Fachhochschule Köln
University of Applied Sciences Cologne
Campus Gummersbach

Diplomarbeit

Servervirtualisierung unter Linux

Analyse und Bewertung

Prüfer:

Prof. Dr. H. L. Stahl
Dipl. Ing. Sebastian Wenner

Vorgelegt von:

Björn Groß-Hohnacker
Technische Informatik
Mat. Nr.: 11029835

Vorgelegt am:

25. August 2006

Inhaltsverzeichnis

1 Einleitung **9**

1.1 Was ist Virtualisierung 9

1.2 Zielsetzung der Diplomarbeit 10

1.3 Grundlagen von Virtualisierungsverfahren 11

1.3.1 Geschichte der Virtualisierung 11

1.3.2 Aufbau eines Virtualisierungssystems 13

1.3.3 Anwendungsgebiete 13

1.3.3.1 Serverkonsolidierung 14

1.3.3.2 Sicherheit und Fehlerisolierung 14

1.3.3.3 Multi-Tier-Umgebungen 15

1.3.3.4 Live Migration und Load Balancing 16

1.3.3.5 Hochverfügbarkeit 17

1.3.3.6 Administrierbarkeit und Wartbarkeit 18

1.3.3.7 Verlässlichkeit und Verfügbarkeit 19

1.3.3.8 Virtualisierung und IT-Sicherheit 19

1.4 Funktionsweise unterschiedlicher Virtualisierungsansätze 21

1.4.1 Kapselung von einzelnen Prozessen 21

1.4.2 Virtualisierung auf Betriebssystemebene 23

1.4.3 Virtualisierung auf Systemebene 24

1.4.4 Virtualisierung durch Emulation 25

1.4.5 Paravirtualisierung 27
1.4.6 Das IA-32 Ringkonzept 28
1.4.7 Hardwarevirtualisierung 31
1.5 Freie und kommerzielle Virtualisierungslösungen 33
1.6 Virtualisierungssysteme im Überblick 35
1.7 Eingrenzung des Themenbereichs der Diplomarbeit 36

2 Analyse ausgewählter Virtualisierungsverfahren 37
2.1 OpenVZ 38
2.1.1 Implementierung und Funktionsweise 38
2.1.2 OpenVZ VPS-Management 40
2.1.3 Ressourcenmanagement 41
2.1.3.1 CPU-Management 42
2.1.3.2 Festplattenmanagement 44
2.1.3.3 IPC-Ressourcenmanagement 45
2.1.3.4 Speichermanagement 47
2.1.3.5 Netzwerkvirtualisierung 49
2.1.4 Installation eines OpenVZ-VPS 50
2.1.5 Spezielle Funktionen 52
2.1.5.1 Checkpointing 52
2.1.5.2 Live Migration 53
2.1.6 Virtuozzo 53
2.2 Xen 55
2.2.1 Implementierung und Funktionsweise 55
2.2.1.1 Hardwarevirtualisierung mit Xen 56
2.2.1.2 Hardwareunterstützung unter Xen 56
2.2.2 Das Xen-Domänenkonzept 57
2.2.2.1 Domäne 0 58

2.2.2.2 Domäne U 59
2.2.3 Xen-Domänenmanagement 60
2.2.3.1 Xen Control Daemon 60
2.2.3.2 Xen Management User Interface 61
2.2.3.3 Virtuelle Gerätearchitektur 62
2.2.4 Ressourcenmanagement 63
2.2.4.1 CPU-Management 63
2.2.4.2 Festplattenmanagement 65
2.2.4.3 Speichermanagement 65
2.2.4.4 Netzwerkvirtualisierung 66
2.2.5 Installation einer Domäne U 67
2.2.6 Spezielle Funktionen 69
2.2.7 Erweiterungen für Xen 70

