nicht beachten und die Ausführung nicht anhalten. Nach der Einzelschritt-Ausführung wird, sofern vorhanden, der Post-Handler ausgeführt. Treten während der Behandlung des Pre- oder Post-Handlers Fehler (z.B. Seitenfehler) auf, so wird der Fault-Handler aufgerufen, der dann den Fehler behandelt. [KPH10]

2.3.3 SystemTap

Will man im Kernel Daten sammeln, so gibt es verschiedenste Ansätze, von Debugging mittels `printk()`-Anweisungen bis hin zu KProbes. Das Ziel von KProbes war es, einen Mechanismus zur Verfügung zu stellen, mit dem man an jeder Stelle im Kernel nicht nur Daten lesen, sondern auch dynamisch Breakpoints setzen und sogar die Umgebung durch Verändern von Kernel-Daten anpassen kann.[14]

Selbst das dynamische Einspielen von Testszenarien, sowie das Testen von Bugfixes sind durch KProbes ermöglicht worden. Der Einsatz von KProbes ist zumindest für Testzwecke recht aufwendig. So muss erst eine KProbe als Kernel-Modul

[14]Siehe: http://www.ibm.com/developerworks/library/lkprobes.html

programmiert, kompiliert und im Kernel registriert werden, bevor Daten gesammelt werden können.

SystemTap ist eine auf KProbes basierende Skriptsprache und bietet Entwicklern und Systemadministratoren die Möglichkeit, wiederverwendbare Skripte zu schreiben, die automatisch KProbes erzeugen und diese im Kernel registrieren. Selbst SystemTap-Neulinge können nach der Installation vom Paket SystemTap sofort starten, indem Sie ein Skript von der SystemTap-Homepage herunterladen und dieses mit SystemTap aufrufen.

Ein SystemTap Script besteht aus Ereignissen (*Event*) und Behandlungsroutinen (*Handler*). Immer wenn ein Ereignis ausgelöst wird, ruft der Linux-Kernel den Handler auf und kehrt zurück. Ereignisse können Eintritt (*Enter*) in eine Funktion oder Austritt (*Exit*) aus einer Funktion, ein abgelaufener Timer oder sogar die Bedingung, ob ein Ausführungsfaden (*Thread*) im Kernel- oder User-Kontext läuft, sein. SystemTap automatisiert den Vorgang eine KProbe als Modul zu programmieren, diese zu kompilieren und in den Kernel einzufügen. Sobald SystemTap eine KProbe als Modul in den Kernel geladen hat, fängt diese alle Ereignisse, die die zu untersuchende Instruktion betreffen, ab, und ruft die entsprechenden Handler auf, die dann übergebene Argumente auf der Konsole ausgeben, Zähler für das Auftreten dieses Ereignisses erhöhen oder Zeiten messen können. [DC09]

```
# cat strace-open.stp
probe syscall.open
{
  printf ("%s(%d) open (%s)\n", execname(), pid(), argstr)
}
probe timer.ms(4000) # after 4 seconds
{
  exit ()
}

# stap strace-open.stp
vmware-guestd(2206) open ("/etc/redhat-release", O_RDONLY)
hald(2360) open ("/dev/hdc", O_RDONLY|O_EXCL|O_NONBLOCK)
hald(2360) open ("/dev/hdc", O_RDONLY|O_EXCL|O_NONBLOCK)
hald(2360) open ("/dev/hdc", O_RDONLY|O_EXCL|O_NONBLOCK)
df(3433) open ("/etc/ld.so.cache", O_RDONLY)
df(3433) open ("/lib/tls/libc.so.6", O_RDONLY)
df(3433) open ("/etc/mtab", O_RDONLY)
hald(2360) open ("/dev/hdc", O_RDONLY|O_EXCL|O_NONBLOCK)
```

Abbildung 2.5: Strace Simulation für den `Open()` Syscall [Eig10]

Abbildung 2.5 vermittelt ein Bild davon, wie leicht es ist, ein Skript mit SystemTap zu erstellen. Die zu untersuchende Stelle wird mit dem Schlüsselwort

`probe` bezeichnet und kann sowohl eine einzelne Instruktion oder Funktion wie im obigen Beispiel sein, als auch eine Menge von Instruktionen oder Funktionen, die dann per Platzhalter angesprochen werden.

Der zu untersuchende Event läßt sich leicht in der SystemTap-Syntax ausdrücken. So kann der Rücksprung aus einem `open()`-Systemaufruf mit `syscall.open.return` abgefangen werden. Durch die Verwendung der Befehle `kernel.function(*@net/socket.c).call` erfolgt das Abfangen des Einstiegs und mit `kernel.function(*@net/socket.c).return` erfolgt das Abfangen des Rücksprungs in eine beliebige Funktion. In diesem Beispiel wurden ebenso Platzhalter für den Funktionsnamen benutzt. So liefert `*@` alle Funktionsnamen in dem dahinter definierten Quelltext. Es ist ebenso möglich, direkt an eine Adresse zu springen, und dort eine KProbe zu installieren. Mit `module("ext3").statement(0xdeadbeef)` wird direkt die Anweisung an Adresse deadbeef im Ext3 Dateisystem instrumentiert. Gerade wegen der einfachen Benutzung der SystemTap-Skriptsprache, sollte man beim Einsatz von Platzhaltern aufpassen. SystemTap erzeugt für jede auf den Ausdruck passende Anweisung oder Funktion eine eigene KProbe. So können bei zu argloser Benutzung schnell hunderte oder tausende von KProbes unabsichtlich erzeugt werden.

Nachdem man festgelegt hat, welche Funktionen instrumentiert werden sollen, hat man eine Vielzahl von Ausgabemöglichkeiten. Die häufigsten finden sich in Tabelle 2.1

Funktion	Bedeutung
`tid()`	ID des aktuellen Threads
`pid()`	Prozess (Task Group) ID des aktuellen Threads
`uid()`	ID des aktuellen Benutzers
`execname()`	Name des aktuellen Prozesses
`cpu()`	Cpu, auf der der Thread ausgeführt wird
`gettimeofday_s()`	Zeit seit Beginn der Epoche
`get_cycles()`	Wert des Hardware-Zyklen-Zählers (*Hardware Cycle Counter*)
`pp()`	String, der die aktuelle Probe beschreibt
`probefunc()`	Funktion gibt, wenn vorhanden, den Namen der Funktion in der die Probe platziert wurde, zurück

Tabelle 2.1: SystemTap - Verschiedene Ausgabemöglichkeiten kurz vorgestellt

2.3.4 OProfile

Flaschenhälse in der Prozessor-Performance zu finden, ist nicht leicht. OProfile kann konfiguriert werden, um Daten inklusive eines Zeitstempels zu sammeln,

um herauszufinden welcher Code wie lange für seine Ausführung braucht. Unter vielen Architekturen, wie auch der x86-Architektur, hat OProfile Zugriff auf von der CPU bereitgestellte Performance-Counter. Ausgelesen werden können zum Beispiel Cache-Misses, Prozessor-Zyklen oder ähnliches. Eine vollständige Liste der vom Prozessor unterstützten Performance-Counter (OProfile nennt diese Events) erhält man, wenn man mit installierten OProfile folgenden Befehl ausführt: `opcontrol -list-events`

Durch die spätere Auswertung der Daten lassen sich recht genau eventuelle Flaschenhälse identifizieren. [Coh04] Ähnlich wie mit SystemTap lassen sich auch mit Oprofile Callgraphen und annotierter Quellcode als Auswertungsmöglichkeiten zum Aufspüren von Flaschenhälsen benutzen.

2.3.5 Kernel Konfiguration

Einen minimalen Kernel, der auf die Bedürfnisse der virtuellen Maschine zugeschnitten ist, zu konfigurieren, ist anhand der Menge an Konfigurationsmöglichkeiten nicht leicht.

Das Bauen eines neuen Kernels nimmt viel Zeit in Anspruch, weswegen es wichtig ist, nicht einfach nur sukzessiv alles herauszunehmen, was nicht mehr benötigt wird und den Kernel währenddessen ständig neu zu kompilieren, zu installieren und anschließend zu starten, sondern strukturiert an diese Aufgabe heranzugehen.

Eine gute Einführung in das Thema „Kernel selber bauen“, liefert „Linux kernel in a nutshell“ von Kroah-Hartman [KH06]. In dieser Publikation werden nicht nur die Grundlagen zum Kernel neu kompilieren erklärt, also welche Pakete für das Kompilieren erforderlich sind, wie man einen neuen Kernel installiert, etc., sondern auch Vorgehensweisen erklärt, mit Hilfe derer man den Linux-Kernel schnell anpassen kann.

Viele dieser Vorgehensweisen werden direkt mit Beispielskripten erklärt, wie zum Beispiel zur leichteren Lokalisierung von benötigten Modulen. Der Leser erhält so direkt Mittel, um sofort seinen eigenen Kernel bauen zu können.

2.3.6 KSplice

Um laufende, hochverfügbare Systeme mit Linux-Kernel ohne Neustart auf dem neuesten Stand zu halten, wurde KSplice entwickelt. KSplice beschreitet neue Wege, indem es anders als bisherige „Hot-Update“ Systeme, auf Objektcodeebene arbeitet. Trotz der vielen Herausforderungen, die damit einhergehen, schafft KSplice größtenteils vollautomatisch die Umwandlung normaler Quellcode-Patches in „Hot-Updates“.

In dem Fall, dass ein Patch keine Semantik von dauerhaft (*persistant*) geladenen Datenstrukturen verändert, kann die Umwandlung in ein „Hot Update“ automatisch erfolgen. Sollten persistente Datenstrukturen verändert werden, so muss manuell Code in den Patch eingefügt werden, um den Patch in ein „Hot Update“ zu verwandeln. In 88% der Fälle (56 Patches) waren sicherheitskritische Patches in dem Zeitraum von Mai 2005 bis Mai 2008 vollautomatisch umzuwandeln. Bei den restlichen 12% (8 Patches) mussten lediglich 17 Zeilen Quellcode ergänzt werden. So konnten alle in dem genannten Zeitraum bekannten Patches ohne Neustart des Kernels benutzt werden. [AK09]

Bisher standen Administratoren vor dem Problem, für jedes Einspielen eines Kernel Patches den Kernel neu zu kompilieren, zu installieren und den Rechner erneut zu starten. Für hochverfügbare Rechner, die wichtige Webseiten oder unternehmenskritische Anwendungen bereit stellen, ist eine längere Zeit (meist wird „lange“, schon mit wenigen Minuten definiert) in der sie nicht verfügbar sind, untragbar.

Desweiteren verursacht gerade bei großen Firmen die Wartung der Server hohe Kosten. Je mehr Server, desto komplizierter und teuerer die Wartung. KSplice bietet inzwischen einen kostenpflichtigen Service, KSplice Uptrack an, der es erlaubt, fertige, getestete Hot-Updates vollautomatisch einspielen zu lassen. Einen Server ein Jahr lang automatisch mit Updates zu versorgen, kostet zurzeit ca. 48 US$. [Ksp10]

3 Analyse

Will man das Betriebssystem an eine Anwendung anpassen, benötigt man als erstes die Anforderungen der Anwendung. Um herauszufinden, welche Ansprüche eine Anwendung an das Betriebssystem stellt, können statische und dynamische Analysetechniken benutzt werden. Die statische Analyse, bei der durch fachbezogenes Spezialwissen Anforderungen abgeleitet werden können, liefert strukturelle Maßschneiderungsmöglichkeiten. Man stellt bei dieser Methode zum Beispiel fest, welche Indirektionen Ballast darstellen oder welche großen Subsysteme, wie etwa die Rechteverwaltung, entfernt werden können.

Dynamische Analysetechniken, wie Profiler, können zur Laufzeit der zu untersuchenden Anwendung weitere Optimierungschancen aufdecken. Mit Hilfe von Profilern ist es möglich, die Anforderungen der Anwendung genauer zu bestimmen, um so zum Beispiel feststellen zu können, ob eine Anwendung stark CPU-gebunden (*CPU-Bound*) oder Eingabe-Ausgabe-orientiert (*I/O-Bound*) ist. Die einzelnen Systemfunktionen, die von der Anwendung benutzt werden, können zum Beipiel mit Hilfe von Aufrufgraphen genauer untersucht und kritische Stellen bestimmt werden.

Eine genaue Untersuchung, der vom Anwendungsprogramm benutzten Kernelfunktionen, stellt einige Anforderungen an die Werkzeuge, mit deren Hilfe man die Anwendung analysieren will. Für ein detailliertes Profiling des Betriebssystems ist es wichtig, dass man überall im Kernel messen kann. Die Bordmittel von Linux sind für die meisten Fragestellungen im Bereich Profiling nicht geeignet. Meist möchte man auch mehr wissen als die Tatsachen, welcher Systemaufruf (*system call*) benutzt wird und wie lange dieser ausgeführt wurde. Des Weiteren sollte die Messmethode nur einen geringen Overhead besitzen, um die Messungen nicht zu stören. Der wichtigste Punkt im Sinne des Workflows ist aber, dass es möglich sein muss, das Profiling schnell und flexibel an neue Fragestellungen anzupassen.

KProbes sind von IBM für den Zweck entwickelt worden, an jeder beliebigen Stelle im Betriebssystem dynamisch Breakpoints setzen und Debugging betreiben zu können. KProbes können recht unhandlich sein, da ihre Programmierung für jedes neue Testszenario manuell angepasst werden muss.

SystemTap, eine auf KProbes aufbauende Skriptsprache, benutzt KProbes nicht mehr zum Debugging, sondern zum Profiling.

Keniston et al. [KMPP07] vergleichen mehrere Profiling-Werkzeuge auf ihre Tauglichkeit zur Untersuchung von vielfädigen (*multi-threaded*) Prozessen. Das Hauptproblem bei der Benutzung von User-Space-Werkzeugen ist der Overhead gerade bei zeitkritischen Messungen. So besitzen einige dieser Werkzeuge die im User-Space arbeiten, wie etwa strace einen Overhead von 25 μs. Beim Vergleich der Profiling-Werkzeuge besitzt das KProbes-Interface mit 0,25 μs den zweitkleinsten Overhead. Es ist im Gegensatz zu dem leicht besseren UTrace jedoch in der Lage, jede Stelle im Kernel instrumentieren zu können und einen Trace auf Funktions- und nicht wie DTrace nur auf Syscall-Ebene erzeugen zu können. Zusätzlich gibt es SystemTap, einen Mechanismus, der auf KProbes aufbaut und eine Event basierte Skriptsprache zur Verfügung stellt. Da das zu untersuchende Objekt in dieser Arbeit, der quelloffene Linux-Kernel ist, können KProbes und SystemTap sehr gut zum Profiling benutzt werden.

Rein technische Optimierungen auf Quellcode-Ebene, wie zum Beispiel Loop-Unrolling oder Inlining, sind Aufgaben, die einer automatisierten Lösung wie etwa dem GCC Compiler überlassen werden können. Der GCC Compiler wird ständig weiterentwickelt und beherrscht inzwischen eine Vielzahl an Optimierungsvarianten.

3.1 Methode

Die hier vorgestellte Methode sollte zuerst nur die Performance der Anwendung beziehungsweise die Ausführungsdauer der Systemaufrufe im Kernel messen und analysieren können. Es stellte sich jedoch heraus, dass bei virtuellen Maschinen eine isolierte Betrachtung nicht ausreicht. Die Performance hängt zu einem nicht unerheblichen Teil von der Wahl der Virtualisierungslösung ab.

Für die Untersuchung auf Maßschneiderungsmöglichkeiten, wurde sowohl eine statische Analyse des Linux-Kernels, als auch eine Profiler gestützte Analyse verwendet. Die Analyse verlief nicht sequentiell sondern iterativ. Ergebnisse aus dem Profiling, beispielsweise dass die Anwendung stark Eingabe-/Ausgabe-Operationen benutzt, fokussierten die Suche nach Maßschneiderungsmöglichkeiten bei der Analyse des Betriebssystemdesigns.

Der Workflow soll vor allem die schnelle Umsetzung von Maßschneiderungsmöglichkeiten ermöglichen. McNamee et al. [MWP+01] nennen als Gründe, warum die Spezialisierung außerhalb der Wissenschaft wenig beachtet wird, dass die häufig manuellen und komplizierten Verfahren zeitraubend und fehleranfällig seien. Dieser Workflow soll daher die schnelle Umsetzung von Maßschneiderungsmöglichkeiten unterstützen.

Die Analyse betrachtet mit Hilfe eines SystemTap-Skripts den Teil der innerhalb der VM passiert. SystemTap erlaubt schnell, vollautomatisch neue Tests zu

generieren. Ein langwieriger Workflow, wie er beispielsweise bei der in der Systemprogrammierung immer noch recht gebräuchlichen Form des „Printk-Debuggings“[1] vorkommt, nimmt häufig bis zu einer halben Stunde in Anspruch. Einen Test mit SystemTap neu zu generieren und diesen vollautomatisch als Modul in den Kernel zu laden, dauert meist weniger als zwei Minuten.

Das Design von KSplice ist so angelegt, dass nur zwei Eingaben benötigt werden: die Kernel Sourcen und die in Form eines Patches gespeicherten Änderungen. Die maßgeschneiderten Quellcodes können auf diese Weise schnell eingespielt, aber auch archiviert werden. Ein weiterer Vorteil von KSplice liegt im Patchen bereits gepatchter Kernel. Für KSplice reichen die aktuellen, gepatchten Kernel-Sourcen und ein Patch mit den Änderungen aus, um erfolgreich einen neuen Patch in das Betriebssystem einzuhängen. Durch die hohe Wiederverwendbarkeit kann mit der Zeit eine Art Toolbox mit fertigen Patches aufgebaut werden, die ermöglichen, eine virtuelle Maschine schnell und wirtschaftlich maßzuschneidern.

So war es möglich mehrere Patches gleichzeitig benutzen und die Gesamt-Performance messen zu können. KSplice fügt Patches dynamisch in den Kernel ein. Der Overhead des Kernelinstallierens und Rechnerneustartens wird somit eliminiert.

Ein weiteres denkbares Tool, das in den Workflow eingebunden werden könnte, ist das in [MWP+01] erwähnte Tempo. Dieses Werkzeug unterstützt sowohl Optimierungen zur Übersetzungszeit, wie der GCC, als auch Optimierungen zur Laufzeit. Hierzu generiert Tempo zur Übersetzungszeit einen Laufzeitspezialisierer und erzeugt Templates des Objekt-Codes mit passenden „Löchern“ für statische Werte. Der Spezialisierer, der zur Laufzeit läuft, berechnet, welche Werte statisch sind und wählt dann das passende Template aus, um dieses mit den statischen Werten zu füllen.

Die Gesamtperformance wird im Host mit Hilfe von Benchmarks gemessen. Diese Benchmarks sind abhängig von der untersuchten Anwendung zu wählen. Im Falle von Lighttpd, einem als schlanke und performante Variante umworbenen Webservers, sind diese Benchmarks der bekannte Benchmark Httperf und ein eigens kreierter Benchmark mit dem Wget-Befehl.

3.1.1 Lighttpd

Lighttpd ist ein kleiner und schlank konzipierter Webserver, der sich auch in Kreisen von Eingebetteten Systemen einen Namen gemacht hat.

[1] An passenden Stellen `printk()`-Anweisungen platzieren, Kernel kompilieren, installieren und anschließend den Rechner neustarten. Für jede Änderung am Testszenario ist derselbe Workflow erneut durchzuführen.

Im aktuellen Developmentzweig der Entwicklung, Lighttpd 1.5, wird viel getan, um die effizientesten Systemaufrufe zu nutzen. Entgegen dem Ansatz, das Betriebssystem maßzuschneidern, kann man bei Lighttpd bereits in der Konfiguration aussuchen, welche Systemaufrufe, zum Beispiel für das Netzwerk oder die Input-Output-Operationen benutzt werden. Bei Lighttpd 1.5.x sollen neben der bereits bestehenden Auswahl an Systemaufrufen, auch noch eine Auswahl asynchroner I/O Operationen hinzugefügt worden sein.

Jedoch berichten verschiedene Anwender in Foren über ungewöhnliche Effekte. So kann es nach einiger Zeit zu einem sehr langen Blocken (ca. 10 Sekunden) kommen. Eine kompilierende und funktionstüchtige Version aus dem Developmentzweig konnte nicht gefunden werden.

Diese Diplomarbeit untersucht daher Lighttpd 1.4.19 aus dem stabilen Zweig (*Stable Branch*). An der Konfigurierung von Lighttpd wurden nur wenig Änderungen vorgenommen, da diese Diplomarbeit die Auswirkung der Maßschneiderung und nicht umgekehrt die Auswirkungen der Anpassung an einen Kernel untersucht.

3.1.2 Lastszenarien

Will man die Anwendung testen, so sind anwendungsspezifische Tests und Benchmarks heranzuziehen. Mosberger und Jin argumentieren, dass das interessante Verhalten für einen Webserver das Verhalten in Überlastsituationen ist. Um solche Überlastsituationen zu erzeugen, sei ein flexibles System[2] nötig, welches unterschiedlichste Überlastverfahren darstellen könne. [MJ98]

Da Überlastsituationen das normale Nutzerverhalten nicht widerspiegeln können, wurde noch ein zweites Testverfahren, die Simulation des User-Verhaltens mit Hilfe des wget Befehls, durchgeführt.

Natürlich müssten bei anderen Anwendungen entsprechend andere Benchmarks und Testverfahren gewählt werden.

3.1.2.1 Peak-Performance mittels httperf und autobench

Eine unter Administratoren und in der Wirtschaft gerne benutzte Größe ist die Peak-Performance. Man möchte wissen, wieviel Leistung eine Anwendung erbringen kann. Zu diesem Zweck hat Hewlett Packard (HP) Httperf entwickelt, ein Werkzeug, um Performance von Webservern messen zu können. HP wollte jedoch keinen neuen Benchmark entwerfen, sondern ein Werkzeug schaffen, mit dessen

[2]Hiermit ist Httperf gemeint

Hilfe, schnell Testszenarien erstellt und getestet werden können. Das Hauptaugenmerk bei der Entwicklung lag daher auf einer guten und vorhersagbaren Performance und einer leichten Erweiterbarkeit von Httperf [MJ98]. Httperf stellt ein Framework dar, um verschiedene HTTP-Stresstests gegen Server zu fahren und ihre Performance zu messen.

Für die Überlastungstests ruft Autobench Httperf mit wechselnden Parametern auf, um sukzessive die Last zu erhöhen. Diese Tests werden alle mit dem Aufruf derselben Webseite gemacht. Ein typischer Fall wäre also, die Performance der Haupt-Index-Seite zu testen, da diese als Einstieg zur Webseite sicherlich überdurchschnittlich häufig aufgerufen wird. Autobench liefert ein Script namens bench2graph mit, das zur Umwandlung der auf der Textkonsole ausgegebenen Daten in einen Graphen dient. Der am Ende dieses Kapitels gezeigte Graph wurde mit Hilfe von bench2graph und gnuplot erzeugt.

3.1.2.2 Standard Userverhalten mittels wget

Da das Verhalten unter Vollast jedoch nicht das typische Verhalten widerspiegelt, das bei dem Besuch einer Webseite auftritt, wurde noch ein zweites Anwendungsszenario getestet. Dieses Verhalten wird durch einen wget-Befehl simuliert. Bei der Simulation des Surfverhaltens werden folgende Punkte berücksichtigt:

- viele verschiedene Seitenaufrufe,
- Dateien verschiedener Größen (Index-Dateien mit wenigen Kilobytes, bis hin zu PDF-Dateien mit mehreren Megabytes),
- randomisierte Wartezeiten zwischen den Seitenabrufen.

Wget verfügt als beliebter Downloadmanager über eine recht beachtliche Anzahl von Optionen. Einige eignen sich sogar zur Nachahmung eines menschlichen User-Verhaltens.

3.2 Statische Analyse

Viele Einsparpotenziale sind direkt durch eine statische Analyse des Codes und des Studiums des vorliegenden Betriebssystemdesigns erkennbar.

3.2.1 Virtuelles File System (VFS)

Bei lighttpd handelt es sich um eine Input-Output-lastige (*I/O-Bound*) Anwendung. Da in der virtuellen Maschine das Dateisystem bekannt ist, kann man überlegen, die Indirektion des Dateisystems zu entfernen. Dies ist dann möglich, wenn

sämtliche Dateizugriffe nur noch auf ein Dateisystem erfolgen. Zu berücksichtigen ist, dass Dateisysteme wie procfs oder tmpfs weiterhin benötigt werden. Deshalb ist ein vollständiges Entfernen des VFS nicht möglich. Es bleibt die Möglichkeit einen Bypass zu legen.

Das VFS ist eine Abstraktionsschicht, die sich zwischen der Benutzer-Anwendung (*application program*) und der Implementierung des Datei-Systems befindet. Durch die Einführung dieser Abstraktionsschicht ist es möglich, dass das Anwendungsprogramm kein Wissen über die zugrunde liegende Implementierung des Dateisystems benötigt. [BC06]

Die Abstraktionsschicht besteht aus dem „Common File Model“. Dieses Modell spiegelt die Eigenschaften von Unix-Dateisystemen wider. Alle Operationen, die auf einem Unix-Dateisystem ausgeführt werden können, wie zum Beispiel Lesen (`read()`), asynchrones Lesen (`aio_read`), Schreiben (`write`), und vieles mehr sind Bestandteil des Common File Model.

Ein einfacher Befehl wie das Kopieren (`cp`) kann auf diese Weise mit den verschiedensten Dateisystemen umgehen. Denn das Programm braucht dem Betriebssystem nur abstrakt – über das Common File Model – mitzuteilen, dass es die nötigen Dateioperationen ausführen will, zum Beispiel `read()`. Die entsprechende Implementierung des zugrunde liegenden Dateisystems wird dann mit einer Indirektion[3] aufgerufen.

3.2.2 Scheduler des Block I/O Layers

„Block-Devices sind Hardware-Devices, die den sogenannten *Random Access* (wörtlich: zufälliger Zugriff, soll heißen: nicht notwendigerweise sequentieller Zugriff) auf »Datenstücke« fester Größe erlauben, die Blocks genannt werden.“ [LK05]

Block-Devices sind sowohl Festplatten, als auch CDROM- und Disketten-Laufwerke, sowie Flash-Speicher. Im Gegensatz hierzu werden Character-Devices, wie Tastaturen oder Ports definiert. Der wesentlichste Unterschied zwischen beiden ist der, dass Block-Devices den direkten Zugriff auf eine Position (seek-Operation) durchführen können. Charakter-Devices hingegen kennen keine Position[4], so dass ein Zugriff auf zehn Positionen vorher nicht möglich ist. Dateisysteme können auf Block-Devices gemountet werden. Da Dateisysteme jedoch sehr komplex sind und der Kernel „*mehr Sorgfalt, Vorbereitung und Aufwand*“ [LK05] in sie stecken muss, gibt es ein Subsystem, den Block-I/O-Layer, der sich um die Verwaltung der Disk-Operationen auf Blockebene kümmert.

[3] Pointer auf die Funktion, die die Dateisystemoperation implementiert.
[4] Eigentlich kennen sie exakt eine, die aktuelle Position.

Ein durchaus performancekritischer Teil des Block-I/O-Layers ist die Strategie zur Abarbeitung von anstehenden Block-I/O-Requests. I/O-Anforderungen (requests) werden in doppelt verlinkten Listen (queues) verwaltet. Würden I/O requests einfach hintereinander ausgeführt, so wäre das Resultat eine schlechte Performance. Der Lesekopf der Festplatte müsste in ungünstigsten Fällen jedes Mal den passenden Sektor suchen, bevor dieser gelesen werden kann. Suchen nach Sektoren auf der Festplatte (*Disk-Seek*) ist eine recht langsame Operation[5]. Um I/O-Anforderungen effektiv zu verarbeiten, gibt es ein Subsystem, den I/O-Scheduler, der durch das Zusammenfassen (*Merging*) und das Sortieren (*Sorting*) von I/O-Anforderungen die Performance des Gesamtsystems zu steigern versucht.

Da Lighttpd, wie in Abschnitt 3.2.1 erwähnt, stark I/O lastig ist und zusätzlich nochmals eine starke Präferenz für Leseanforderungen[6] hat, sollte die Scheduler-Strategie, die zur Abarbeitung der I/O-Anforderungen benutzt wird, überdacht werden. Standardmäßig ist der „Complete Fairness Queueing“ Scheduling-Algorithmus aktiv. Dieser versucht durch eine hohe Anzahl von Queues[7] die Bandbreite der Festplatte fair auf alle laufenden Prozesse aufzuteilen. Hierfür wird per Hashfunktion aus dem Thread-Group-Identifier ein Hash erzeugt, der als Index für die Queue des Prozesses dient. Jede neue I/O Anforderung wird am Ende der für sie zuständige Prozess-Queue eingefügt. Innerhalb einer Queue werden die I/O Anforderungen nach Lokalität auf der Festplatte sortiert und gruppiert. Die Queues werden im RoundRobin-Verfahren durchsucht und die erste nicht leere Queue gewählt. Dann wird ein „Schwung“ der anstehenden Requests in die sogenannte Dispatcher-Queue eingereiht. Requests in der Dispatcher-Queue werden anschließend abgearbeitet.

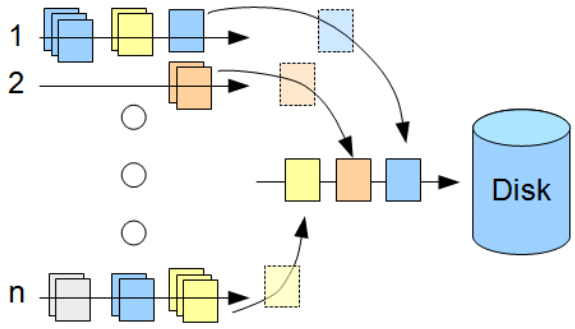

Abbildung 3.1: Complete Fairness Queueing

Durch die hohe Anzahl von Lese-Anforderungen (*read request*), eignet sich der „Deadline“-Scheduling-Algorithmus eventuell besser. Dieses Scheduling-Verfahren ist einfacher und verwaltet abgesehen von der Dispatcher-Queue insgesamt nur

[5]Ein Seek benötigt mehrere Millisekunden.
[6]Hervorgerufen durch das Lesen der Webseiten (`read()`-Systemaufruf).
[7]Der Standardwert beträgt 64 Queues.

vier Queues: zwei sortierte Queues (read und write Queue, sortiert nach Sektornummern) und zwei Deadline Queues (ebenfalls read und write Queue, mit denselben Requests, diesmal nach Deadline sortiert).[BC06]

Definition 3.1 (Starvation)
„A process that is ready to run but waiting for the CPU can be considered blocked. A priority scheduling algorithm can leave some low priority processes waiting indefinitely.“[SGG05]

Verhungern (*Starvation*) tritt beim Block-Layer-Scheduling auf, wenn eine I/O Anforderung ewig auf eine Ressource, in diesem Fall die Dispatcher-Queue, warten muss, dieser aber nicht zugeteilt wird, weil der Scheduling-Algorithmus die Dispatcher-Queue mit anderen bevorzugteren I/O Anforderungen füllt.

Dieses Scheduling Verhalten kann entstehen, sobald eine Gruppe von I/O Anforderungen, der anderen vorgezogen wird, wie es die Strategie zum Minimieren der Kopf-Bewegung des Festplattenarms vorsieht.

Der Scheduling-Algorithmus bevorzugt dann längere Zeit andere I/O-Anforderungen, da diese (sektorenmäßig) näher an der zuletzt abgesetzten I/O Anforderung sind. Durch die Bevorzugung von I/O-Anforderungen, die physikalisch nahe beieinander liegen, erhofft man sich eine geringe Anzahl teurer Festplatten-Suchen (*Disk-Seek*). Die nicht bevorzugten I/O-Anforderungen werden jedoch nicht ausgewählt und an den Dispatcher weitergegeben. Bei Schreibanforderungen mag dieses Verhalten eventuell akzeptabel sein, da die Anwendung, die eine Schreibanforderung absetzt, normalerweise nicht blockiert, sondern fortfährt. Bei Leseanforderungen wartet der Prozess jedoch auf die zu verarbeitenden Daten und blockiert.

In dem Deadline-Scheduling-Verfahren bekommt jede I/O-Anforderung eine Verfallszeit. Lese-Anforderungen (*read requests*) verfallen nach 500 ms, Schreib-Anforderungen (*write requests*) nach 5000 ms. Wie schon bei anderen Verfahren[8] werden die eingehenden I/O Anforderungen der Reihe nach in eine First-In / First-Out (FIFO) Queue sortiert und bei physikalischer Lokalität zusammengefasst (*Merging*). Wie in Abbildung 3.2 zu sehen ist, wird die neue Lese-Anforderung (grün) mit der Nummer 1 sowohl in die sortierte Queue, als auch in die Deadline-Queue für Leseanforderungen am Ende eingereiht. Schreibanforderungen (rot) werden dementsprechend in die sortierte Liste und die Deadline-Queue für Schreibanforderungen eingereiht.

Die Deadline-Queues wurden eingeführt, um ein Verhungern (*Starvation*) der I/O-Anforderungen weitestgehend zu verhindern und gleichzeitig einen hohen

[8] z.B. dem alten Linus-Elevator (siehe hierzu [LK05])

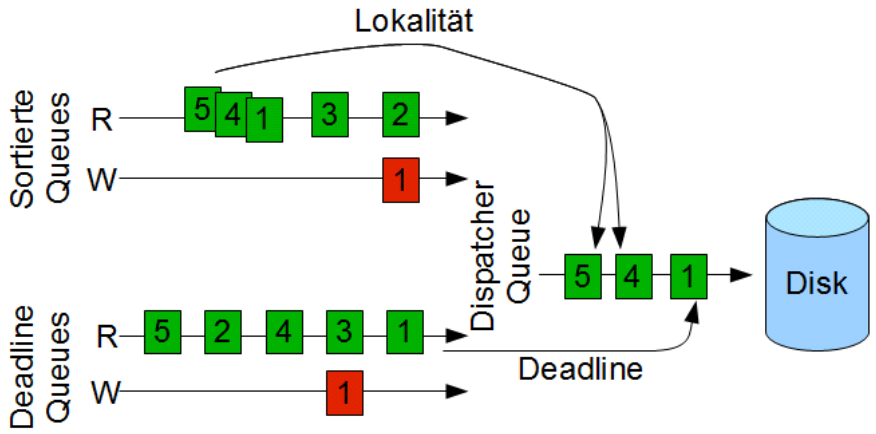

Abbildung 3.2: Deadline-Scheduling

Durchsatz zu erreichen. Sollte die Dispatcher-Queue leer sein und aufgefüllt werden müssen, so wird geprüft, ob in den Deadline-Queues Anforderungen mit abgelaufener Deadline vorhanden sind. Diese werden sofort in die Dispatcher-Queue eingereiht. Danach wird ein „Schwung“ von Anforderungen aus der sortierten Liste übernommen, beginnend mit der Anforderung, die derjenigen, die ihre Deadline überschritten hat, folgt. Die Anzahl der in diesem „Schwung“ übernommenen Anforderungen variiert je nach Lokalität der Anforderungen auf der Festplatte.

In diesem Fall hat Nummer 1 seine Deadline verpaßt und wird sofort in die Dispatcher-Queue eingereiht. Aufgrund der Sektoren-Ähnlichkeit mit Nummer 4 und Nummer 5, wird dieser „Schwung“ an I/O Anforderungen mit in die Dispatecher Queue übernommen.

Ist keine Anforderung über ihre Deadline, so wird von dem Scheduler ein „Schwung“ von I/O-Anforderungen aus der sortierten Liste genommen, beginnend bei der ersten Anforderung, die der zuletzt ausgeführten folgt.

3.2.3 Kernel Mode / User Mode Grenze (Adressraumtrennung)

Um eine korrekte Ausführung des Betriebssystems zu gewährleisten, muss man in der Lage sein, zwischen der Ausführung von Betriebssystemcode und Anwendungscode zu unterscheiden. Der heute gebräuchlichste Ansatz ist der, dass die CPU Hardwareunterstützung zur Unterscheidung des Ausführungsmodus bereitstellt. [SGG05] Die x86 Architektur unterstützt Betriebssysteme, indem sie seit dem 80286 den in Abschnitt 2.1.1.2 vorgestellten Protected Mode zur Verfügung stellt. Die Virtualisierung wird durch Hardware-Erweiterungen wie etwa Intels

VT und AMDs Pacifica weiter unterstützt (Abschnitt 2.1.3.2.1). Moderne Betriebssysteme wie Linux und Windows 2000, XP, etc. kennen zwei verschiedene Ausführungsmodi. Den User-Mode (Ring 3) und den Kernel-Mode (Ring 0). Um den Zustand des Rechners zu bestimmen, wird ein spezielles Bit, das *Mode Bit*, in die Computerhardware eingebaut. Auf diese Weise kann unterschieden werden, ob ein Task für eine Benutzer-Anwendung oder für den Kernel ausgeführt wird.

Jede Benutzer-Anwendung (*user application*) wird früher oder später einmal das Betriebssystem beauftragen, Aufgaben auszuführen, wie etwa Dateien zu öffnen oder zu speichern.

Privilegierte Instruktionen, wie etwa Kontrolle von Input-Output-Operationen oder das Interrupt-Management, können von Benutzer-Anwendungen nicht direkt, sondern nur über Systemaufrufe ausgeführt werden.

Wird ein Systemaufruf aufgerufen, wird er von der Hardware wie ein Software-Interrupt behandelt und die Kontrolle durch den Interruptvektor an eine Service Routine im Betriebssystem weitergegeben. Das Mode-Bit wird auf 0 (*Kernel-Modus*) gesetzt und der Kernel untersucht die Instruktion und die übergebenen Parameter, um herauszufinden, welcher Syscall aufgetreten ist und welche Art von Diensten (*Services*) die Benutzer-Anwendung erfordert. Nachdem der Kernel die Parameterwerte auf ihre Korrektheit und Gültigkeit überprüft hat, wird der Syscall ausgeführt und am Ende die Kontrolle an die Instruktion übergeben, die dem Syscall folgt.

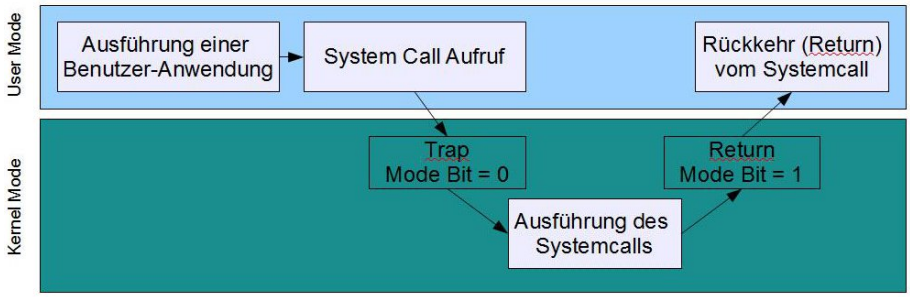

Abbildung 3.3: Übergänge zwischen User- und Kernel-Mode

Wie in Abbildung 3.3 zu sehen ist, tritt bei jedem Systemaufruf ein Moduswechsel auf.

Desweiteren müssen sämtliche Daten in Registern gespeichert werden. Alle Register pro Eintritt in den Kernel und Austritt aus dem Kernel zu speichern, ist recht kostspielig. In der älteren Variante mit der int $0x80 Instruktion werden zusätzlich noch sehr viele Abfragen gemacht. So wird zum Beispiel mit TIF_SYSCALL_TRACE und TIF_SYSCALL_AUDIT überprüft, ob das Programm gerade durch einen Debugger ausgeführt wird. [BC06]

Der Moduswechsel an sich kann recht teuer sein, da zwischen dem Kernel-Modus und dem User-Modus der Adressraum sich ändert und auszutauschende Daten kopiert werden müssen und nicht einfach nur referenziert werden können[9].

Wie später im Detail erläutert wird, ist die Anzahl der Systemaufrufe in den Anwendungsszenarien recht hoch, so dass eine Einsparung von Taktzyklen bei jedem Systemaufruf ein sehr hohes Einsparpotenzial birgt.