3 Exemplarische Implementierung und Benchmarks 72

3.1 Grundlagen der Implementierung 73
3.1.0.1 Der Logical Volume Manager (LVM) 75
3.2 Die OpenVZ-Testumgebung 78
3.2.1 Installation der OpenVZ-Umgebung 78
3.2.1.1 Einrichtung OpenVZ-Host 78
3.2.1.2 Einrichtung OpenVZ-Gast 81
3.2.2 Inbetriebnahme 82
3.3 Die Xen-Testumgebung 83
3.3.1 Installation der Xen-Umgebung 83
3.3.1.1 Einrichtung Domäne 0 83
3.3.1.2 Einrichtung Domäne U 86
3.3.2 Inbetriebnahme 89
3.4 Benchmarks 90

3.4.1 Unixbench 90
3.4.1.1 Unixbench Benchmarkergebnisse 91
3.4.2 Siege 94
3.4.2.1 Siege Benchmarkergebnisse 94

4 Bewertung und Fazit **96**
4.1 Bewertung 96
4.2 Fazit 98

A Literaturverzeichnis **100**

B Erklärung **103**

Abbildungsverzeichnis

1.1 Change Root Umgebung 22
1.2 Virtualisierung auf Betriebssystemebene 24
1.3 Virtualisierung auf Systemebene 26
1.4 IA-32 Ring Konzept 29
1.5 Virtualisierung auf Prozessorebene 31

2.1 Xen-Domänenkonzept 58

3.1 Logical Volume Management 76
3.2 CPU Benchmarks 92
3.3 IPC Benchmarks 93
3.4 Dateisystem Benchmarks 94

Tabellenverzeichnis

1.1 Vergleich von Virtualisierungsverfahren 35

3.1 Netzwerk-Benchmarks 95

Abkürzungsverzeichnis

AMD SVM	AMD Secure Virtual Machine
chroot	change root
CPU	Central Processing Unit
GPL	GNU General Public License
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IA-32	Intel Architecture 32
Intel VT	Intel Virtualization Technology
ID	Identifikationsnummer
IP	Internet Protocol
IT	Informationstechnik
PC	Personal Computer
RPM	Red Hat Package Manager
TCO	Total Cost of Ownership
VM	Virtual Machine
VMM	Virtual Machine Monitor
VMX	Virtual Machine Execution
VPS	Virtual Private Server
yum	Yellowdog Updater Modified

1 Einleitung

"Virtualisierung ist eine Herangehensweise in der IT, die Ressourcen so zusammenfasst und verteilt, dass ihre Auslastung optimiert wird und automatisch Anforderungen zur Verfügung steht."[1]

1.1 Was ist Virtualisierung

Virtualisierung in der Informationstechnik (IT) ist ein Verfahren, bei dem die physikalischen Hardwareressourcen eines Computersystems, wie Prozessor, Speicher und Netzwerk, auf eine oder mehrere virtuelle Umgebungen verteilt werden, um die vorhandenen Ressourcen optimal auszunutzen. Eine virtuelle Umgebung verhält sich wie ein reales System. Virtualisierung stellt also eine Abstraktionsebene zwischen der vorhandenen Hardware und dem Betriebssystem dar, um mehrere voneinander getrennte Betriebssystemumgebungen auf einer Hardware parallel zu betreiben.

[1] Quelle: [10], Virtualisierung: Rechenleistung aus dem großen Topf

1.2 Zielsetzung der Diplomarbeit

In dieser Diplomarbeit soll, nach einer Einführung in den Themenbereich *Virtualisierung*, zunächst ein Überblick über die Funktionsweise von Virtualisierungsverfahren gegeben werden. Im Folgenden wird anhand einer Analyse einiger ausgewählter Open Source Verfahren zur Virtualisierung unter Linux, mit anschließender exemplarischer Implementierung, eine Bewertung dieser Verfahren durchgeführt werden. Die Implementierung soll es ermöglichen, einen Vergleich der Leistung der einzelnen Systeme mit Hilfe einer Durchführung von Benchmarks anzustellen. Das Ziel ist es zu zeigen, welche Form der Virtualisierung in einer Linux-Serverumgebung sich für welche Anforderungen in der IT am besten eignet.