Kernel Mode Linux (KML) ist ein Projekt[10], um Anwendungen im Kernel-Mode laufen zu lassen und ihnen damit die Möglichkeit zu geben, ohne Kosten für das Überschreiten der User-Kernel-Grenze und damit für den Adressraumwechsel zu bezahlen. Der Kopieraufwand der zwischen Kernel- und User-Mode auszutauschenden Daten würde ebenso entfallen. [Mae03]

3.2.4 Virtualisierung

Bis eine Datei auf der physikalischen Festplatte gespeichert werden kann, ist es ein langer Weg. Abbildung 3.4 zeigt diesen Weg[11].

Die Anwendung, die eine Datei speichern will, übergibt ihre Daten an einen Systemaufruf zum Speichern im Gast-Betriebssystem. Dem Treiber, der die Daten jetzt auf der Hardware speichern will, wird eine täuschend echte IDE-Schnittstelle vorgespiegelt. Normalerweise würde die IDE-Schnittstelle die Festplattenelektronik zum Speichern der Daten auf der physischen Festplatte veranlassen.

Nicht so die IDE-Schnittstelle der virtuellen Hardware. Sie reicht die Daten nach unten weiter durch und bestätigt dem Gast-Betriebssystem, dass alle Operationen ausgeführt wurden. Für das Gast-Betriebssystem und die Anwendung ist die Aufgabe damit erledigt, dennoch wurde bis jetzt kein Byte übertragen.

Nun ist der Virtualisierungslayer an der Reihe und muss sich darum kümmern, wo die Daten gespeichert werden. Die schon als geschrieben bestätigten Sektoren des Gast-Betriebssystems, die die zu speichernde Datei darstellen, müssen in die virtuelle Festplatte auf dem Host geschrieben werden. Die virtuelle Festplatte ihrerseits ist jedoch wieder nur eine Datei im Host-System[12]. Um die Daten endgültig in der virtuellen Behälterdatei zu speichern, übergibt der Virtualisierungslayer der Treiberroutine im Host-Betriebssystem die zu speichernden Daten.

Der Treiber des Host-Betriebssystems kommuniziert nun endlich mit der echten Schnittstelle, die jetzt die Speicherung der Daten des Gast-Betriebssystems in der

[9] Hierfür gibt es die Systemaufrufe `get_user()` und `put_user()`

[10] Entstanden aus der Arbeit: „Kernel Mode Linux: Toward an Operating System Protected by a Type Theory“[MY03]

[11] vgl. [Ahn09]

[12] VMWare unterstützt auch die Speicherung von Daten per Netzwerk. Die Behälterdatei kann somit auch im Netzwerk liegen, was die Latenz noch weiter erhöhen würde.

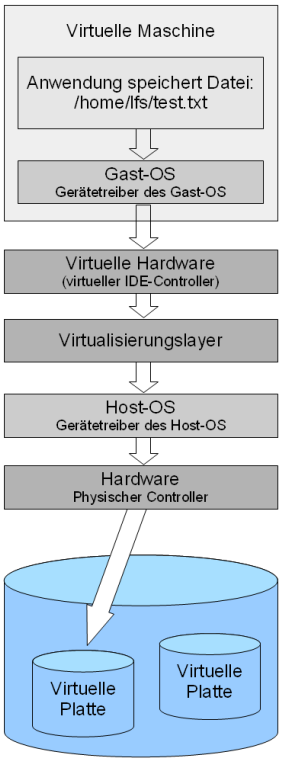

Abbildung 3.4: Die verschiedenen beteiligten Schichten beim Speichern einer Datei (Typ 2 Hypervisor)

virtuellen Behälterdatei auf dem Host veranlasst.[Ahn09]

Dieser lange und komplizierte Weg, legt nahe, dass je nach eingesetzter Virtualisierungsstrategie wesentlich kürzere Ausführungszeiten im Gast-Betriebssystem möglich sind. Zwei Ansatzpunkte, die die Performance stark beeinflussen können, sind die Paravirtualisierung und der Austausch des virtuellen Hardwarelayers.

3.2.4.1 Paravirtualisierung

Auch wenn der Weg, den eine zu speichernde Datei zu nehmen hat, recht weit ist, stellt man sich berechtigterweise die Frage, ob Paravirtualisierung noch einen signifikanten Teil zur Leistungssteigerung leisten kann. Das VMWare WhitePaper „Performance des VMI“ [VMW08] vergleicht die Leistung der reinen Binär-Translation eines VMWare ESX Servers mit der Leistung eines ESX Servers sowohl mit Binär-Translation, als auch aktiviertem Virtual Machine Interface. Der VMWare ESX Server, der einen Hypervisor vom Typ 1 darstellt, profitiert trotz

seiner architekturbedingten guten Performance von den paravirtualisierten Treibern des Virtual Machine Interfaces (VMI). [VMW08]

In den interessanten Benchmarks des VMWare WhitePaper „Performance des VMI“ (den Messungen mit einem Workload) zeigt sich, dass die Performance teilweise um Faktor 2-3 besser war als ohne VMI[13]. In dem White Paper wird deutlich, dass viele verschiedene Workloads von den paravirtualisierten Treibern profitieren. Zwei dieser Workloads sind für die Anwendung lighttpd interessant: I/O-Performance und schnellere Systemaufrufe (schnelle sysenter und sysexit Funktionen). Lighttpd sollte von daher gerade während eines Stresstests mit Autobench gute Ergebnisse zeigen.

3.3 Dynamische Analyse

Durch eine dynamische Analyse mit Hilfe eines Profilers wie SystemTap, können Probleme die zur Laufzeit auftreten, erkannt werden. Als Ausgangslage für die Messungen dient ein selbstgeschriebenes Systemtap-Skript, welches pro Systemaufruf folgende Daten sammelt: CPU-Zyklus beim Betreten der Funktion, User-Anwendung, Systemaufruf, übergebene Parameter, Rückgabewert und Ausführungszeit gemessen in CPU-Zyklen. Am Ende weist das Systemtap-Skript noch Minimal-, Maximal- und Durchschnittswerte der einzelnen Messungen aus. Diese Daten werden in einer großen Datei gesammelt und nach der Messung, um diese nicht zu stören, ausgewertet. Ein etwas komplizierteres Shell-Skript erzeugt aus den Daten dann viele kleine Dateien. Zum Beispiel beinhaltet eine Datei namens `Messung.lighttpd.syscall_counts` nur die von lighttpd abgesetzten Systemaufrufe, eine andere, deren Name `Messung.syscalls` lautet, listet alle Systemaufrufe auf.

Obwohl das Testsystem schon auf grafische Oberflächen verzichtet und selbst nur eine konsolenbasierte Standardinstallation eines Debian Lenny Systems ist, fällt auf, dass noch viele Prozesse, die nicht für den Betrieb des Webservers benötigt werden, vorhanden sind und ebenfalls Prozessorzeit beanspruchen, die dann für die Ausführung des Webservers fehlt. Zu diesen Prozessen gehören der exim4 genauso wie zum Beispiel der dhclient, der durch eine statische Konfiguration komplett überflüssig wird. Ebenso können vorhandene Logging-Dienste deinstalliert und die Aufgaben des cron-Dienstes (*Log-Rotation* für Betriebssystem-Logdateien) gelöscht werden.

Der Linux-Kernel unterstützt, in der benutzten Version 2.6.28.10, 332 Systemaufrufe. Während eines Testlaufs mit dem unoptimierten Distributions-Kernel und der Standardinstallation fallen insgesamt 70 Systemaufrufe auf. Werden die

[13]Siehe: Kernel-Microbenchmarks und I/O-Benchmarks [VMW08]

nicht benötigten Programme entfernt, die hauptsächlich Logging-Aufgaben erfüllen oder wie der DHCP-Client Netzwerkverbindung aufbauen, um Verbindungstests durchzuführen, so bleiben nur noch zwölf Systemaufrufe übrig, die der Lighttpd direkt verwendet.

Diese Systemaufrufe sind `close()`, `fcntl()`, `ioctl()`, `open()`, `poll()`, `read()`, `select()`, `setsockopt()`, `shutdown()`, `stat()`, `time()`, `write()` und `writev()`.

Sieht man sich die verbleibenden Systemaufrufe an, fällt auf, dass sich diese hauptsächlich in zwei Gruppen einteilen lassen: I/O Operationen und Netzwerk-Operationen.

3.4 Maßschneiderungsmöglichkeiten

Die statische und die dynamische Analyse ergeben eine starke Abhängigkeit der Anwendung von I/O Operationen und Netzwerkoperationen. Tabelle 3.1 zeigt einen Überblick über die verschiedenen Ansatzpunkte auf den verschiedenen Ebenen. Abbildung 3.5 zeigt den nichtoptimierten Webserver lighttpd. Dabei fällt auf, dass bei nur 500 bis 600 Anfragen pro Sekunde, die Performance stark einbricht und der Netzwerktraffic (Linie net_io_172.16.68.128) signifikant zurückgeht. Die Anzahl der Fehler (errors_172.16.68.128) erhöht sich. Fehler im Sinne von Autobench sind auch die auftretenden „client-timo“s, die Client-Timeouts. Dies bedeutet, dass der Webserver innerhalb von 4 Sekunden keine Antwort auf die Anfrage geschickt hat. Insgesamt zeigt der Webserver eine unbefriedigende und sehr wechselhafte Performance.

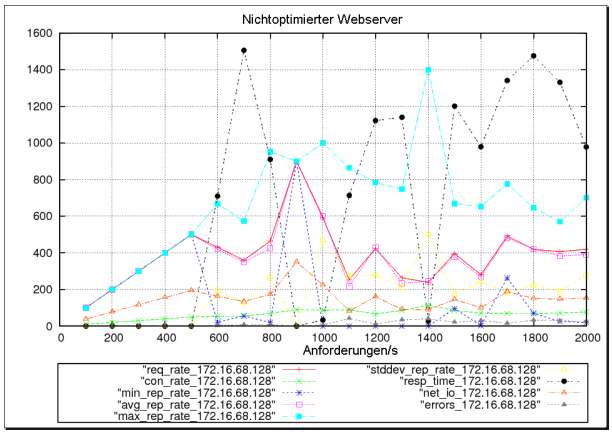

Abbildung 3.5: Performance des lighttpd in der virtuellen Maschine

Ebene	**Optimierungsansatz**	
	Manuell	**Automatisiert**
Paketkonfiguration		debfoster, deporphan
Kernelkonfiguration	Minimierung der Kernelkonfiguration	
Code	VFS Indirektion Bypass Rechteverwaltung entfernen LSM[14] entfernen Entfernen von Überprüfungen	User – Kernel-Mode-Grenze GCC Code Optimierungen
Virtuelle Maschine		Paravirtualisierung paravirtualisierter Treiber

Tabelle 3.1: Möglichkeiten der Optimierung auf den einzelnen Ebenen

4 Implementierung

Manuelle Maßschneiderungen am Linux-Kernel vorzunehmen und dabei signifikante Performance-Vorteile zu erzielen, ist kein leichtes Unterfangen.

Auf der einen Seite ist der Linux-Kernel bereits sehr häufig im Fokus wissenschaftlicher Arbeiten gewesen. Hierdurch ist der Linux-Kernel stets verbessert und im Hinblick auf Performance erweitert worden. Die Anzahl der allgemeinen Ansatzpunkte (Verwendung von Caches, Buffer, etc.) zur weiteren Optimierung sind somit recht gering.

Auf der anderen Seite ist es, wie später am Beispiel des `open()`-Systemaufrufs im Abschnitt 4.2.1 gezeigt wird, nicht immer leicht, Einsparungspotenziale in Performance-Gewinne um zu setzen, da die Aufrufgraphen eine große Tiefe (ext3 Aufrufe müssen stets auch das Journaling berücksichtigen) aufweisen können. Das Einsparen von ein oder zwei Überprüfungen auf dem Weg zum Öffnen der Datei, aufgrund der bekannten Umgebung, kann daher nicht allein zum Erfolg führen.

Vielmehr ist ein Zusammenspiel der einzelnen Faktoren nötig. Abschitt 4.1 beschreibt die vorgenommenen strukturellen Änderungen, die aufgrund der Analyse des Betriebssystemdesigns entstanden sind. Hierzu gehören die Maßschneiderung des „Virtuellen File Systems“ (*VFS*) im Abschnitt 3.2.1, sowie das Entfernen des Rechtemanagements im Abschnitt 4.1.2. Abschließend gibt der Abschnitt 4.1.4 einen Überblick über die Entfernung von Überprüfungen, die aufgrund der Anpassung an die Umgebung in der virtuellen Maschine aus den untersuchten Funktionen entfernt werden konnten. Im Abschnitt „Bordmittel“ 4.2 werden Optimierungsansätze, sowohl für die Code-Ebene, als auch für die Virtualisierungs-Ebene vorgestellt.

Im Abschnitt 4.3 wird eine Technik von Maeda [Mae03] vorgestellt, mit der die Grenze zwischen User-Mode und Kernel-Mode aufgehoben werden kann, indem es ermöglicht wird, Anwendungen im Betriebssystemkern laufen zu lassen.

4.1 Manuelle Maßschneiderung

Dieser Abschnitt stellt die manuellen Maßschneiderungen, die anhand der Analyse identifiziert wurden, vor. Abschnitt 4.1.1 erläutert Implementierungsdetails

zum Bypass des „Virtuellen File Systems“. Im Abschnitt 4.1.2 wird als manuelle Maßschneiderung die Verwaltung der Zugriffsrechte (DAC[1]) entfernt. Bei dem Sicherheitsmodell der DAC ist die Authorisation an Identität, Gruppenzugehörigkeit und Zugriffsattribute der Objekte gebunden.

Diese Art der Zugriffskontrolle hat teilweise erhebliche Schwachstellen:

- Grobe Kontrolle durch Prozesse - Die Berechtigungen werden durch Prozesse, die `fork()` oder `exec()` aufrufen, uneingeschränkt weitergegeben. So haben viele Prozesse mehr Zugriffsrechte als sie zur Ausführung benötigen.
- Identität - Mit Hilfe von setuid/setgid können Berechtigungen, die an Identitäten gebunden sind, leicht geändert werden.
- Faktor Mensch - Benutzer können die Berechtigungen selber setzen und anderen den Zugriff auf ihre Daten ermöglichen.

Daher gibt es seit längerem viele Projekte, die sich mit der Erhöhung der Sicherheit im Linux-Kernel beschäftigen.

Linux Security Modules (LSM) ist eines dieser Projekte und wurde in den Mainstream-Kernel aufgenommen. Es bietet eine erhöhte Sicherheit beim Zugriff auf interne Kernel Datenstrukturen. Die Entfernung von Linux Security Modules ist im Abschnitt 4.1.3 aufgeführt.

Bei der Anpassung des Systems an die virtuelle Maschine sind auch weitere Passagen mit Überprüfungen aufgefallen, die von der virtuellen Maschine und den durch die Konfiguration des Betriebssystems gegebenen Rahmenbedingungen überflüssig geworden sind. Ihre Entfernung wird im Abschnitt 4.1.4 erläutert.

4.1.1 Virtuelles Filesystem

Das „Virtuelle File System“ bietet den Anwendungsprogrammen eine genormte Schnittstelle, auf die sie zugreifen können, wenn sie mit einem bestimmten Datei-System, wie zum Beispiel ext3, kommunizieren wollen. Wie im Abschnitt (3.2.1) erwähnt, stellt das „Virtuelle File System“ eine Zwischenschicht dar, die Aufrufe, wie etwa das Öffnen von Dateien, an das entsprechende Dateisystem weiterleitet.

Eine vollständige Entfernung dieser Zwischenschicht ist leider nicht möglich, da andere Dateisysteme, wie etwa procfs oder tmpfs noch über die Schnittstellen kommunizieren. Daher wurde ein Bypass zum verwendeten ext3-Dateisystem gelegt.

Zusätzliche Maßschneiderungspotenziale ergeben sich vor allem bei dem Systemaufruf `writev()`. Sieht man sich die Aufrufgraphen zu den verwandten Systemaufrufen `readv()` und `writev()` an, so stellt man fest, dass ab einer Tiefe

[1]DAC = discretionary access control

von zwei Aufrufen, die gemeinsame Funktion `do_readv_writev()` benutzt wird. Dieser Funktion wird, unter anderem, als ein Parameter entweder „WRITE“ oder „READ“ übergeben, und so wird das Verhalten der Funktion dynamisch bestimmt.

Insgesamt können einige Überprüfungen auf dem Weg in das unter dem VFS liegende Dateisystem beim Erstellen des Bypass zusammengelegt und auf einige, wie beispielsweise die erwähnten Verhaltens-Überprüfungen bei `writev()` ganz verzichtet werden.

4.1.2 Verwaltung der Zugriffsrechte

Die Verwaltung der Zugriffsrechte ist ein zentraler Bestandteil moderner Betriebssysteme.

Im Gegensatz zu anderen Funktionen[2], die in der Kernel-Konfiguration durch Setzen verschiedener Optionen, ein- und ausgeschaltet werden können, geht dies bei der Verwaltung der Zugriffsrechte nicht.

Zum einen ist die Verwaltung der Zugriffsrechte — als zentraler Bestandteil des Betriebssystems — nicht dafür vorgesehen, konfiguriert zu werden. Zum anderen wäre dies nicht ganz so leicht möglich, da die Rechteverwaltung an verschiedenen Stellen Funktionen im Linux-Kernel betrifft.

Ein guter Anfang für die Maßschneiderung der Zugriffsrechte im Linux-Kernel sind die Fehlercodes (*error code*), die im Falle eines Zugriffsfehlers entstehen würden.

EACCES ist der Fehlercode, der für die Verletzung der Zugriffsrechte zuständig ist. Fehlercodes sind als Makro-Konstanten definiert, die positive Integer-Werte größer „0“ besitzen. Eine vollständige Liste der Fehlercodes befindet sich in entsprechenden Header-Datei der jeweiligen Architektur[3].

Das Verfahren Maßschneiderungsstellen auf diese Art zu finden, ist nicht sehr genau. Viele Systemaufrufe verzögern die Entscheidung, in der die Zugriffsrechte an weitere Aufrufe durchgereicht werden oder sie delegieren diese Aufgabe ganz.

Ein Beispiel hierfür ist die Maßschneiderung des Systemaufrufs `open()`. Der Systemaufruf `sys_open()` erhält drei Parameter, den Pfadnamen der zu öffnenden Datei, die „access mode flags“ und eine „permission bit mask“, die die Zugriffsrechte regelt. Der Aufrufgraph verzweigt sich immer weiter. Die interessante Stelle ist hier die Verzweigung nach `filp_open()`. `Filp_open()` ist jedoch nicht

[2] Es ist zum Beispiel möglich, die Unterstützung für Mehr-Prozessor-Systeme zu entfernen.

[3] Bei x86: include/asm-i386/errno.h.
Um eine Portabilität zwischen Unix-Systemen zu gewährleisten, wird diese Header-Datei auch in die Header-Datei der Standard C Library /usr/include/errno.h eingebunden [BC06]

mehr Teil des `open()`-Systemaufrufs, sondern eine von namei.c zur Verfügung gestellte Funktion, um einen Inode anhand seines Pfadnamens zu öffnen.

NAMEI ist der Mechanismus, der in Linux die Auflösung von Pfadnamen bereitstellt. Hierzu werden Pfadnamen solange durchsucht, bis ein Endpunkt erreicht wird und zu einem finalen Dentry-Objekt (*Directory Entry*)[4] aufgelöst werden kann.

Da die Überprüfung der Pfadnamen recht langwierig sein kann, werden einmal durch die Pfad-Auflösung gesuchte Dentry-Objekte im Dentry-Cache gespeichert.

Ein wichtiger Punkt, der bei der Auflösung der Pfadnamen zu beachten ist, ist die Berechtigung auf Dateien und Verzeichnisse zugreifen zu dürfen. Somit spielt bei der Auflösung die Rechteverwaltung (DAC) ebenfalls eine große Rolle. Die in der Rechteverwaltung benutzten Flags für den Zugriff, werden in NAMEI - Auflösungen umcodiert (siehe Tabelle 4.1). Das Bit mit der niedrigsten Priorität, Bit 0 drückt aus, dass eine Lese-Berechtigung erforderlich ist, während Bit 1, eine Schreib-Berechtigung voraussetzt.

Das Flag „00“ hat nun eine neue Bedeutung: Es werden keine Zugriffsrechte benötigt. Dies ist sinnvoll, um zum Beispiel beim Folgen symbolischer Links (*Symlinks*) die Zugriffsberechtigung zu einem späteren Zeitpunkt zu klären.

Flag Bit0 | Bit1	Bedeutung der Zugriffs-Flags für Systemaufrufe[5]	Bedeutung der NAMEI-Flags
0 | 0	Nur Schreiben	Keine Rechte erforderlich
0 | 1	Nur Lesen	Nur Schreiben
1 | 0	Schreiben/Lesen	Nur Lesen
1 | 1	Spezial	Schreiben/Lesen

Tabelle 4.1: Bedeutung der Zugriff-Flags für NAMEI-Funktionen

An dieser Stelle greift die Maßschneiderung ein und ermöglicht durch das Entfernen der Rechteverwaltung einem normalen Benutzer, zum Beispiel für ihn nicht zugängliche Ordner einzusehen oder diese sogar umbenennen oder löschen zu können.

Aufgrund dieser zum Teil tief verschachtelten Aufruf-Beziehungen im Linux-Kernel ist es nicht möglich, alle Maßschneiderungsstellen exakt durch eine Suche nach dem Auftreten der Fehlercodes (EACCES) zu bestimmen. Fällt eine Funktion die Entscheidung den Zugriff zu verweigern, zum Beispiel die Funktion

[4]Ein Dentry-Objekt repräsentiert eine bestimmte Komponente in einem Pfad. Der Pfad `/root/text.file` besteht aus drei Dentry-Objekten: `/`, `root` und `text.file`. Alle Komponenten eines Pfades, einschließlich der Dateien, sind Dentry-Objekte. In der Struktur des Dentry-Objekts ist ein Verweis auf den entsprechenden Inode (und somit widerum auf die auf die im Dateisystem gespeicherte Datei) gelistet.

`inode_permissions()`, so gibt diese den Wert EACCES zurück, welcher bis zum System-Aufruf hochgereicht wird. Gleichwohl gibt eine solche Suche sehr gute Hinweise auf sämtliche Maßschneiderungsstellen.

In Teilen dieser Maßschneiderung konnten KSplice-Module nicht immer benutzt werden. Versucht man die Funktion `write()` Maß zu schneidern, verweigert KSplice den Dienst, da es beim Erzeugen der neuen Binär-Datei selber diese Funktion benötigt. Daher kann diese Funktion nicht per KSplice in den Kernel eingebracht werden, sondern sie muss im Originalquellcode verändert werden.

4.1.3 Linux Security Modules

In der Linux-Gemeinde gab es lange Zeit eine große Diskussion darum, wie eine Erweiterung der bestehenden — als nicht sicher erachteten Discretionary Access Control (DAC) — aussehen soll. In einer E-Mail soll Linus Torvalds die Anforderungen an ein Sicherheits-Framework wie folgt formuliert haben[6]:

- wahrhaft generisch - Das Ändern eines Sicherheits-Models (*security model*) soll nicht mehr erfordern als das Laden eines Kernel-Moduls.
- einfaches Design - Dieses sollte sowohl konzeptionell einfach und schlank, sowie effizient sein.
- Posix 1.e Capabilities Unterstützung - Die vorhandenen Posix 1.e Capabilities sollen als optionales Sicherheits-Model unterstützt werden.

Ein Gleichgewicht zwischen dem Trade-Off, soviele funktionale Fähigkeiten von anderen Projekten wie möglich zu integrieren und der gleichzeitigen Minimierung der Auswirkungen auf Codeänderungen am Linux-Kernel, herzustellen, war ein großes Problem beim Entwerfen des Design der Linux Security Modules. Um die Auswirkungen auf den Linux-Kernel — wie von Linus gefordert — so gering wie möglich zu halten, beschränkt sich das Framework auf die Einführung von restriktiven Regeln(*Policies*). Sogenannte „authoritative hooks“, also Regeln bei denen das geladene Sicherheitsmodul Entscheidungen des DAC übergehen kann, würden wesentlich mehr Eingriffe in den Linnux-Kernel benötigen und wurden daher nicht in das Framework aufgenommen.

Wie in Abbildung 4.1 zu sehen ist, stellen die „Security Hooks“ eine Nachfrage zur weiteren Einschränkung der bereits gewährten Rechte an das Sicherheitsmodul. Nur wenn zuvor Zugriff auf ein Objekt gewährt wurde, werden die „Security Hooks“ ausgeführt. Ein Verweigerung des Zugriffs auf DAC-Ebene wird sofort mit Hilfe des Return-Wertes nach oben propagiert.

[6] Siehe [WCS+02]

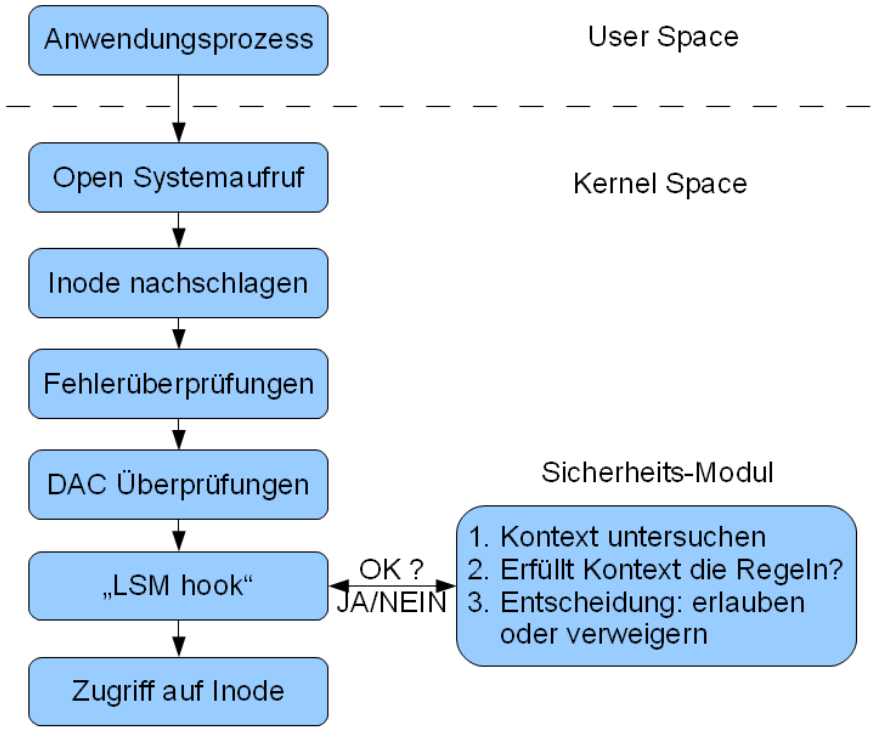

Abbildung 4.1: Überprüfung der „Security Hooks“ der Linux Security Modules

4.1.4 Entfernen weiterer Überprüfungen durch Anpassung an die virtuelle Maschine

Einige Funktionen führen Überprüfungen durch, die durch die Anpassung an die virtuelle Maschine überflüssig geworden sind. Stellvertretend für die teils sehr unterschiedlichen Überprüfungen, die noch entfernt werden konnten, werden im folgenden drei Beispiele genannt:

- So ist bei der Minimierung der Kernelkonfiguration ebensfalls die Unterstützung für große Dateien (> 2 Gigabyte) entfernt worden. Der Systemaufruf `open()` benutzt an mehreren Stellen Abfragen, die das Setzen eines Flags zur Behandlung von großen Dateien forciert.
- Ein anderes Beispiel für das Entfernen überflüssiger Überprüfungen ist die Überprüfung, ob das Datei-System mit dem Flag „*noexec*“ gesetzt wurde. Da bekannt ist, dass dieses Flag nicht in `/etc/fstab` gesetzt ist, können diese Abfragen auch entfernt werden.
- Ein letztes Beispiel findet sich in der sehr häufig aufgerufenen Funktion `inode_permissions()` aus der Pfadnamensauflösung NAMEI (namei.c). In dieser wird allgemein überprüft, ob ein Schreibzugriff auf ein nur lesend eingehängtes Dateisystem angefordert wurde.

Dies sind einige Beispiele von Stellen, an denen durch die Entfernung der Überprüfungen und eventuellen weiteren Behandlungen Zeit eingespart werden konnte.

4.2 Bordmittel

Um die Umgebung — das Betriebssytem und die Virtualisierung — weiter auf die Anwendnung anzupassen, müssen die folgenden Punkte noch zusätzlich durchgeführt werden. Diese Punkte werden zusammengefasst unter dem Begriff „Bordmittel“ behandelt. Rein technische Maßschneiderungen auf der Ebene der C-Sprache selbst, wie etwa das „Loop-Unrolling“, können dem Compiler überlassen werden und werden im Abschnitt 4.2.1 behandelt. Den Scheduler für ein Blockdevice zu ändern ist trivial und wird in diesem Abschnitt nicht betrachtet. Die Ergebnisse dieser Anpassung finden sich im Abschnitt 5.2.2.6. Änderungen auf der Virtualisierungs-Ebene, wie das Einschalten der Paravirtualisierung oder die Benutzung paravirtualisierter Treiber für I/O und Netzwerk werden im Abschnitt 4.2.2 Paravirtualisierung und im Abschnitt 4.2.2.1 paravirtualisierte Treiber behandelt.

4.2.1 GCC Compiler Optimierungen

Optimierungen, die auf der technischen und nicht der logischen Seite der Quell-Code-Ebene stattfinden, werden im Allgemeinen von Compilern sehr gut umgesetzt. Bekannte Verfahren wie Loop-Unrolling, Inlining von Funktionen und die Analyse „auf nicht erreichbaren Code“ werden von Compilern unterstützt.

4.2.1.1 GCC Optimierungsstufen

Der GCC Compiler unterscheidet verschiedene Standard Optimierungsstufen. Nach Gough [Gou04] sind dies insgesamt vier Optimierungsstufen: -O1, -O2, -O3 und -Os.

Optimierungsstufe -O1

Sinn und Zweck der ersten Optimierungsstufe ist es, ein optimiertes Image in einer kurzen Zeit zu bekommen. Optimierungen, die in dieser Standardstufe aktiviert sind, brauchen wenig Zeit, um ihre Optimierungsaufgabe zu lösen. In dieser Optimierungsstufe werden Optimierungen verwendet, die teilweise unterschiedliche Optimierungsziele haben: die Optimierung der Performance und die Optimierung der Größe des Quellcodes. Potenziell kann es hier also stets zu Trade-Offs kommen.

Optimierungsstufe -O2

Die zweite Optimierungsstufe enthält alle Optimierungen aus der ersten Stufe, sowie alle in einer Architektur unterstützten Optimierungen, die keinen Trade-Off zwischen Rechengeschwindigkeit und Codegröße besitzen. Beispielsweise sind die Optionen für das Loop-Unrolling[7] und das Inlining von Funktionen, wegen ihreres Trade-Offs nicht in der Optimierungsstufe -O2 enthalten.

Optimierungsstufe -Os

Die Optimierungsstufe -Os benutzt alle Optimierungen der Optimierungsstufe -O2, die nicht die Code-Größe verändern. Alignment-Optimierungen können sowohl die Performance steigern, wie auch den Verbrauch an Code-Größe und Speicherplatz. Das Ausrichten (*Alignment*) der Adressen an den Architekturgrenzen geschieht meistens in Faktoren von zwei, so dass beim Ausrichten Leerräume entstehen können, die nicht effektiv genutzt werden. Aus den zuvor genannten Gründen befinden sich in der Optimierungstufe -Os jedoch keine Alignment-Optimierungen von Funktionen, Schleifen oder Sprüngen — auch nicht, wenn diese bereits in -O2 aktiviert waren.

Optimierungsstufe -O3

Die Optimierungsstufe -O3 liefert das höchste Standardoptimierungslevel und ist damit die letzte der vier Standard-Einstellungen. Hier wird Ausführungs-Geschwindigkeit vor Code-Größe bevorzugt. Alle Optimierungen des Optimierungslevels -O2 sind in dieser Stufe bereits aktiviert. Das Inlining von Funktionen ist bereits aktiviert.

Da in der Optimierungsstufe -O3 viele Optimierungen zusätzlich vorgenommen werden, die die Code-Größe teilweise erheblich vergrößern können, wird die Standardstufe -O2 zum Kompilieren des Linux-Kernel beibehalten und nur die erwünschten weiteren Optionen hinzugeschaltet.

Neben den verschiedenen Optimierungsstufen ist es auch möglich den Quell-Code für eine bestimmte Rechner-Architektur zu optimieren. Dies geschieht über die Flags `-march=` und `-mtune=`. Man kann entweder die Rechner-Architektur direkt angeben, zum Beispiel `-march=i686`, um Code zu generieren, der auf allen CPUs der i686-Familie und dazu abwärtskompaktiblen CPUs läuft, oder man überlässt mit der Einstellung `-march=native` dem Compiler die Entscheidung, welche Architektur-Merkmale[8] ausgewählt werden. Zusätzlich zur Generierung

[7] Loop-Unrolling bezeichnet das vorteilhafte „Abrollen“ einer Schleife, mit dem die Anzahl der Schleifendurchläufe reduziert wird.

[8] Befehlssatzerweiterungen wie MMX, SSE, etc.

von optimiertem Code für die in der virtuellen Maschine genutzten Code wurden noch zwei weitere Optimierungen ausgewählt:

- `-funroll-loops`[9] Eine Technik die Schleifen für weniger Iterationen optimiert, jedoch zu Lasten der Code-Größe geht.
- `-finline-functions`[10] Diese Technik bindet Funktionen, die „klein genug“ sind, direkt in die aufrufende Funktion ein. Der Overhead durch den Funktionsaufruf kann so gespart werden.

Ob die beiden zu letzt genannten Compiler-Flags den Linux-Kernel beschleunigen oder nicht, kann nicht pauschal beantwortet werden. Daher müssen die Compiler-Flags auf ihre Performance hin untersucht werden.

Das Makefile des Linux-Kernels wird daher um die für die Optimierungsflags `-march=native` `-mtune=native` `-funroll-loops` `-finline-functions` ergänzt.

4.2.2 Paravitualisierung

Um im Gast-System Paravirtualisierung benutzen zu können, muss das Gast-System angepasst werden. Ab der Version 2.6.21 ist die VMI-Schnittstelle von VMWare im Mainstream-Kernel integriert. Um die Paravirtualisierung zu aktivieren sind zwei Schritte notwendig.

Erstens müssen in der Kernelkonfiguration die entsprechenden Schalter für die Paravirtualisierung unter Linux gesetzt sein. Nach dem Kompilieren und Neustarten ist der Gast vorbereitet. Das Gast-System ist nun bereit, über die VMI Schnittstelle mit dem Hypervisor zu kommunizieren. Hierdurch entfällt die aufwendige Emulation von einigen Befehlen, zum Beispiel bei Festplatten- oder Netzwerkzugriffen.

Zweitens muss die Option `Support VMI Paravirtualisation` im VMWare Server für die entsprechende VM eingeschaltet werden. Obwohl der VMWare Server angeblich die Paravitualisierung beherrschen soll, ist das Einschalten dieser Option in den Menüs nicht möglich, die Option ist, selbst bei ausgeschalteter VM, ständig ausgegraut. Jedoch lässt sich die Paravirtualisierung auch manuell über die vmx-Konfigurationsdatei der virtuellen Maschine setzen.

4.2.2.1 Paravirtualisierte Treiber

Die zusätzliche Nutzung der paravirtualisierten Treiber, die mit den VMWare Tools installiert werden, gestaltet sich unter Linux jedoch ein wenig komplizierter

[9] Siehe: http://www.technovelty.org/code/arch/funroll.html
[10] Siehe: An Inline Function is As Fast As a Macro — http://gcc.gnu.org/onlinedocs/gcc/Inline.html

als unter Windows. Während unter Windows Direkt-Installer zum Installieren gestartet werden können, sind die vorgefertigten Pakete unter Linux bereits recht alt. Die als Tarball mitgelieferten VMWare Tools sind also zu entpacken und für den entsprechenden Kernel zu kompilieren. VMWare bietet hier ein Perl-Script an, mit dem der Benutzer geführt alle zu kompilierenden Module selbst übersetzen kann.

Das gestartete Skript bricht nach Eingabe einiger Daten beim Kompilieren mehrerer Module ab. Eine intensive Internetrecherche ergibt, dass der benutzte Linux-Kernel 2.6.28.10 leicht geänderte Datenstrukturen besitzt und die VMWare Sourcen gepatcht werden müssen. Nach dem Patchen der Sourcen können die Module kompiliert und die paravirtualisierten Treiber für I/O und Netzwerk genutzt werden.

Will man in der virtuellen Maschine mehrere Kernel-Versionen bereitstellen und verwalten, um etwa einen maßgeschneiderten und einen nicht maßgeschneiderten Kernel starten zu können, so ist die Kernel-Kennung(*Magic Version*) stets eine andere. Die VMWare Tools müssen, da sie als Module in den Kernel eingebunden werden, zur Nutzung erneut kompiliert werden. Dies ist über das mitgelieferte Perl-Skript jederzeit möglich.

Die „Kernel Virtual Machine“ ist bei der Inbetriebnahme der paravirtualisierten Treiber erheblich unkomplizierter. Der Kernel muss lediglich mit den passenden Kernel-Parametern für die Nutzung der VIRTIO Treiber, sowie mit Parametern zur Erstellung von VIRTIO Netzwerk- und Datei-System-Treibern kompiliert und installiert werden. Beim Start der virtuellen Maschine müssen die VIRTIO Treiber entsprechend als Schalter mit angegeben werden. Beim Booten der zuvor bereits installierten Kernel ist zu beachten, dass die Festplatte nicht mehr wie unter VMWare als devsda oder wie unter KVM normalerweise unter devhda, sondern unter devvda zu finden ist.

Editiert man beim Booten die Kernel-Einträge passend, so ist es möglich, eine virtuelle Maschine von VMWare unter KVM mit und ohne paravirtualisierte Treiber zu nutzen.

Wird die Hardwareversion geändert, so ist ein direktes Starten der virtuellen Maschine mit KVM nicht mehr möglich[11].

4.3 Kernel Mode Linux

Im Abschnitt 3.2.3 wurde die Grenze zwischen User- und Kernel-Mode als eine Optimierungsmöglichkeit identifiziert. In Maeda [Mae03, Mae10] wird eine Lö-

[11] Siehe [WR10]

sung für dieses Problem dargestellt und hier kurz vorgestellt. Anwendung fand diese Lösung, aufgrund der zum Schluss ausgeführten Gründe, jedoch nicht.

Kernel Mode Linux verändert den `execve()` Systemaufruf so, dass eine spezialisierte Variante von start_thread aufgerufen wird. Diese neue Funktion „start_kernel_thread“ schreibt dann anstelle des Wertes ___USER_CS, den Wert ___KERNEL_CS in das Code-Segment-Register (CS-Register).[Mae10]

Das CS-Register ist eines von sechs Registern, die in der x86 Architektur dazu verwendet werden, Segment-Selektoren zu speichern. Darüber hinaus hat das CS-Register die Besonderheit, ein 2-Bit großes Feld für das Current Privilege Level zu verwalten (siehe Abschnitt 2.1.1.2- Protected Mode). [BC06]

Durch den Start des Threads mit ___KERNEL_CS wird ein normales Anwender-Programm (*user program*) im Kernel Modus gestartet. Möglichkeiten das Privilege-Level zu ändern, ergeben sich nur durch Far-Calls und Interrupts, sie erhöhen das Privilege-Level, oder durch die Rückkehr von einem Far-Call oder Interrupt. Das Programm bleibt also effektiv im Kernel Mode. [Mae10]

Nachdem ein Programm im Kernel-Mode gestartet ist, gibt es drei große Herausforderungen:

- Vermeidung von „Stack Starvation“,
- manuelles Austauschen des Stacks,
- verlorene Interrupts.

Um den IA32-Hardware-Task-Mechanismus zu motivieren, wird hier das Problem der Stack Starvation vorgestellt, die beiden anderen Problemen jedoch nicht. Weitere Informationen dazu gibt [Mae10].

4.3.1 Vermeidung von „Stack Starvation“

Das Verhungern des Stacks ist das größte Problem im Zusammenhang mit Kernel Mode Linux.