1.3 Grundlagen von Virtualisierungsverfahren

Es gibt diverse Ansätze, ein System oder eine Umgebung zu virtualisieren. Computersysteme bilden bei dem Begriff der Virtualisierung nur einen Teilbereich. Grundsätzlich lässt sich die Virtualisierung eines Systems als eine Abbildung einer Umgebung verstehen, welche so weit abstrahiert ist, dass sie immer noch die erforderlichen Eigenschaften der realen Umgebung vorweist, ohne eine exakte Kopie dieser zu sein. Im Folgenden wird speziell auf die Virtualisierung von Betriebssystemumgebungen von Serversystemen unter Linux eingegangen werden.

1.3.1 Geschichte der Virtualisierung

Virtualisierung von Computersystemen ist kein neuer Gedanke. Schon in den 70er Jahren wurde von G. Popek und R. Goldberg in dem Auszug *„Formal requirements for virtualizable third generation architectures“*[13] genau aufgezeigt, welche Vorraussetzungen eine Rechnerarchitektur erfüllen muss, um virtualisierbar zu sein. Schon vorher, Mitte der 60er Jahre, hatte IBM bereits die erste virtuelle Maschine *VM/370* entwickelt, die auf den Modellen der Serie 370 zum Einsatz kam, um mehreren Benutzern virtuelle eigenständige IBM System/370 Betriebssysteme zur Verfügung zu stellen.[2]

Heute wird die Weiterentwicklung dieses Betriebssystems unter dem Namen *z/VM* in Systemen der IBM Z-Series eingesetzt, um zum Beispiel mehrere vollständig voneinander getrennte Linux-Installationen auf einem Großrechnersystem in virtuellen Maschinen zu betreiben. Des weiteren kann ein Großrechner durch Hardwarevirtualisierung in mehrere so genannte *Logical*

[2]Vgl. [6], The Origin of the VM/370 Time-Sharing System

Partition (LPAR) aufgeteilt werden, um voneinander unabhängige virtuelle Systeme zu erhalten, in denen dann jeweils ein Betriebssystem wie Linux oder auch wieder z/VM installiert werden kann.

Doch erst in den letzten Jahren wurden Computersysteme so weit entwickelt, dass es sich nicht mehr nur auf Großrechnern lohnt, Virtualisierung zu betreiben. Moderne Computersysteme nutzen häufig nur einen Bruchteil ihrer vorhandenen Ressourcen und werden trotzdem immer leistungsstärker. Während noch vor einigen Jahren eher ein Trend zur Dezentralisierung von Serverumgebungen zu beobachten war, da immer komplexer werdende Systeme, aus Gründen der Adminstrierbarkeit und Stabilität, auf mehrere Rechner verteilt wurden, bieten virtualisierte Systeme mit Hilfe von Zentralisierung, eine effizientere Ausnutzung der vorhandenen Ressourcen in modernen Rechnersystemen.

Ein großes Problem in heutigen Rechenzentren ist es, zum einen genug Platz für die immer größer werdende Anzahl von Systemen bereitzustellen und zum anderen die vorhandenen Systeme wirtschaftlich im Bezug auf entstehende Kosten für Strom, Wartung und weitere Bereiche auszunutzen. Virtualisierung kann dazu dienen, auf einer physikalischen Hardware mehrere Dienste oder auch ganze Betriebssysteme vollständig voneinander getrennt parallel zu betreiben und somit das Leistungspotenzial moderner *Central Processing Unit's* (CPUs)[3] weiter auszuschöpfen.

[3]engl. zentrale Recheneinheit eines (Mikro)prozessors

1.3.2 Aufbau eines Virtualisierungssystems

Ein *Virtualisierungssystem* in einer Rechnerumgebung besteht grundsätzlich aus zwei Hauptkomponenten, dem Host[4], der die Verwaltungsinstanz darstellt, und seinen Gästen, die in so genannten virtuellen Maschinen laufen.

Eine *virtuelle Maschine* bezeichnet in diesem Zusammenhang eine Laufzeitumgebung, in der Prozesse isoliert von denen anderer virtueller Maschinen betrieben werden.