Definition 4.1 (Stack Starvation)
Wird ein Anwendungsprogramm im Kernel Modus ausgeführt und ein Seitenfehler (page fault) tritt dabei auf, so erzeugt das Anwendungsprogramm eine Seitenfehler-Ausnahme (page fault exception) auf seinem — im Kernel-Modus ausgeführten — Benutzer-Stack. Der Benutzer-Stack liegt nun aber im Kernel. Im Falle eines Fehlers versucht die IA-32 CPU verschiedene Register auf dem Stack zu sichern. Da das Programm im Kernel-Modus ausgeführt wird, versucht die CPU effektiv die Register auf demselben Stack zu sichern, der den page fault ausgelöst hat und scheitert erneut. Schließlich gibt die CPU auf und startet neu.[Mae10]

Um das Problem zu lösen, benutzt KML [Mae10] den „IA-32 hardware task mechanism“, um Ausnahmen, wie einen Page-Fault, zu behandeln. Anstatt die Register auf dem Stack zu speichern, wechselt die CPU den Ausführungskontext zu einem speziellen anderen Kontext. Die „Stack Starvation“ tritt nicht auf. Sämtliche Ausnahmen auf diese Weise zu behandeln, nimmt viel Rechenzeit in Anspruch. Allerdings sind doppelte Seitenfehler auf einem Memory-Stack recht selten. Desswegen hat diese Behandlung für die meisten Programme vernachlässigbare Kosten.

Ein Problem, welches mit dem letzten der drei großen Herausforderungen zusammenhängt, ist die Behandlung von „Non-Maskable-Interrupts“ durch den IA32-Task-Mechanismus und die eventuelle Inkonsistenz der „Descriptor Tables“. Wenn der „IA32-hardware-task-mechanism“ benutzt wird, um Tasks zu wechseln, so müssen alle Segmentregister (CS,DS,ES,FS,GS,SS) und das LDTR[12] neugeladen werden. Zum Wechseln ist es daher erforderlich, dass die globalen und lokalen Tabellen konsistent sind. Andernfalls wird ein „Task State Segment“-Fehler ausgelöst, aus dem zurückzukehren zu kostspielig ist „[...] and it is too complex to recover from the exception“ ([Mae10]). Kernel Mode Linux löst das Problem der konsistenten Tabellen, geht jedoch davon aus, dass die exklusiv vom Kernel verwalteten Register CS,DS,ES nicht gesichert werden müssen. Da der Kernel in einer virtuellen Maschine ausgeführt wird, verändern die Virtualisierungslösungen unter anderem das CS-Register, um ihrerseits dem Betriebssystem vorzugaukeln, es liefe in Ring 0 (siehe Definition 2.8). Aus diesem Grund konnte diese Technik in der virtuellen Maschine nicht angewendet werden, sondern führt sowohl unter KVM als auch VMWare zu einem Absturz des Kernels. Im Host jedoch ist KML ohne Probleme zu verwenden.

[12]LDTR = Local Descriptor Table Register

5 Evaluation

Im Abschnitt 5.1 wird das verwendete System und seine Vorbereitung für die Testläufe genauer beschrieben.

Um die in dieser Arbeit gefundenen Optimierungsmöglichkeiten zu evaluieren, wurden umfangreiche Tests durchgeführt. Die Ergebnisse der Tests werden in Abschnitt 5.2 Quantitative Analyse vorgestellt. Die Erkenntnisse aus den Ergebnissen dieser Tests werden in Abschnitt 5.3 zusammengefasst.

Mit Hilfe der verschiedenen Messmethoden — innerhalb der VM mit SystemTap und außerhalb mit den anwendungsspezifischen Benchmarks — konnten die jeweiligen, teils sehr verschiedenartigen, Flaschenhälse identifiziert werden. Durch die enge Verzahnung vieler Mechanismen und Methoden ist ein komplexer Workflow entstanden, der die schnelle Entwicklung von Maßschneiderungscode unterstützt.

Die Untersuchungen wurden zunächst unter VMWare durchgeführt und später dann die Unterschiede bei der Verwendung von KVM herausgearbeitet.

5.1 Methode

Das zur Messung benutzte Testsystem wird die zugrunde liegende Messmethodik in Abschnitt 5.1.1 vorgestellt. Anschließend werden in Abschnitt 5.1.2 einige Besonderheiten der eingesetzten Werkzeuge erläutert.

5.1.1 Testumgebung

Das Testsystem, ein Toshiba Satellite L550, besitzt einen „Core 2 Duo“-Prozessor mit Penryn-Kern. Von den zwei Kernen des Prozessors wurde einer dem Gast-System zugeordnet. Von den 4 Gigabyte RAM des Rechners wurde dem Gast-System 1 Gigabyte Speicher fest zugeordnet. Die normale Speicherverwaltung des VMWare Servers sieht vor, dass der unbenutzte Speicher eines Gast-Systems anderen Gästen zur Verfügung gestellt werden kann (*Memory Page Trimming*). Hierdurch können mehr virtuelle Maschinen gleichzeitig betrieben werden, als es das zur Verfügung stehende RAM erlauben würde. Wenn ein Gast-System auf

seinen ihm versprochenen Speicher besteht und der physische Speicher nicht ausreicht, dann erzeugt VMWare einen virtuellen Speicherbereich, mit Hilfe einer Auslagerungsdatei, auf der Festplatte. Hierdurch wird die Performance stark einbrechen [Ahn09].

Durch den Verwaltungsoverhead bei der Verwaltung des Hauptspeichers kann das Problem des Auslagerns von Speicherbereichen in eine Auslagerungsdatei auf der Festplatte bereits auftreten, obwohl der Speicher des Host-Systems + dem versprochenen Speicher aller Gast-Systeme gerade ausreichend ist.

Ahnert [Ahn09] nennt einen Wert von 5 - 10% zusätzlichem Speicher pro virtueller Maschine zu Verwaltungszwecken. Das *Memory Page Trimming* wurde daher in der Konfigurationsdatei des Gast-Betriebsystems deaktiviert.

Tabelle 5.1 zeigt die Daten des Testsystems im Überblick:

Komponente	Spezifikation
Rechner	Toshiba Satellite L550
CPU	Core 2 Duo (P8700)
RAM	4 Gigabytes
Host OS	Linux 2.6.30.9
Virtualisierungslösung	VMWare Server 2.0.1 Build 156745 KVM -72
Gast OS	Linux 2.6.28.10
Gast RAM	1 Gigabyte
Gast Festplatte	4 Gigabyte, SCSI
Gast CPU	1 Prozessor (P8700)

Tabelle 5.1: Testsystem

5.1.2 Vorbereitung des Kernels des Gast-OS

Da alle Werkzeuge sehr eng am Kernel arbeiten, waren anfangs einige Schwierigkeiten zu überwinden. Verwaltet man, wie in dieser Arbeit, mehrere Kernel zu Testzwecken gleichzeitig, wird man unterschiedliche Versionen desselben Kernels verschieden kennzeichnen und in grub installieren wollen. Eine Möglichkeit zu dieser Verwaltung bietet die Kernel-Konfiguration. Unter `General Setup -> Local Version` kann man einen „version string“ vergeben, der automatisch an die Release-Nummer des Linux-Kernels angefügt wird. Dies führt dazu, dass alle Module, die für eine spezielle Kernelversion gebaut wurden, erneut für den neuen Kernel übersetzt werden müssen. Im Detail mussten folgende Anpassungen vorgenommen werden:

- paravirtualisierte Treiber
  Die ersten neu zu übersetzenden Module fallen sofort beim Neustart des

Kernels auf. Es sind die paravirtualisierten Treiber von VMWare. Diese lassen sich leicht mit „vmware-config.pl“ neu kompilieren und fügen sich reibungslos in den neuen Kernel ein.

- Oprofile
  Oprofile benötigt zum Debuggen eine passenden vmlinux-Kernel[1]. Dieser Vorgang läßt durch eine Änderung in der Konfigurations-Datei der Paketverwaltung[2] leicht automatisieren.

- KSplice
  Bei KSplice muss die aktuelle System.map, sowie die aktuelle Konfigurationsdatei[3] aus dem boot-Verzeichnis in ein Unterverzeichnis `ksplice/` in den Linux Sourcen kopiert werden.

- SystemTap
  SystemTap verhält sich hierbei unkritisch, solange für jeden Kernel anstelle des Symlinks /lib/modules/“Kernel-Bezeichnung“/build ein kopiertes Build-Verzeichnis unter /lib/modules/“Kernel-Bezeichnung“/ liegt oder die Sourcen, auf die der Symlink zeigt, aktuell für diesen Kernel sind.

Obwohl die Methodik recht komplex ist, muss man nur selten den Kernel neu konfigurieren, nachdem man den Maßschneiderungsschritt Kernel-Konfiguration durchgegangen ist. Trotz der Seltenheit des Auftretens empfiehlt es sich mit Scripten diese Vorgänge zu automatisieren, nicht allein um die Vorgänge zu beschleunigen, sondern um sie wartungsarm zu halten.

5.2 Quantitative Evaluation

Im Abschnitt 5.2.2 werden die Auswirkungen von verschiedenen Optimierungsmöglichkeiten, die nicht zur Maßschneiderung des Linux-Kernels gehören, aber teilweise signifikante Auswirkungen auf die Performance der Anwendung haben, untersucht. Diese Optimierungen werden unter dem Begriff „Bordmittel“ zusammengefasst. Den Auswirkungen der Maßschneiderungen auf das Virtual File System (VFS), der Zugriffskontolle (DAC), sowie der erweiterten Sicherheit durch die Linux Security Modules (*LSM*) und der Entfernung von Überprüfungen, die in der von dem Betriebssystem benutzten Konfiguration stets gleich ausgewertet werden, geht der Abschnitt 5.2.3 nach. Der interessanten Fragestellung, ob ein Plattformwechsel die Ergebnisse der Maßschneiderung erheblich beeinflusst, wird im Abschnitt 5.2.4 untersucht. Die Ergebnisse werden im Abschnitt 5.3 diskutiert.

[1] Man beachte, dass es sich nicht um die komprimierte Variante vmlinuz handelt.
[2] Bei Debian: `/etc/`
[3] Diese muss zum Beipiel von config-2.6.28.10 in .config umbenannt werden.

5.2.1 Messgrößen

Die, mit 10 parallel gestarteten wget-Befehlen, ausgeführten Tests[4] zeigten keine IO-Blocking-Zeiten. Daher werden nachfolgend nur die Stresstests durch den Autobench-Benchmark betrachtet.

Um die unterschiedlichen Fragestellungen angehen zu können, muss man zunächst sinnvolle Messgrößen definieren.

Als erstes muss man selektieren, welche Systemaufrufe relevant für die quantitative Untersuchung des Problems sind. Bei der gewählten Anwendung eines Webservers sind dies die neun Routinen `close, fcntl, ioctl, read, setsockopt, shutdown, stat, write` und `writev`. Nicht betrachtet werden dürfen natürlich Systemaufrufe, die Leerlaufprozesse oder andere für die Anwendung irrelevante Prozesse beinhalten, wie z.B. `poll, time` oder `select`.

Um maschinenunabhängige Aussagen treffen zu können, werden hier in der Regel Taktzyklen gemessen und keine Zeiten. Dies erlaubt auch bei den heutigen Computern, die im GHz-Bereich arbeiten, eine deutliche Genauigkeitssteigerung, da Zeiten unter Linux nur auf eine μs genau angegeben werden, Taktzeiten aber unterhalb von ns, also im ps-Bereich angesiedelt sind.

Man muss bedenken, dass jeder Systemaufruf durchaus durch Interrupts unterbrochen werden und daher in der Ausführungszeit oder Gesamtzahl der Zyklen erheblich erhöht werden kann. Ein typisches Beispiel für die Verteilungsfunktion der resultierenden Ausführungszeiten ist in Abb.5.1 zu sehen. Man sieht deutlich, dass der grösste Teil der Aufrufe sich bei den niedrigsten Werten konzentriert. In Einzelfällen, in denen offensichtlich Interrupts stattfanden, wurden sehr viel höhere Ausführungszeiten beobachtet. Es wurde daher bei Fragestellungen, die die Effizienz der einzelnen Maßschneiderungen hinsichtlich der Beschleunigung in den unteren Layers betrifft, über die 30 minimalen Taktzyklenzahlen gemittelt. Damit wird eine typische Maßzahl für die beschleunigte Ausführung bestimmt.

Soll die Gesamtperformance für die Modellanwendung beurteilt werden, ist die Gesamtzahl der Zyklen für die relevanten neun Systemaufrufe die bessere Messgröße.

Zu beachten ist, dass, auch bei dieser Messgröße, Interrupts auftreten können. Die Verteilung im Histogramm 5.1 vom Systemaufruf `writev()` zeigt deutlich, dass es trotz einer hohen Last vernachlässigbar wenige Aufrufe gibt, die nicht nahe der Minimalen CPU Zyklen ausgeführt werden. Die Gesamtzahl der Zyklen aller Systemaufrufe multipliziert mit der Taktzeit ist identisch mit der Zeit des gesamten Testlaufs.

[4] Folgen der Verlinkungen von der Startseite aus
Seitenabrufe alle 0,5-1,5 Sekunden

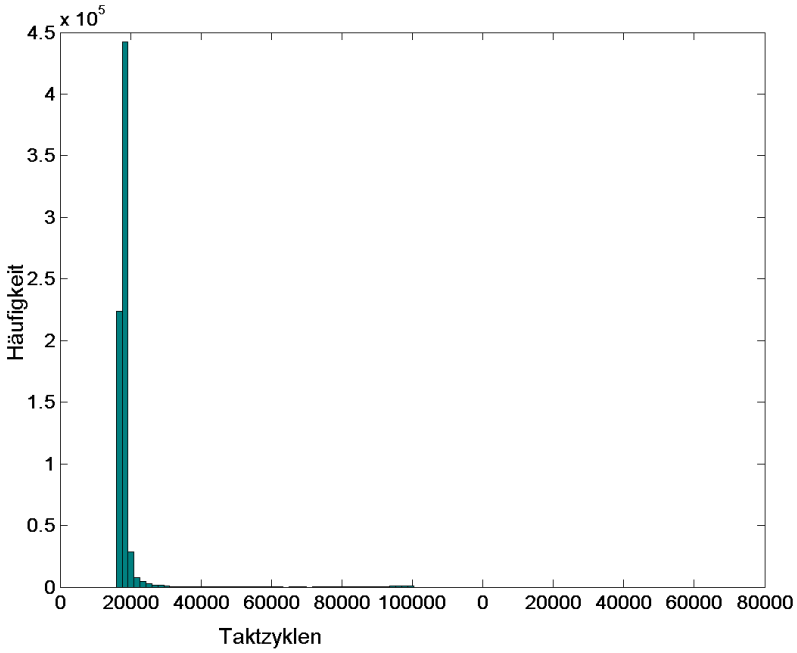

Abbildung 5.1: Beispiel einer Verteilung der Ausführungszeiten eines Systemaufrufes. Man erkennt die starke Häufung bei kurzen Ausführungszeiten.

5.2.2 Auswirkung der „Bordmittel“

Wie eingangs im Abschnitt 2.2.1 beschrieben, gibt es verschiedene Granularitäten der Maßschneiderung. Die unter „Bordmittel“ bezeichneten Optimierungen vertreten sowohl verschiedene Ebenen (Virtualisierungs-, Paket- und Konfigurationsebene), als auch Granularitäten (Paket = sehr grob, Konfiguration = grob) der Maßschneiderung.

Abschnitt 5.2.2.1 gibt einen Überblick über die Auswirkungen von Optimierungen auf die Paketebene, also dem Entfernen nicht benötigter Software. Einsparungspotenziale auf der Kernelkonfigurationsebene werden im Abschnitt 5.2.2.2 erörtert.

5.2.2.1 Paketoptimierung

Durch das Deinstallieren nicht benötigter Dienste und Programme werden auf einfachste Weise CPU-Zyklen gespart. Auch wenn die meisten Dienste ihre Aufgaben im Hintergrund erledigen und deshalb nicht dauerhaft CPU-Zyklen verbrauchen, können auf diese Weise eine Menge CPU-Zyklen gespart werden.

Selbst bei einer konsolenbasierten Installation können noch der cron-Dienst,

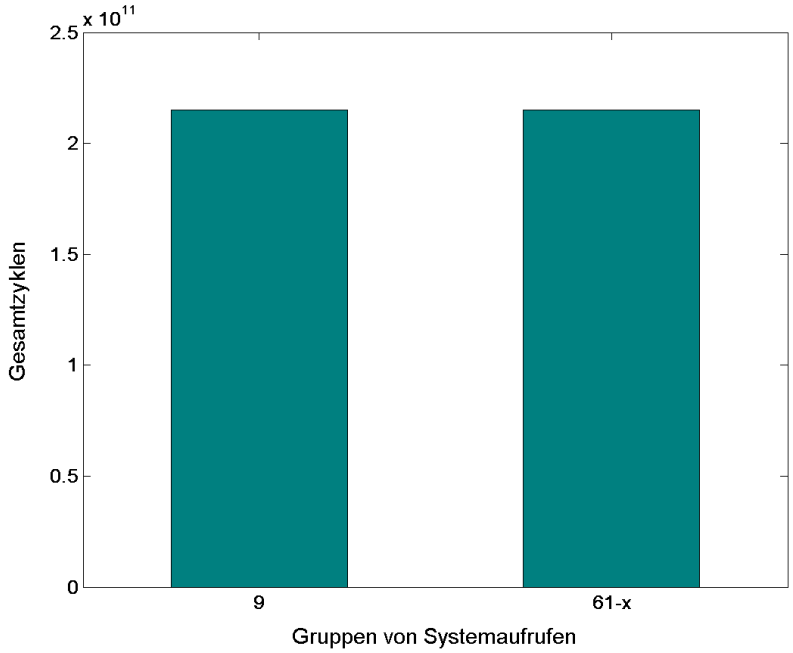

Abbildung 5.2: Einsparungen durch Paketoptimierung bei den neun wichtigen (**9**) bzw. allen(abzüglich idle-Prozessen)(**61-x**) Systemaufrufen

der Log-Files im Hintergrund archiviert, der DHCP-Client oder etwa der Maildienst exim4 deinstalliert werden. Insgesamt erreicht man eine Ersparnis, wie in Abbildung 5.2 zu sehen ist, von 0,04 % oder $8,35 \cdot 10^7$ Taktzyklen. Obwohl diese Einsparungen gegenüber der Maßschneiderung sehr gering ausfallen, sind diese die mit dem wenigsten Aufwand eingesparten CPU-Zyklen — denn es muss nur der Paketmanager zum Deinstallieren der Pakete aufgerufen werden. Wie zuvor erwähnt, unterstützen bei der automatischen Deinstallation Tools, wie deborphan und debfoster, den Anwender um die Abhängigkeiten zu anderen Paketen aufzulösen und eventuell weitere nicht mehr benötigte Pakete zu finden.

5.2.2.2 Kernel-Konfiguration

Bei der Kernel-Konfiguration entfernt man viele Treiber für nicht vorhandene Hardware. Dadurch wird die Zeit des Kernel-Kompilierens erheblich beschleunigt. Viel wesentlicher ist, die auf diese Art und Weise erreichte Verringerung der Komplexität.

Es können nicht nur Treiber aus der Konfiguration, sondern auch einige Belange entfernt werden. Bei der Maßschneiderung sind eine Menge Subsysteme, von denen die größten das Sound-System, die Energieverwaltung und das SMP-System waren, entfernt worden.

Kernel	Anzahl der Module	aktiv nach dem Booten	Linux Image (Bytes)	Konfigurationsdatei (Bytes)
Original	2059	45	277473486	96231
Minimal	50	0	30011208	46550

Tabelle 5.2: Vergleich der Eigenschaften eines Kernels der Originaldistribution und eines selbsterstellten minimalen Kernels

Minimiert man den Distributions-Kernel und passt ihn auf die Umgebung der virtuellen Maschine an, so fallen zwei Unterschiede sofort auf. Der erste Unterschied ist die Größe des selbst gebauten Debian Paketes, welches von 264 Megabytes auf 33 Megabytes (siehe Tabelle 5.2) fällt. Der zweite Unterschied ist die durchschnittliche Zeit die zum Kompilieren des Kernels benötigt wird; Sie fällt von 87 Minuten auf 6 Minuten.