Der *Host* verwaltet in erster Linie die physikalische Hardware und stellt seinen Gästen vorhandene Ressourcen zur Verfügung. Je nach Virtualisierungslösung kann er Ressourcenmanagement vornehmen und ist auch für die Verwaltung der Gäste zuständig.

1.3.3 Anwendungsgebiete

Das Problem der überdimensionierten Rechnersysteme wird speziell in großen Serverlandschaften deutlich. Die Betriebskosten jedes Rechners verteilen sich auf die Anschaffungs- oder Leasingkosten und die Unterhaltskosten für zum Beispiel Strom und Wartung. Es ist nicht immer ohne weiteres möglich einen einzelnen Server so zu dimensionieren, dass alle vorhandenen Ressourcen optimal genutzt werden. Daraus folgt, dass viele Ressourcen in einer Serverlandschaft ungenutzt bleiben. Die Vorteile, die man durch Virtualisierung von einzelnen Diensten oder auch eines ganzen Betriebssystems in Serverumgebungen erhalten kann, sind daher vielfältig.

[4] engl. Wirt, Gastgeber

1.3.3.1 Serverkonsolidierung

Einer der am häufigsten genannten Vorteile im Zusammenhang mit Virtualisierung ist die Möglichkeit der *Konsolidierung*, die in der IT den Prozess der Vereinheitlichung und Zusammenführung oder Verschmelzung von Systemen in Serverlandschaften bezeichnet. Das Ziel ist es, möglichst viele Serverdienste in eine einzelne physikalische Maschine zu bringen und diese dort mit der optimalen Ausnutzung der vorhandenen Rechenleistung zu betreiben. Moderne Serversysteme in Rechenzentren sind zumeist für ihre Aufgaben überdimensioniert und verschenken daher einen großen Teil ihrer Ressourcen.

Virtualisierung kann dazu beitragen, die Leistung einer einzelnen physikalischen Maschine optimal auszunutzen. Durch eine Konsolidierung erhält man häufig eine Senkung der *Total Cost of Ownership* (TCO). Das heißt, die laufenden Kosten für Hardware, Strom, Software, Reparatur und Wartung können gesenkt werden. Ein Nachteil, der sich aus der Konsolidierung jedoch ergibt, ist die Entstehung eines so genannten *Single Point of Failure*, also einer Stelle im System, die bei Problemen die gesamte virtualisierte Umgebung beeinträchtigen wird, da alle Gäste vom fehlerfreien Betrieb des Hosts abhängig sind. Möglichkeiten dem entgegen zu wirken werden unter Kapitel 1.3.3.7 beschrieben.

1.3.3.2 Sicherheit und Fehlerisolierung

Die Grundvorraussetzung einer virtuellen Umgebung ist es, dass alle virtuellen Maschinen vollständig isoliert voneinander laufen. Dies ist in erster Linie nötig, damit eventuell auftretende Probleme in einem Gast keine Auswirkung auf den Rest der vorhandenen Gäste haben. Es resultiert daraus aber auch der

Vorteil, dass mehrere kritische Prozesse, wie zum Beispiel öffentliche Web- und Filetransferserver, gleichzeitig auf dem selben physikalischen Server laufen können, ohne dass ein Einbruch in einen dieser Prozesse dazu führen könnte, die gesamte virtuelle Umgebung zu kompromittieren.

Wenn es also gelingt, über eine Sicherheitslücke in einem Prozess unberechtigten Zugriff auf eine virtuelle Maschine zu erlangen, sind die anderen virtuellen Maschinen trotzdem geschützt. Nahezu alle Virtualisierungssysteme erfüllen diese Anforderung einer Isolierung der einzelnen Gäste voneinander. Diese Trennung wird von den jeweiligen Virtualisierungsansätzen jedoch meist unterschiedlich realisiert und kann teilweise auch nur eingeschränkt garantiert sein.