Die Minimierung der Kernel-Konfiguration, durch Anpassen an das vorliegende System, wirkt sich nicht nur beim Kompilieren des Linux-Kernels aus, sondern ebenso in der Gesamtlaufzeit.

In der Abbildung 5.3 wird die Gesamtausführungszeit des Debian Lenny Distributionskernel gegenüber der Gesamtausführungszeit eines minimierten Kernels aufgelistet. Durch die Kernel-Konfiguration konnten 11% an Taktzyklen eingespart werden. Der Distributionskernel bietet einen Umfang von 2058 Modulen, von denen nach dem Starten des Rechners 45 Module benutzt werden. Darunter einige Module der Bereiche Energieverwaltung und Soundsystem, die in der minimierten Version ganz fehlen. Module, wie etwa der Netzwerkkartentreiber, sind im Minimal-Kernel bereits direkt einkompiliert.

Der minimierte Kernel profitiert von einigen Seiteneffekten. Der Kernel muss nun erheblich weniger Datenstrukturen verwalten. Durch die Verkleinerung der Menge an Datenstrukturen ist der Kernel schlanker und verbraucht weniger RAM. Desweiteren profitieren einige Kernel-Strukturen, wie etwa der sogenannte „Big Kernel Lock“, von der Vereinfachung des Linux-Kernels auf einen Prozessor. Der „Big Kernel Lock“, der bei einigen, über die verschiedenen CPUs verteilten Datenstrukturen eingesetzt werden muss, und dafür Sorge trägt, dass der Zugriff auf, über mehrere CPU's verteilte, Datenstrukturen sicher erfolgt, wird durch das Entfernen des Mehrprozessoren-Codes erheblich schlanker.

5.2.2.3 Paravirtualisierung

Zusätzlich zur verwendeten Virtualisierungslösung ist es beim VMWare Server möglich, Paravirtualisierung zu benutzen. Um die Paravirtualisierung bei dem

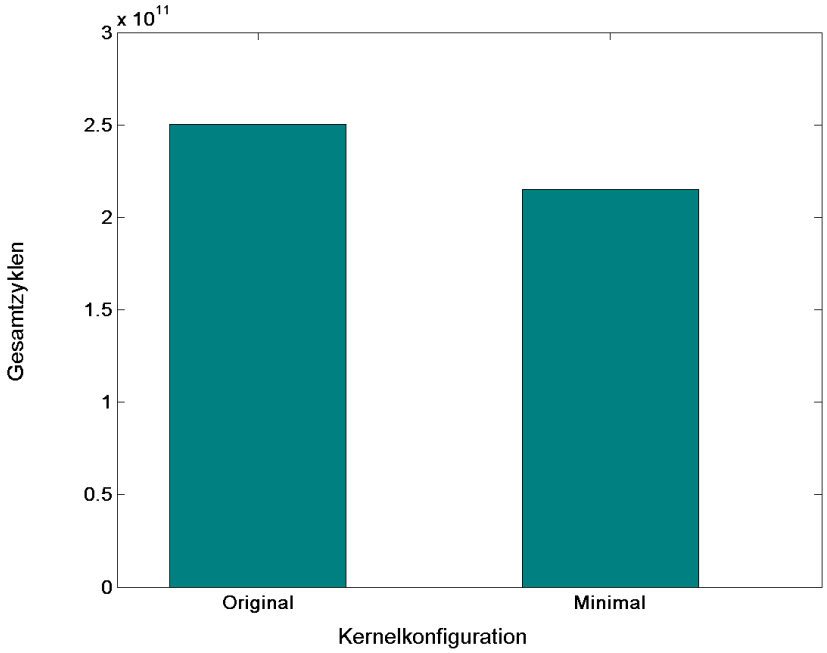

Abbildung 5.3: Auswirkung der Minimierung der Kernel-Konfiguration während eines Stresstests

VMWare Server einzusetzen, ist es nötig, dem Gast-System bewusst zu machen, dass es virtualisiert wird. Das geschieht durch eine Anpassung des Gast-Systems, genauer durch das Kompilieren des Linux-Kernels mit der Konfiguration zur Nutzung der VMI-Schnittstelle. Bevor der neue Kernel mit VMI-Unterstützung gestartet werden kann, muss die virtuelle Maschine ausgeschaltet und die vmx-Konfigurationsdatei der virtuellen Maschine angepasst werden. In der Konfigurationsdatei ist die Variable `vmi.present = true` zu setzen.

5.2.2.4 Virtuelle Hardware

Eine Betrachtung, welche Auswirkungen die Aktualisierung der virtuellen Hardware hat, konnte aus Kompaktibiliätsgründen zu KVM nicht berücksichtigt werden. Der Testaufbau beinhaltete exakt eine vmdk-Datei, welches das VMWare-Format für die virtuelle Behälterdatei darstellt. Diese kann sowohl unter VMWare, als auch unter KVM ausgeführt werden. KVM kann neuere Hardware-Versionen von VMWare jedoch nicht ohneweiteres ausführen, so dass mehrere Versionen der zu testenden virtuellen Maschine entstanden und eine Vergleichbarkeit nicht unbedingt mehr gegeben wäre. Durch die Möglichkeit des grub beim Start des Rechners den Boot-Eintrag zu editieren, konnte KVM die Festplatte passend nach /dev/hda mounten[5].

[5]VMWare benutzt SCSI-Treiber und mountet daher nach `/dev/sda`.

In „Performance Evaluation of VMXNET3 Virtual Network Device“ [VMW09] stellt VMWare die Verbesserungen des VMXNET 3 Treibers[6] vor und vergleicht diese mit dem alten VMXNET 2 Treiber[7].

Teil der virtuellen Hardware ist die Netzwerkvirtualisierung mit dem VMXNet-Treiber. Ein verbesserter Netzwerktreiber könnte sich positiv auf das Anwendungsverhalten auswirken, da der Lighttpd I/O-lastig ist. Der durch Netzwerkvirtualisierung geschaffene Overhead wird von drei Faktoren beeinflusst:

- der Virtualisierung der CPU,
- der Memory Management Unit (MMU)
- und der I/O Performance.

Hat man einen Performancevorteil durch mindestens einen der drei Faktoren erreicht, so ist die Leistung der Netzwerkvirtualisierung bereits gesteigert worden. Hardwareversion 4 der virtuellen Hardware unterstützt den paravirtualisierten Netzwerktreiber VMXNet 2. In Hardwareversion 7 ist bereits die neue Generation der Netzwerktreiber, VMXNet 3 enthalten. In der Studie „Performance Evaluation of VMXNET3 Virtual Network Device“ von VMWare werden unter anderem die Neuerungen des VMXNet 3 Treiber erklärt. Die für Linux Gäste interessanten Neuerungen sind: NAPI kompaktibilität, Large Receive Offload (LRO) und größere Transmit / Receive Buffer. Large Receive Offloads ist eine Technik, die nur für Transfers zwischen zwei VMs zugelassen ist; andere Transitionen werden nicht unterstützt.[VMW09] Während Large Receive Offload für unsere Szenarien uninteressant ist, versprechen NAPI-Kompatibilität und größere Transmit / Receive Buffer eine Steigerung der Performance. NAPI ist ein Mechanismus der durch das Hin- und Herwechseln zwischen Interrupt- und Polling-Mode während der Paket-Empfangsphase fähig ist, höhere Paketströme zu verarbeiten. Von größeren Transmit / Receive Buffern profitieren Burst und Peek-Performance Szenarien, so wie der Autobench-Stresstest.

5.2.2.5 GCC-Optimierung

Die automatisierte Maßschneiderung mit Hilfe des GNU C Compilers zeigte wenig positive Auswirkungen und verschlechterte in einigen Fällen die Performance. Wie im Abschnitt 4.2.1 erwähnt, gibt es verschiedene Standard-Stufen der Optimierung. In den Tests zur GCC-Optimierung wurde die Stufe -O2 gewählt, da in dieser Stufe nur Optimierungen sind, die auch Performance-Vorteile bringen. Zusätzlich wurden noch die Flags `-finline-functions -funroll-loops -march=native`
und `-mtune=-native` gewählt. Durch die Optionen `-march=native` und

[6]Dieser Treiber ist in Hardwareversion 7 enthalten.
[7]Dieser Treiber ist in Hardwareversion 4 enthalten.

`-mtune=-native` werden der vom Prozessor unterstützte Befehlssatz und alle unterstützten Erweiterungen[8] aktiviert.

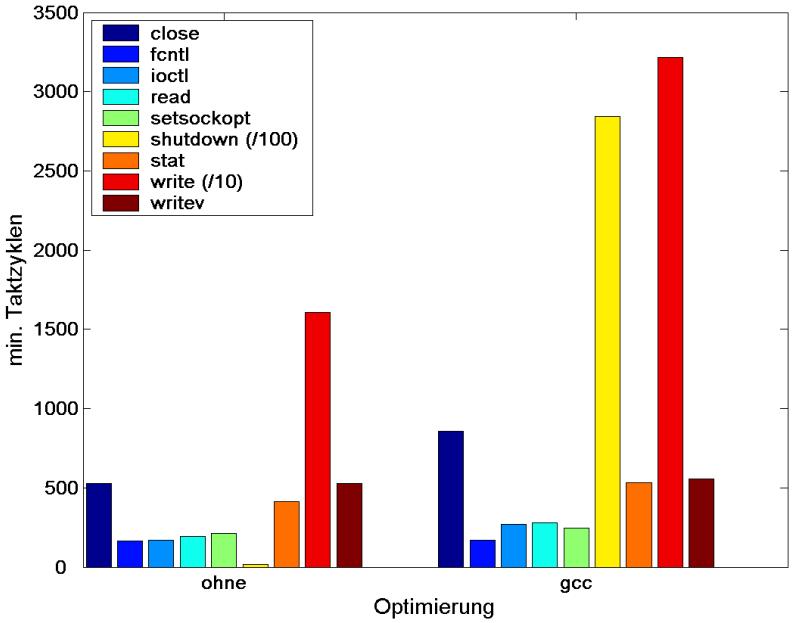

Abbildung 5.4: Einfluss einer Optimierung mittels GCC auf die minimale Anzahl von Taktzyklen der neun wichtigsten Systemaufrufe

In der Abbildung 5.4 sieht man, dass die Optimierungen bestenfalls keine Auswirkungen hatten, im schlimmsten Falle jedoch zur Verdopplung der minimalen Taktzyklen führen konnten.

Die besten Compiler-Einstellungen zu finden, scheint durch die umfangreiche Dokumentation der Compiler-Flags, eine leichte Aufgabe zu sein. Ladd [Lad05] erkannte aber, dass trotz der umfangreichen Dokumentation der einzelnen Compiler-Flags, viele Optimierungen sich gegenseitig beeinflussten. Er schlägt daher vor, seinen genetischen Algorithmus zu benutzen, um ein möglichst optimales Set von Compiler-Flags zu finden. Das Finden von optimalen Compiler-Flags mit Hilfe von Acovea war in der Dauer dieser Diplomarbeit jedoch nicht möglich. Acovea benötigt für ein eingermaßen gutes Ergebnis 4000 Kompilierungen und genauso viele Testläufe der zu untersuchenden Software. Dieser Aufwand ist für den Linux-Kernel aus zeitlichen Gründen nicht praktikabel.

[8] Zum Beipspiel sind dies MMX, SSE, SSE2, etc.

5.2.2.6 I/O Scheduler

Die richtige Wahl des I/O Schedulers ist für Webserver entscheidend. Ruft ein Client eine Webseite bei dem Webserver ab, so schaut der Webserver erst im Cache, dann auf der Festplatte nach der auszuliefernden Datei. Hat der Webserver die Datei gefunden, benutzt er den `write()` Aufruf, um die Datei über den Socket zu versenden.

Mit dem voreingestellten „Completely Fair Queueing“-Scheduler (siehe Abbildung 5.5) treten nach 1000 Sekunden, im von autobench erzeugten Stresstest, sehr viel höhere Wartezeiten für die I/O Anforderungen auf. Zu dieser Zeit finden bereits 1100 Anfragen an den Webserver statt. Die Wartezeiten der abzusetzenden I/O Anforderungen, die in Abbildung 5.5 aufgetragen sind, schnellen stark in die Höhe. Auf der Client-Seite werden zu spät oder gar nicht beantwortete Webseitenabrufe mit dem Fehler „client-timo“ im autobench-Stresstest bezeichnet.

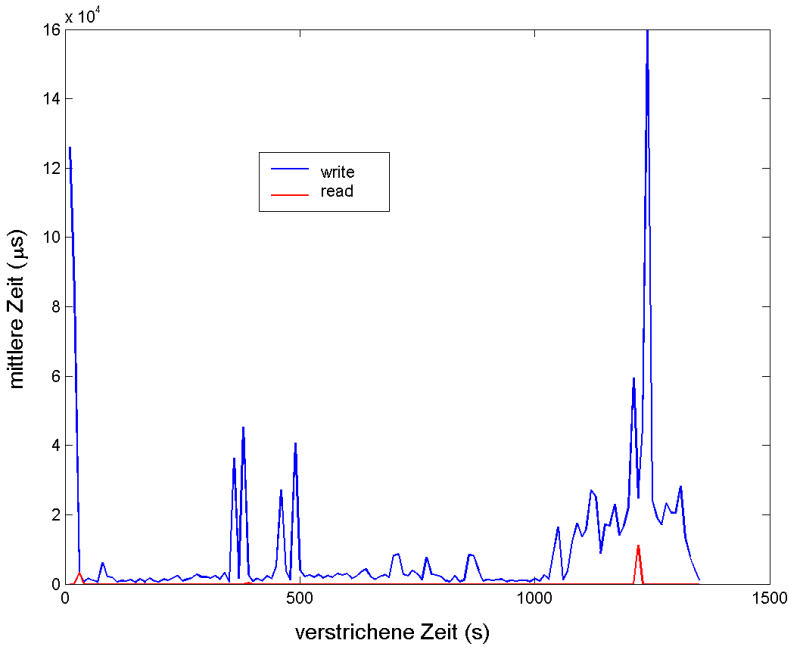

Abbildung 5.5: Complete Fairness Queueing - Scheduler im Stresstest

Wählt man den Deadline-Scheduler aus, so garantiert dieser zwar nicht die Ausführung innerhalb der Deadline, sondern setzt I/O Anforderungen, die ihre Deadline verpassen sofort ab, das heißt er reiht sie direkt in die Dispatcher-Queue ein (vgl. Abschnitt 3.2.2). Dies bedeutet, dass I/O Anforderungen beim Überschreiten der Deadline bereits wenige Zeiteinheiten später ausgeführt werden. In Abbildung 5.6 ist gut zu erkennen, dass der Deadline-Scheduler ein gutes, an die steigende Last angepasstes Durchschnittsverhalten hat. Je mehr Webseitenabrufe pro Sekunde erfolgen, desto länger werden die durchschnittlichen Blockingzeiten.

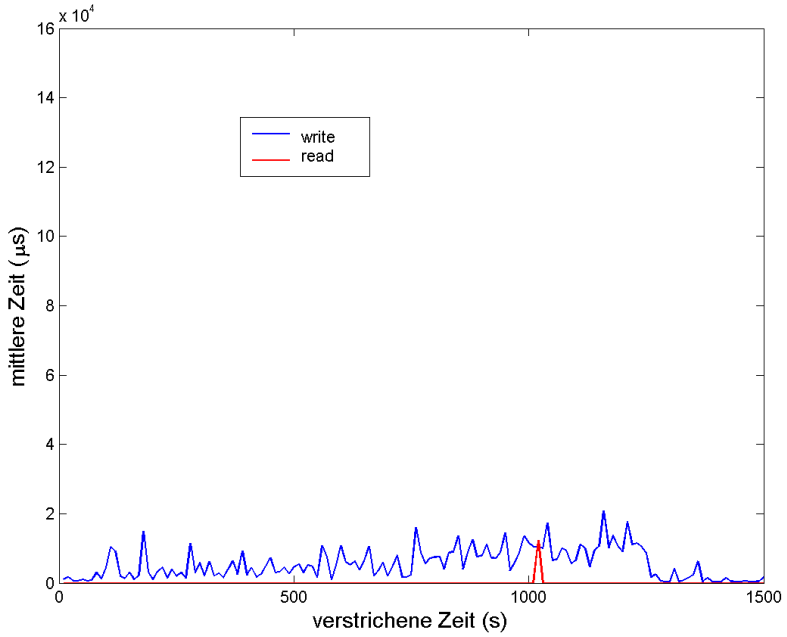

Abbildung 5.6: Deadline - Scheduler im Stresstest

Auch unter großer Last verändert sich das Verhalten des Schedulers nicht, wie es im Gegensatz hierzu bei dem CFQ-Scheduler der Fall ist. Selbst zum Schluss des Tests bei dem 2000 Verbindungen pro Sekunde aufgebaut werden, gibt es mit dem Deadline-Scheduler keine Client-Timeouts.

5.2.3 Auswirkung der Maßschneiderung

Um die Auswirkung der einzelnen Maßschneiderungsschritte zu überprüfen, wurden die während eines Stresstests gesammelten Werte für die Systemaufrufe selektiert.

Um die Einsparung an Gesamtzyklen zu bestimmen, wurden die Mittelwerte dieser Minimalwerte mit der Anzahl der Aufrufe des jeweiligen Systemaufrufs während des Stresstests multipliziert. In Abbildung 5.7 sieht man deutlich den Abfall von knapp 14,3 % in der Anzahl der Gesamtzyklen. Durch die kürzeren Ausführungszeiten wird die Chance einer vorzeitigen Unterbrechung (*Preemption*) durch andere Prozesse geringer, wodurch auch die maximalen Ausführungszeiten sinken.

Da die minimale Taktzahl der einzelnen Systemaufrufe einen untere Schranke für die Gesamttaktzahl der Anwendung vorgibt, stehen für Optimierungen, die in der Mehrzahl die Zahl der Unterbrechungen minimieren, nur die Taktzahlen

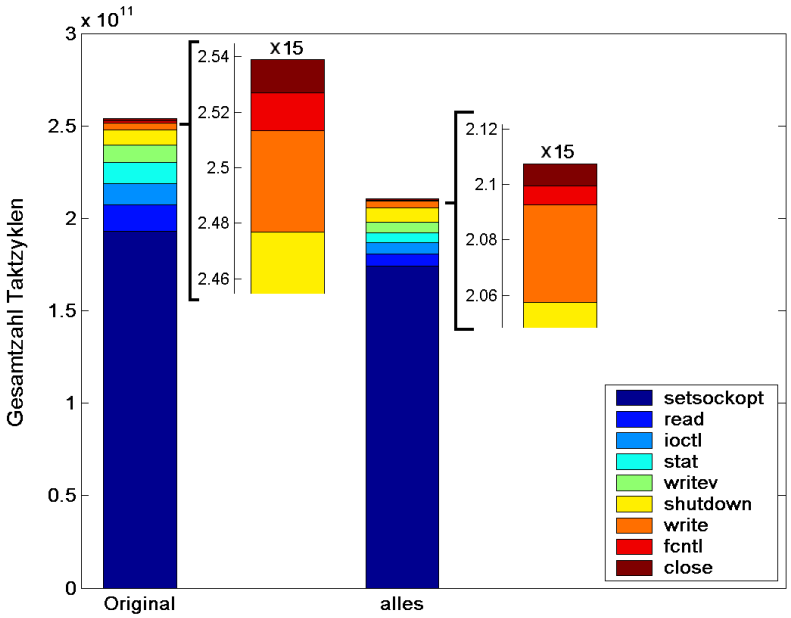

Abbildung 5.7: Mindesteinsparung Gesamtzyklen

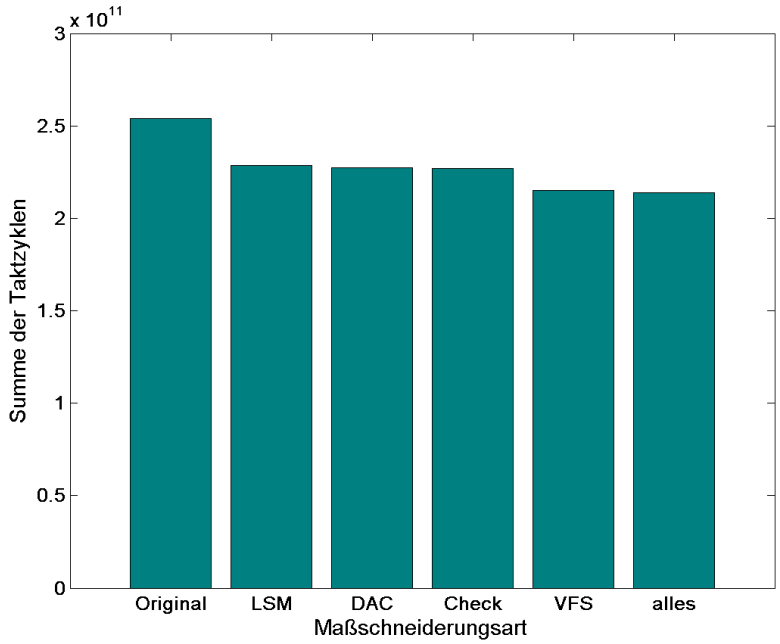

Abbildung 5.8: Überblick über die einzelnen Maßschneiderungen

oberhalb dieser Schranke zur Verfügung. Daraus folgt, dass die Performancesteigerung durch die gleichzeitige Anwendung aller Maßschneiderungen nicht additiv sein kann, sondern sich einem Grenzwert annähert. Je weniger Unterbrechungen durch die Anwendung einer bestimmten Maßschneiderung noch vorkommen, umso weniger können weitere Ansätze, noch starke Verbesserungen erzielen.

Sieht man davon ab, dass der größte Teil der Einsparungen durch die Minimierung der Unterbrechungen kommt, wirken sich die durch die Maßschneiderung zur Verschlankung des Kernels vorgenommen Änderungen kaum auf die Performance von Anwendungen unter dem 2.6er-Kernel aus.

Das oben beschriebene Verhalten ist sehr deutlich in Abbildung 5.8 zu erkennen. Während jede Maßschneiderungsart (LSM, DAC, Check und VFS) einzeln angewendet eine deutliche Reduktion der Gesamttaktzyklen ergibt, bringt die Anwendung aller Maßnahmen gemeinsam nur noch eine geringfügige Verbesserung.

Der gleichzeitige Einsatz mit den in Abschnitt 5.2.2 hinsichtlich ihrer Wirkung genauer untersuchten Methoden, die hier als Bordmittel bezeichnet werden, kann allerdings weitere deutliche Verbesserungen erzielen, da diese die Performance auf den anderen Ebenen, vor allem der Virtualisierung, steigern. Dies sieht man in der Abbildung 5.9 besonders eindrucksvoll. Die gemeinsame Anwendung der Maßschneiderungen und Bordmittel reduziert die Gesamtanzahl der benötigten Taktzyklen für die Serveranwendung während des zwanzigminütigen Stresstests von $2,5410 \cdot 10^{11}$ auf $1,2986 \cdot 10^{11}$. Es wurde also insgesamt eine Leistungssteigerung von 95,7 %, nahezu einem Faktor zwei, erreicht.

5.2.3.1 kmalloc-Aufrufe

Eine weitere interessante Maßschneiderung, die Optimierung von `kmalloc()`-Aufrufen, konnte aus Zeitgründen nicht mehr durchgeführt werden. Mit SystemTap ist es möglich, die Aufrufe im Kernel, die einen `kmalloc()`-Aufruf hervorrufen, zu untersuchen. Ein solches Systemtap-Skript könnte dazu dienen, Stellen zu finden, an denen ein Slab-Cache als Schicht zwischen die `kmalloc()` benutzende Datenstruktur und dem `kmalloc()`-Aufruf selbst eingeführt werden kann.

„*Der Slab-Layer unterteilt verschiedene Objekte in Gruppen, die Caches genannt werden. Jede [Gruppe] speichert einen anderen Typ von Objekt. Es existiert ein Cache pro Objekt-Typ.*“[LK05]

Definition 5.1 (Slab)
Die Caches ihrerseits werden in Slabs unterteilt, die eine Anzahl von gecachten Datenstrukturen enthalten. Ein Slab kann in einem von drei Zuständen sein: Full, Partial oder Empty. Fordert ein Teil des Kernels ein Objekt an, welches in einem

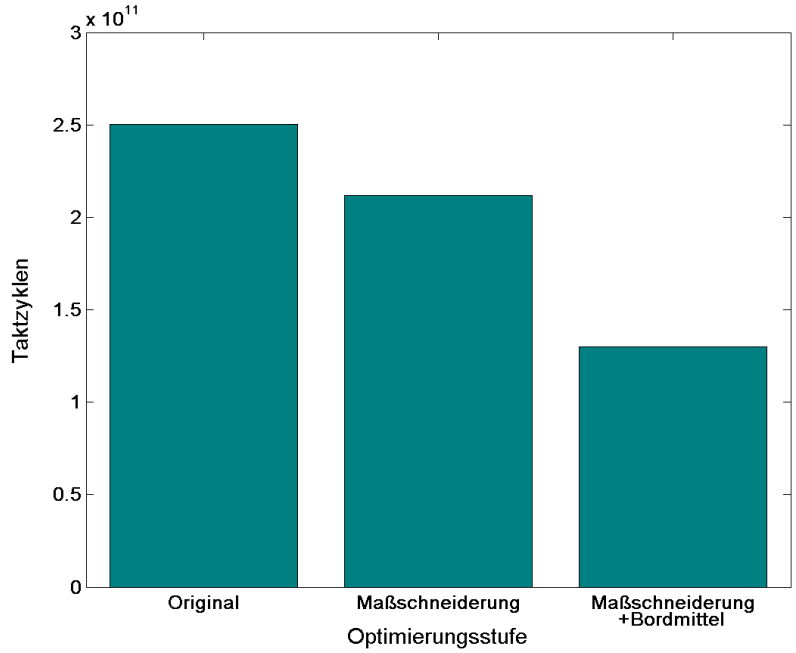

Abbildung 5.9: Vergleich der Verbesserungen durch Anwendung der Maßschneiderungen allein und bei gleichzeitiger Anwendung von Bordmitteln

Slab verwaltet wird, so wird die Anfrage von einem Partial-Slab[9] *erfüllt, wenn dieser vorhanden ist. Sonst wird die Anfrage von einem Empty-Slab erfüllt, der wenn er nicht vorhanden ist, erst erzeugt wird.*

Slabs können mit Hilfe des Slab-Allocator-Interfaces angesprochen werden.

Benutzt man anstelle von `kmalloc()` einen Slab-Allocator der auf die entsprechenden Slabs zugreift, so können bei häufig benutzten Datenstrukturen enorme Performancevorteile entstehen. Die angeforderte Datenstruktur muss, wenn eine freie, unbenutzte Datenstruktur in einem Partial-Slab bereits vorhanden ist, nicht neu erzeugt und damit erneut Speicher alloziert werden, sondern kann, ohne Zugriffe auf den Speicher erforderlich zu machen, übergeben werden.

Mit Hilfe eines Slabs können Datenstrukturen, die später mit hoher Wahrscheinlichkeit häufig wieder verwendet werden, effizient verwaltet werden ohne den Speicher für die Datenstruktur freigeben und später erneut, unter Umständen sehr zeitaufwendig, wieder allozieren zu müssen.

[9] Verwaltet belegte und freie Objekte.

5.2.4 Auswirkungen eines Plattformwechsels

Wie in dem Abschnitt 5.2.2 Bordmittel bereits beschrieben, hat die Wahl der Virtualisierungsarchitektur, einen großen Einfluss auf die Geschwindigkeit und Performance der virtuellen Maschine. In diesem Punkt sollen zusätzlich die Performanceunterschiede zwischen KVM und VMWare näher betrachtet werden. Mellor [Mel08] sagt, dass RedHat, die die Entwicklerfirma von KVM inzwischen gekauft und zu ihrer zentralen Virtualisierungslösung gemacht hat, für je drei virtuelle Maschinen die mit VMWare auf einem Host betrieben werden, fünf virtuelle Maschinen unter KVM auf dem selben Host laufen lassen können.

Durch diese Perfomance-Aussage motiviert, wird auch der Leistungsunterschied zwischen den beiden Virtualisierungslösungen KVM und VMWare Server untersucht. Im Gegensatz zu VMWare setzt KVM voll und ganz auf Hardware-Beschleunigungen. Der Virtualisierungslayer bei KVM ist daher mit den neuesten Techniken zur hardware-gestützten Virtualisierung, wie etwa Intel VT und AMD Pacifica, sowie Page Tables (Nested Page Tables - AMD, Extended Paging Tables - Intel) und IOMMU (I/O Memory Mapping Unit) vertraut[WR10].

Paravirtualisierung ist unter KVM nicht vorgesehen[10], da die hardware-gestützten Verfahren die Lücke zwischen der Virtualisierung und der nativen Ausführung auf dem Host besser schließen können. Der Vorteil der hardware-unterstützten Virtualisierung liegt nicht nur in den Geschwindigkeitsvorteilen und dem schlanken Virtualisierungslayer, sondern vielmehr in der Tatsache, dass das Gast-System keine Kenntnis davon besitzen muss, ob es direkt ausgeführt oder virtualisiert wird.

Abbildung 5.10 zeigt die Ausführungszeiten für das Kompilieren des Linux-Kernels mit der für diese Anwendung minimalen Konfigurationsdatei von 46 Kilobytes. Die Ausführungszeiten errechnen sich als Mittelwert aus jeweils drei Kompilierungsvorgängen. VMWare benötigt 47,0% mehr Zeit als die native Ausführung auf dem Host benötigt hat. KVM führt das Kompilieren mit 42% Overhead um insgesamt 5% schneller aus, als dies bei VMWare der Fall ist. Aus zeitlichen Gründen konnte ein gepatchtes KVM mit Paravirtualisierung nicht zum Vergleich zur Kompilierung mit VMWare und Paravirtualisierung gemessen werden. Deshalb wurde in der Grafik auf eine vergleichende Darstellung mit Paravirtualisierung verzichtet. Dennoch sei angemerkt, dass die zusätzlich Nutzung der Paravirtualisierung beim Neukompilieren des Kernels eine erhebliche Beschleunigung brachte. Anstelle der zuvor benötigten 563 Sekunden, brauchte VMWare mit Paravirtualisierung nur noch 466 Sekunden im Durchschnitt. Damit ist diese Variante nur noch 24,60 % schlechter als die Ausführung im Host.

[10] Sie kann aber mittels Patch eingespielt werden.

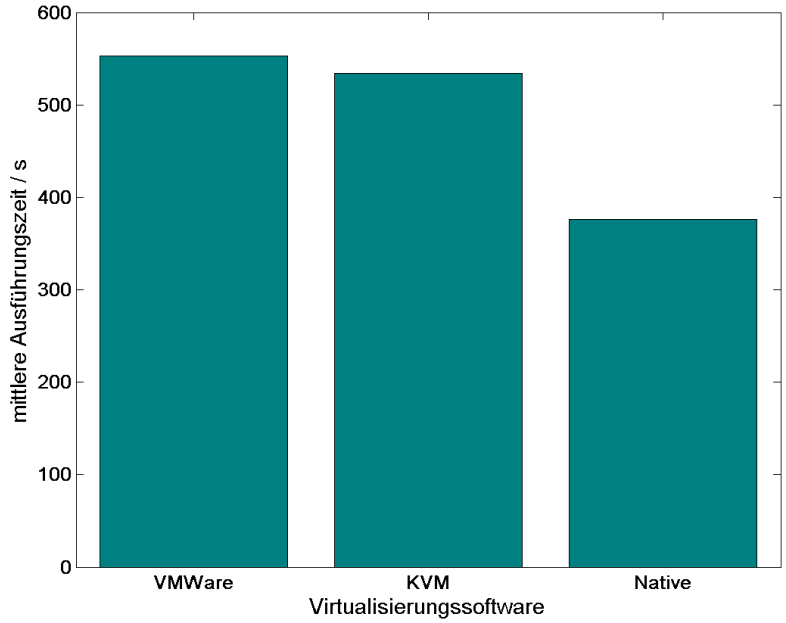

Abbildung 5.10: Vergleich der Ausführungszeiten von *Make* auf verschiedenen Virtualisierungsplattformen

5.3 Diskussion der Ergebnisse

Vergleicht man die Ergebnisse, so wird deutlich, dass die Maßschneiderung einen sehr großen Beitrag zur Performanceverbesserung bringt und exakt auf das Problem, die Anwendung zu beschleunigen, zugeschnitten ist.

Soll die Anwendung jedoch wirklich performanter werden, so reicht es nicht aus, nur das Beste aus den Funktionen des Betriebssystems herauszuholen. Vielmehr müssen bei virtuellen Maschinen die verschiedenen Abstraktionsebenen allesamt berücksichtigt werden. Durch die richtige Konfiguration des Betriebssystems auf die von der Anwendung benötigten Merkmale[11] und die von der virtuellen Maschine zur Verfügung gestellten Hardware, können viele Einsparungen, die sich positiv auf die Ausführungszeit der benötigten Systemaufrufe auswirken, erreicht werden. Gerade der Wechsel des I/O Scheduler hat bei der Anwendung lighttpd eine enorme Verbesserung des Antwortverhaltens hervorgerufen. Die durch die Maßschneiderung erzielten Verbesserungen hätten sich ohne diese Konfiguration niemals in einem Performancegewinn der Anwendung manifestiert.

Auch auf Ebene der Virtualisierung kann durch den Einsatz von KVM als Virtualisierungslösung anstelle von VMWare die Performance um 5% gesteigert wer-

[11] Server brauchen so gut wie nie Sound; Energie-Management ist bei Servern auch eher zweitrangig.

den. Obwohl KVM kein Hypervisor vom Typ 1 ist, befinden sich die KVM Treiber direkt im Linux-Kernel des Host-Systems (siehe Abbildung 5.11), während die Emulation der virtuellen Hardware mittels QEMU im User-Space stattfindet. Es wäre nicht ungewöhnlich, wenn die KVM-Treiber durch den direkten Zugriff auf Kernel-Strukturen daraus einen Vorteil für sich nutzbar machen könnten.

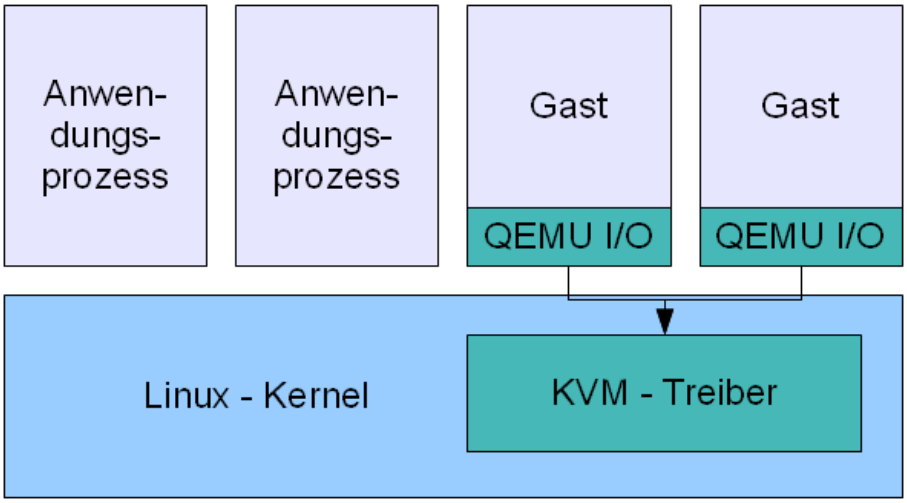

Abbildung 5.11: KVM - Architektur

KVM wird seit Beginn auf Performance optimiert und nutzt konsequent jede Form der Hardware-Unterstützung. Der Virtualisierungslayer dürfte daher eher schlank ausfallen.

Durch alle Verbesserungen konnte die Performance der Anwendung stark gesteigert und die Ausführung der relevanten Systemaufrufe um 14,3% verbessert werden (siehe Abbildung 3.5 in dem Abschnitt Analyse für den Ist-Zustand vorher; Abbildung 5.12 zeigt den Zustand nach den vorgeschlagenen Verbesserungen).

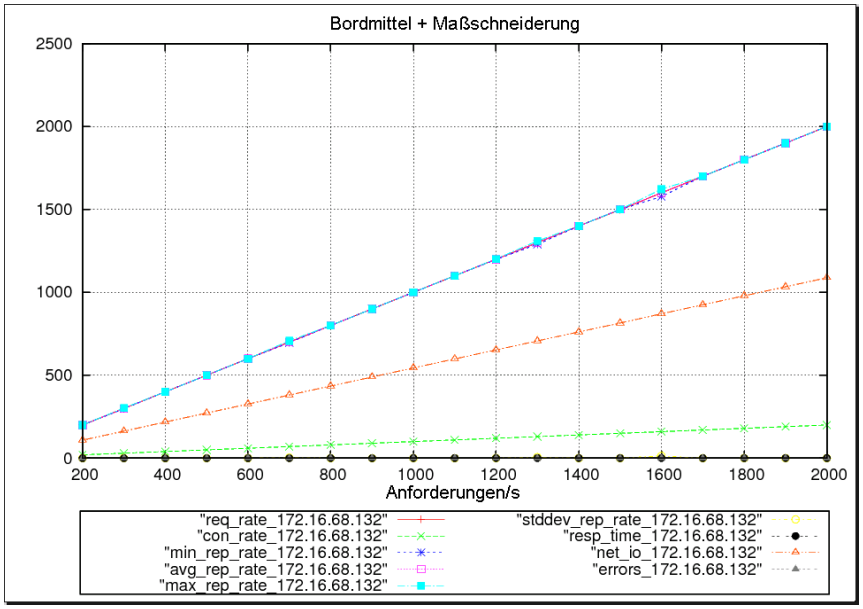

Abbildung 5.12: Ergebnisse des Autobench-Tests nach den Optimierungen

6 Zusammenfassung und Ausblick

Ein häufiger Fall in der Virtualisierung ist die Konsolidierung von einzelnen Rechnern, die bestimmte Server-Dienste, wie etwa Web- oder E-Mail-Server bereitstellen. Die so virtualisierten Rechner besitzen immer noch die gleiche Isolation voneinander, als ob diese noch auf ihrer alten Hardware ausgeführt würden. Durch die Bündelung mehrerer virtueller Maschinen auf einem leistungsfähigen Rechner, können die zur Verfügung stehenden Hardware-Ressourcen besser genutzt werden.

Die Analyse des Webservers lighttpd hat gezeigt, dass eine reine Maßschneiderung des Betriebssystems sehr schnell von anderen Faktoren im Umfeld der virtuellen Maschinen und der Konfiguration des Betriebssystems ausgebremst werden kann. Durch den Einsatz des passenden I/O Schedulers konnte ein Einbrechen der Antworten — aufgrund von zu großen I/O Blocking-Zeiten — verhindert werden. Clients des Webservers registrieren diesen Einbruch als nicht beantwortete Seitenabrufe. Nach der Verwendung des Deadline-Schedulers normalisierte sich dieses Verhalten und der Webserver lieferte auch unter hohen Stressbedingungen (2000 Anfragen pro Sekunde) alle Seiten innerhalb der geforderten Antwortzeit von 4 Sekunden.

Um eine Maßschneiderung auf eine Anwendung hin produktiv zu betreiben, ist also ein ganzheitlicher Ansatz nötig. Diese Diplomarbeit hat nach dieser gewonnenen Erkenntnis einen ganzheitlichen Ansatz gewählt und außer der Maßschneiderung, auch Optimierungen auf den Ebenen der Paketkonfiguration, der Kernel-Konfiguration und der Virtualisierung umgesetzt.

Die Optimierungen auf der Paketebene — also das Entfernen für die Anwendung nicht benötigter Dienste — stellt mit 0,04 % weniger CPU-Zyklen[1] einen kleinen Anteil der Gesamteinsparungen dar. Dafür bringt die Minimierung des Systemkernels durch eine erheblich angepasste Kernel-Konfiguration immerhin 10% mehr Leistung bei den von der Anwendung benötigten Systemaufrufen.