Um diese Sicherheit und Fehlerisolierung zu gewährleisten, ist es jedoch wichtig die Hostumgebung mit Bedacht aufzubauen und abzusichern, da von dieser das komplette Management der Gäste durchgeführt wird. Es muss besonders darauf geachtet werden, dass ein unbefugter Zugriff auf die Hostumgebung weitestgehend verhindert wird, indem beispielsweise nicht benötigte Dienste deaktiviert werden. Im Falle eines Einbruchs auf den Host wäre sonst das gesamte Virtualisierungssystem mit allen Gästen kompromittiert.

1.3.3.3 Multi-Tier-Umgebungen

Durch Virtualisierung ist es möglich, ein komplexes System auf eine *Multi-Tier-Umgebung*[5] aufzuteilen. Dies bedeutet, dass einzelne Dienste dieses Systems auf mehrere eigenständige Umgebungen verteilt werden, ohne zusätzliche Hardware zur Verfügung stellen zu müssen. So kann ein einzelner physikalischer Server, auf dem mehrere kritische Dienste wie Webserver, Applikationsserver und Datenbank zusammen parallel laufen, auf mehrere virtuelle Umgebungen verteilt werden.

[5] engl. mehrstufige Umgebung

Dies hilft die Stabilität der einzelnen Systeme zu erhöhen, da Probleme in einem Bereich eines gesamten Systems so zunächst keine Auswirkung auf andere Bereiche hat.

Des weiteren gibt es die Möglichkeit ein physikalisches oder auch virtuell vorhandenes, produktives System in eine neue virtuelle Maschine zu kopieren und diese dann zum Beispiel als Test- und Entwicklungsumgebung zu nutzen. Dies ermöglicht es geplante Veränderungen wie ein Update eines Systems im Voraus ausgiebig in einer Umgebung zu testen, die ein genaues Abbild der produktiven Umgebung ist, bevor die Veränderungen in das Produktivsystem übernommen werden. Das schließt sowohl den Test von Änderungen einzelner Dienste, als auch Veränderungen am gesamten Betriebssystem ein. Dadurch können die Kosten für eine zusätzliche Testmaschine um ein Vielfaches reduziert werden. Die Implementierung von Multi-Tier-Umgebungen ermöglicht es also für die produktiven Dienste mit relativ geringem Aufwand die *Quality of Service*[6](QOS) zu erhöhen, da die allgemeine Stabilität einer komplexen Umgebung verbessert werden kann.

1.3.3.4 Live Migration und Load Balancing

Der Begriff der *Live Migration* beschreibt die Möglichkeit eine virtuelle Maschine zur Laufzeit von einem physikalischen System auf ein anderes zu verschieben. Dabei kann ein Gast während des laufenden Betriebes direkt von einem physikalischem Host auf einen anderen transferiert werden, wobei das unterliegende Virtualisierungssystem dafür sorgt, dass weder die Speicherinhalte und Prozesszustände noch Netzwerkverbindungen verloren gehen.

[6]engl. Dienstgüte

Dies eröffnet zum Beispiel die Möglichkeit *Load Balancing*[7] für den Host der virtuellen Gäste einzusetzen. *Load Balancing* für eine virtuelle Maschine beschreibt den Vorgang, bei dem ein Hostsystem einige Gäste an einen anderen Host übergeben muss, da die Gäste mehr als die zur Zeit vorhandenen Ressourcen benötigen. Es kann jedoch auch dazu eingesetzt werden um Wartungen am Hostsystem durchzuführen, ohne dass die Gäste des Hosts heruntergefahren werden müssen. Man würde also alle Gäste eines Hosts zur Laufzeit auf einen oder mehrere andere Hosts migrieren, und könnte danach Veränderungen am Host durchführen und diesen sogar neu starten, wobei die Gäste weiterhin ihren Dienst auf einem anderen Host verrichten können.