Die verschiedenen Maßschneiderungen des Linux-Kernels brachten in Summe einen Performance-Gewinn von 15,8 %. Einen großen Anteil hat hierbei die Entfernung der Verwaltung der Zugriffsrechte mit 10,5% und der Entfernung der

[1] Die virtuelle Maschine wurde von Anfang an als konsolenbasiertes System installiert. Bei einem grafischen System werden wesentlich mehr unnötige Pakete installiert. Die Ergebnisse fallen dann entsprechend höher aus.

Linux Security Modules mit 10,0% gebracht. Aber auch die Entfernung von nicht mehr benötigten überflüssigen Überprüfungen in einigen sehr häufig ausgeführten Funktionen brachten insgesamt einen Performance-Gewinn beim Stresstest von 10,7%. Der Bypass des „Virtuellen File Systems“ erlaubte zusätzlich, einige Tests zu entfernen und erreichte so eine Steigerung der Performance um 15,4%.

Für rein technische Optimierungen auf Code-Ebene, wie etwa Loop-Unrolling oder Inlining von Funktionen, kann der GCC genutzt werden. Eine Optimierung des kompletten Linux-Kernels durch den GCC führte nicht zu eingesparten Taktzyklen, sondern sorgte für eine teilweise erhebliche Steigerung der Taktzyklen. Die richtigen Compilerschalter (Flags) zur Optimierung zu finden, kann sich beim GCC jedoch als kleine Herausforderung gestalten. Häufig sind kaum Verbesserungen zwischen -O2 und -O3 erkennbar, Verbesserungen mit einer Anzahl zusätzlicher Schalter aber häufig effektiver (vgl. [Lad05]). Ist Time-to-value (eine kurze Zeit bis zur Wirtschaftlichkeit) bei der Optimierung nicht der dominierende Faktor, lohnt sich eventuell das Kompilieren des Linux-Kernels mit Hilfe des genetischen Algorithmus Acovea auf einer Hochgeschwindigkeitsmaschine, um ein Set pareto-optimaler Schalter für eine Code-Optimierung durch den GCC zu finden.

Betrachtet man die Vorteile durch die Nutzung der Paravirtualisierung des VMWare Servers, so benötigt VMWare nur noch 24,60% mehr Rechenzeit gegenüber der nativen Ausführung auf dem Host. Zuvor waren dies ohne Paravirtualisierung 47,0%.

Selbst die Auswahl der zugrundeliegenden Virtualisierungslösung spielt eine nicht unerhebliche Rolle. So ist KVM bei der Virtualisierung mit 42% Overhead gegenüber der nativen Ausführung immerhin ganze 5% schneller als VMWare. KVM bietet die Möglichkeit der Paravirtualisierung nur über einen nachträglichen Patch an und wurde nicht mit Patch getestet. Gleichwohl dürfte auch KVM mit dem Patch zur Paravirtualisierung weitere Performance-Vorteile gegenüber VMWare bringen. Aussagen von RedHat[2] legen zumindest nahe, dass KVM erheblich ressourcensparender zu sein scheint, als der bisherige Marktführer für Produktivlösungen VMWare.

Insgesamt konnte ein Performance-Gewinn von 95,7% erreicht werden.

Die Ergebnisse zeigen, dass viele Flaschenhälse außerhalb der virtuellen Maschine zu erheblichen Performanceverlusten geführt haben. Gerade die Wahl des I/O Schedulers macht deutlich, wie wichtig eine ganzheitliche Betrachtung ist, wenn man die Performance der Anwendung steigern will.

Im Sinne einer kurzen Time-to-value kann die hier vorgestellte Methode schnell und effizient viele Flaschenhälse auf allen Ebenen finden und ermöglicht durch die Beseitigung von Flaschenhälsen, die im Betriebssystem vorgenommenen Ein-

[2] http://www.channelregister.co.uk/2008/09/10/run_more_vms_with_red_hat_than_vmware/

sparungsmöglichkeiten wirklich durch performanteres Verhalten zu nutzen. Die Anwendung profitiert somit nicht nur theoretisch von einer Einsparung an CPU-Zyklen, die innerhalb des Betriebssystems der VM stattfindet.

KSplice als Teil der hier verwendeten Methode unterstützt das Wiederverwenden von optimiertem Code, sowie das schnellere Testen des Linux-Kernels, indem Änderungen als „Hot Patch“ in die Binär-Dateien des Kernels eingebunden werden. Die aufwendige Prozedur des Kernelkompilierens und -installierens mit anschließendem Neustart kann so effektiv verkürzt werden. Die Messungen konnten so flexibler vorbereitet werden. „Alles“ an Verbesserungen zu messen, bedeutet nur alle Verbesserungen mittels KSplice in den Kernel einzubringen.

Viele Anwendungen lassen sich direkt einer der beiden Gruppen, CPU-Bound (viele Berechnungen von der CPU zu leisten) oder I/O-Bound (viele Input-/Output-Operationen, wie zum Beispiel Datei-Zugriffe), zuordnen. Meist profitieren Anwendungen der gleichen Gruppe dann auch von gleichen Optimierungen. Ist das Entfernen von Zugriffsrechten bei `open()` / `close` Systemaufrufen (system calls) bereits als Patch vorhanden, kann es unter Umständen mit wenig oder keinen Anpassungen[3] bei einer anderen virtuellen Maschine eingesetzt werden. KSplice ermöglicht die Anwendung von Patches auf bereits gepatchte Kernel, sofern die bereits gepatchte Kernel-Source zum Vergleich vorliegt.

Bisher umfasst die hier dargestellte ganzheitlichen Methode nicht die Ebene der Hardwareoptimierung, in diesem Fall die Optimierung des benutzten Codes auf eine entsprechende CPU hin. Wichtig ist dieser Teil vor allem, da wie im Abschnitt 2.1.1 erwähnt wurde, die CPU-Instruktionen der virtuellen Maschine direkt auf der CPU ausgeführt werden. Können also Systemaufrufe in der virtuellen Maschine per Hardware, durch die Nutzung spezieller Befehlsätze, beschleunigt werden, so wäre dies ein enormer Vorteil. Ein Beispiel wäre hierfür die Nutzung des Intel Befehlssatzes SSE4.2, der zum Beispiel bei String-Operationen eine höhere, da Hardware unterstützte, Performance bietet[Int09]. Zu klären bliebe weiterhin, ob diese Anpassung manuell erfolgen muss, oder bereits bei der Optimierung des GCC auf eine entsprechende Architektur automatisch erfolgt. Nachteil dieser Ebene der Anpassungen wäre, dass die Anpassungen damit nicht nur architekturabhängig (zum Beispiel x86), sondern zusätzlich auch noch von der CPU beziehungsweise den Befehlssatzerweiterungen (zum Beispiel SSE4.2) abhängig wären. Die Migration der virtuellen Maschine auf eine andere Hardware würde damit erschwert. Zusätzlich würde entgegen dem Trend, mit virtuellen Maschinen Komplexität zu entfernen, hiermit wieder welche hinzugefügt. Für sehr performancekritische Anwendungen wäre dies jedoch sicherlich eine Überlegung wert.

Die Spezialisierung einzelner Funktionen war nicht Teil dieser Arbeit. Dennoch konnten, wie etwa bei `writev()` einige interessante Stellen gefunden werden, an

[3] Abhängig unter anderem von der Kernel-Version.

denen ein dynamischer Laufzeitoptimierer, wie beispielsweise Tempo, eingesetzt werden kann. Im Sinne einer ganzheitlichen Methode, die zudem eine kurze Time-to-value ermöglicht, wäre die Nutzung dieses Automatismus wünschenswert gewesen. Ein Einsatztest von Tempo verlief jedoch negativ. Tempo konnte aufgrund der Inkompaktibilität mit lauffähigen Version[4] von Standard ML of New Jersey (sml-nj) nicht benutzt werden. Die Korrespondenz mit dem Hersteller von Tempo ergab, dass unter Umständen eine aktualisierte Fassung des Spezialisiers angefertigt wird, die auch unter heutigen ML Umgebungen läuft. Leider wurde bisher keine neue Tempoversion angeboten, die die Spezialisierung der Quellcodes hätte vornehmen können. Sollte sich dies in Zukunft ändern und wieder eine lauffähige Version von Tempo zur Verfügung stehen, so sollte Tempo in diese Methode aufgenommen werden, um die Anpassung des Betriebssystems auf eine Anwendung mit Spezialisierung weiter zu unterstützen.

Obwohl nicht jede mögliche Kombination jeweils mit KVM und VMWare getestet wurde, so ist doch anhand der Eigenschaften und genutzten Virtualisierungsarchitekturen erkennbar, dass KVM einige Vorteile besitzt. KVM ist mit vielen anderen Projekten verzahnt und es findet ein stetiger Gedanken- und Codeaustausch statt. KVM legt zusätzlich Wert auf die Einbindung der neuesten Hardwarebeschleunigungen. Nicht nur Intels VT oder AMDs Pacifica, sondern auch relativ neue Features wie IOMMU werden von KVM genutzt. Nachteile wie fehlende Managementsoftware werden schnell durch Pakete wie libvirt und virsh geschlossen.

Bei VMWare liegt die Entscheidung, beim VMWare Server Hardwareunterstützung nicht zu nutzen, auf politischer Ebene. Der kostenlose VMWare Server soll nur den Einstieg in die Welt der Virtualisierungen darstellen. Management Software, sowie der VMWare ESX Server, der als Hypervisor Typ direkt auf der Hardware sitzt und daher wesentlich performanter ist, sind bei VMWare kostenpflichtig. Das Virtual Machine Interface (VMI) ist seit Kernelversion 2.6.21 Teil des Linux-Kernels.

Eine interessante Fragestellung, der in dieser Arbeit aus Zeitgründen nicht mehr nachgegangen werden konnte, ist, in welchem Maße Maßschneiderungspotenziale bei Virtual Appliances vorhanden sind. Der hier betrachtete Fall, einen Webserver als alleinigen Dienst in einer virtuellen Maschine laufen zu lassen, entspricht der in Firmen sehr häufig angewandten Serverkonsolidierung. Rechner, die ohnehin aus Sicherheits- und Isolationsgründen nur einen Dienst, wie etwa einen Web- oder Mailserver betreiben, werden aus Kostengründen inzwischen in virtuelle Maschinen umgewandelt. Hierdurch kann die vorhandene Hardware besser ausgenutzt werden. Virtualisierung hat die IT in Firmen jedoch in vielen Bereichen verändert. So sind viele Standardtestszenarien oder Standardentwicklungsumgebungen

[4] Obwohl Version 110.73 aktuell ist, wurden Versionen der Standard ML of New Jersey bis 110.42 (Version von 2003) ausprobiert. Es konnte keine funktionierende Konfiguration gefunden werden.

bereits als fertige Virtual Appliances[5] zu bekommen. Diese haben teilweise eine etwas größere Komplexität, da häufig mehrere verschiedene Programme ausgeführt werden[6].

Eine Betrachtung, welche Auswirkungen eine Aktualisierung der virtuellen Hardware hat, konnte aus Kompatibiliätsgründen zu KVM nicht berücksichtigt werden. Der Testaufbau beinhaltete exakt eine vmdk-Datei, welche das VMWare-Format für die virtuelle Behälterdatei darstellt. Diese kann sowohl unter VMWare, als auch unter KVM ausgeführt werden. KVM kann neuere Hardware-Versionen von VMWare jedoch nicht ohne weiteres ausführen, so dass mehrere Versionen der zu testenden virtuellen Maschine entstanden und eine Vergleichbarkeit nicht unbedingt mehr gegeben war. Durch die Möglichkeit des grub, beim Start des Rechners den Boot-Eintrag zu editieren, konnte KVM die Festplatte passend nach /dev/hda mounten[7].

Die Fragestellung, inwiefern eine manuell geführte und durch Fachwissen geprägte Kernelkonfigurationen (zum Beispiel nach [KH06]) Vorteile oder Nachteile zu einer vollautomatisierten Lösung (SARCHECK) mit eingebautem Profiler hat, ist nicht Bestandteil dieser Diplomarbeit. Dennoch erscheint es gerade im Hinblick auf die von McNamee [MWP+01] erwähnten Verbreitungsschwierigkeiten außerhalb der Wissenschaft, die er bei der Spezialisierung festgestellt und die auch für die Maßschneiderung im allgemeinen gelten, eine interessante Fragestellung zu sein, wie gut diese kommerziellen vollautomatischen Lösungen sein können.

[5] Siehe: VMWare Marketplace — http://www.vmware.com/appliances/

[6] Beispielsweise eine komplette Entwicklungsumgebung für PHP-Skripte, inklusive eines Apache-Servers mit MySQL und PHP-Unterstützung, sowie passenden Editoren zur Webseiten-Entwicklung.

[7] VMWare benutzt SCSI-Treiber und mountet daher nach /dev/sda.

Literaturverzeichnis

[Ach06] Achilles, Albrecht: *Betriebssysteme - Eine kompakte Einführung mit Linux*. 2006

[Ahn09] Ahnert, Sven: *Virtuelle Maschinen mit VMWare und Microsoft*. München [u.a.] : Addison-Wesley, 2009

[AK09] Arnold, Jeff ; Kaashoek, M. F.: *KSplice: Automatic Rebootless Kernel Updates*. Version: April 2009. http://www.ksplice.com/doc/ksplice.pdf, Abruf: 29.07.10

[BC06] Bovet, Daniel P. ; Cesati, Marco: *Understanding the Linux Kernel*. 3. Auflage. Beijing [u.a.] : O'Reilly, 2006

[CH07] Chinni, Shefali ; Hiremane, Radhakrishna: Virtual Machine Device Queues - An Integral Part of Intel® Virtualization Technology for Connectivity that Delivers Enhanced Network Performance. (2007). http://software.intel.com/file/1919

[Coh04] Cohen, William: Tuning Programs with OProfile. In: *WIDE OPEN MAGAZINE* (2004), S. 53–62

[CSB+03] Chanet, Dominique ; Sutter, Bjorn D. ; Bosschere, Koen D. ; De, Bjorn ; Koen, Sutter ; Bosschere, De: *Link-time Optimization of a Linux Kernel for Space*. 2003

[CSB+05] Chanet, Dominique ; Sutter, Bjorn D. ; Bus, Bruno D. ; Put, Ludo V. ; Bosschere, Koen D.: System-wide Compaction and Specialization of the Linux Kernel. (2005)

[DC09] Domingo, Don ; Cohen, William: *SystemTap Beginners Guide*. Version: 2009. http://sourceware.org/systemtap/SystemTap_Beginners_Guide.pdf, Abruf: 29.07.2010

[Eig10] Eigler, Frank C.: *Systemtap tutorial*. Version: März 2010. http://sourceware.org/systemtap/tutorial.pdf, Abruf: 29.07.2010

[Gou04] Gough, Brian J.: *An Introduction to GCC - for the GNU compilers gcc and g++*. Bristol : Network Theory, 2004

[Int09] Intel ; Intel (Hrsg.): *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. Intel, November 2009. http://www.intel.com/Assets/PDF/manual/248966.pdf

[KH06] Kroah-Hartman, Greg: *Linux kernel in a nutshell*. O'Reilly, 2006 http://www.kroah.com/lkn/

[KMPP07] Keniston, Jim ; Mavinakayanahalli, Ananth ; Panchamukhi, Prasanna ; Prasad, Vara: Ptrace, Utrace, Uprobes: Lightweight, Dynamic Tracing of User Apps. In: *Linux Symposium* One (2007)

[KPH10] Keniston, Jim ; Panchamukhi, Prasanna S. ; Hiramatsu, Masami: *Kernel Probes*. Version: Juni 2010. http://www.kernel.org/doc/Documentation/kprobes.txt

[Ksp10] Ksplice, Inc.: *KSplice Uptrack*. Version: Juni 2010. http://www.ksplice.com/pricing

[Lad05] Ladd, Scott R.: *Acovea - Using Natural Selection to Investigate Software Complexities*. Version: 2005. http://www.coyotegulch.com/products/acovea/, Abruf: 04.07.2010

[LE88] Lister, A.M. ; Eager, R.D.: *Fundamentals of Operating Systems*. 4. Auflage. 1988

[LK05] Love, Robert ; Keller, Erik: *Linux-Kernel-Handbuch: Leitfaden zu Design und Implementierung von Kernel 2.6*. München : Addison-Wesley, 2005

[Mae03] Maeda, Toshiyuki: Kernel Mode Linux. In: *Linux Journal* (2003), Nr. 109

[Mae10] Maeda, Toshiyuki: *Kernel Mode Linux Implementierung*. Version: Mai 2010. http://web.yl.is.s.u-tokyo.ac.jp/~tosh/kml/, Abruf: 29. Juli 2010

[Mel08] Mellor, Chris: *Red Hat sprints past ESX on VM running*. Version: September 2008. http://www.channelregister.co.uk/2008/09/10/run_more_vms_with_red_hat_than_vmware/, Abruf: 29.07.2010

[Mes00] Messmer, Hans P.: *PC Hardwarebuch - Aufbau, Funktionsweise, Programmierung*. 6. Auflage. München : Addison-Wesley, 2000

[MJ98] Mosberger, David ; Jin, Tai: httperf - A Tool for Measuring Web Server Performance. In: *Performance Evaluation Review* 26 (1998), S. 31–37

[MWK05] M.Youseff, Lamia ; Wolski, Rich ; Krintz, Chandra: Linux Kernel Spezialization for Scientific Application Performance. In: *UCSB Technical Report* (2005)

[MWP+01] McNamee, Dylan ; Walpole, Jonathan ; Pu, Calton ; Cowan, Crispin ; Krasic, Charles ; Goel, Ashvin ; Wagle, Perry: Speciali-

zation Tools and Techniques for Systematic Optimization of System Software. In: *ACM Transactions on Computer Systems* 19 (2001), Mai, Nr. 2, S. 217–251

[MY03] Maeda, Toshiyuki ; Yonezawa, Akinori: Toward an Operating System Protected by a Type Theory. In: *Lecture Notes in Computer Science* 2896 (2003), S. 3–17

[NSL+06] Neiger, Gil ; Santoni, Amy ; Leung, Felix ; Rodgers, Dion ; Uhlig, Rich: *Intel(r) Virtualization Technology: Hardware Support for Efficient Processor Virtualization.* Version: August 2006. http://download.intel.com/technology/itj/2006/v10i3/v10-i3-art01.pdf, Abruf: 29.07.2010. Intel Technology Journal

[PG74] Popek, Gerald J. ; Goldberg, Robert P.: Formal Requirements for Virtualizable Third Generation Architectures. In: *Communications of the ACM* 17 (1974), S. 412–421

[SGG05] Silberschatz, Abraham ; Gagne, Greg ; Galvin, Peter B.: *Operating System Concepts.* 7. Auflage. 2005

[Spi08a] Spinczyk, Olaf: *Betriebssysteme, Rechnernetze und verteilte Systeme 1 (BSRvS1).* Version: 2008. http://ess.cs.tu-dortmund.de/Teaching/SS2008/BSRvS1/Downloads/14-Zusammenfassung-Ausblick-1x2.pdf, Abruf: 29.07.2010

[Spi08b] Spinczyk, Olaf: *Betriebssystemtechnik - Zusammenfassung und Ausblick.* Version: 2008. http://ess.cs.uni-dortmund.de/Teaching/WS2008/BST/Downloads/16-Ausblick.pdf, Abruf: 29.07.2010

[VMW08] VMWare: *Performance of VMI.* Version: 2008. http://www.vmware.com/pdf/VMware_VMI_performance.pdf, Abruf: 29.07.2010

[VMW09] VMWare: *Performance Evaluation of VMXNET3 Virtual Network Device.* Version: 2009. http://www.vmware.com/pdf/vsp_4_vmxnet3_perf.pdf, Abruf: 10.07.2010

[WCS+02] Wright, Chris ; Cowan, Crispin ; Smalley, Stephen ; Morris, James ; Kroah-Hartman, Greg: Linux Security Modules. (2002)

[Wie10] Wiehr, Hartmut: *Angriff auf VMware - Microsoft und Citrix im Nacken.* Version: April 2010. http://www.cio.de/knowledgecenter/server/2232177/, Abruf: 29.07.2010

[WR10] Warnke, Robert ; Ritzau, Thomas: *qemu-kvm & libvirt.* Dt. Orig.-Ausg., 4. Aufl. Books on Demand, 2010 Http://quemu-buch.de

Abbildungsverzeichnis

Tabellenverzeichnis