1.3.3.5 Hochverfügbarkeit

Um eine Umgebung hochverfügbar zu machen, ist es nötig das vorhandene System bei einem Totalausfall in möglichst kurzer Zeit wieder herzustellen. Virtualisierung kann zum Teil einen Beitrag dazu leisten. So kann man beispielsweise wie erwähnt eine Kopie des inaktiven Systems oder einen so genannten *Snapshot*[8] der virtuellen Maschine erstellen. Beim *Snapshot* werden, ähnlich wie bei der *Live Migration*, Prozess- und Speicherzustände eines laufenden Systems gesichert, jedoch kann dieses kopierte System zu beliebiger Zeit reaktiviert werden.

Es ergeben sich so zwei Ausfallszenarien, bei denen eine Sicherung eines Gasts zur schnellen Wiederherstellung eines Systems beitragen kann. Zunächst kann bei einem schwerwiegenden Fehler der virtuellen Maschine, diese wieder durch eine Aktivierung der Sicherung hergestellt werden. Das zweite Szenario bezieht sich

[7] engl. Lastverteilung
[8] engl. Schnappschuss - eine Kopie des Gasts zur Laufzeit

auf Fehler in der Hostumgebung. Hierbei kann man die Sicherung von den Gästen des betroffenen Hosts auf einen oder mehrere andere Hostsysteme portieren und reaktivieren. Zu beachten ist jedoch, dass bei diesem Verfahren die Sicherungen nicht auf dem Hostsystem liegen sollten, sondern auf einem Speicher, der auch von anderen Hostsystemen genutzt werden kann.

So stellt der Einsatz von *Snapshots* und Kopien für Sicherungen von virtuellen Maschinen, eine kostengünstige Alternative für eine einfache Hochverfügbarkeitslösung dar, da in erster Linie lediglich Plattenplatz für die Ablage von Sicherungen benötigt wird.

1.3.3.6 Administrierbarkeit und Wartbarkeit

Virtualisierung kann auch dazu beitragen die Administrierbarkeit und Wartbarkeit von Serversystemen zu verbessern. So ist es zum einen möglich, sehr schnell eine Kopie eines Systems anzulegen und dieses dann mit relativ wenig Aufwand als eine neue virtuelle Maschine zu betreiben. Diese System-Images[9] können bei den meisten Virtualisierungslösungen direkt auf dem Host abgelegt werden und eine vollständig neue virtuelle Maschine innerhalb weniger Minuten erstellen.

Da es nur noch eine Basishardware in der virtuellen Maschine gibt, muss auch nur diese Systemarchitektur für die Gäste eines Hosts gewartet werden und nicht eine ganze Reihe von Servern. Die virtualisierten Betriebssysteme greifen dann entweder direkt auf die vorhandenen Ressourcen zu oder nutzen die vom Virtualisierungssystem zugewiesenen Ressourcen.

[9] engl. Abbilder

1.3.3.7 Verlässlichkeit und Verfügbarkeit

Wie zuvor erwähnt ist ein großes Problem der Virtualisierung, dass der Host den *Single Point of Failure* darstellt. Das bedeutet, dass ein Fehler auf dem Host sich auf alle Gäste auswirken kann. Diesem Problem kann man entgegenwirken, indem man den Host zusätzlich absichert, zum Beispiel durch Redundanz der Hardware und regelmäßige Sicherung der Gäste.

Die Gäste wiederum erhalten durch Virtualisierung häufig einen hohen Grad von Verlässlichkeit und Verfügbarkeit. Durch die Möglichkeit Snapshots von den Gästen jederzeit anlegen zu können, ist es zum Beispiel möglich, das Betriebssystem bei einem Ausfall oder auch fehlgeschlagenen Update wiederherzustellen.

1.3.3.8 Virtualisierung und IT-Sicherheit

Virtualisierung bietet noch eine Reihe von weiteren Anwendungsgebieten. So ist es möglich, Virtualisierung für eine forensische Analyse zu verwenden, indem beispielsweise ein reales System in eine virtuelle Umgebung kopiert wird, um dort Analysen durchzuführen. Wenn die Analyse zu einer Zerstörung des Systems führen sollte, so kann die virtuelle Umgebung wieder hergestellt werden.

Es gibt jedoch auch gewisse Risiken, die im Zusammenhang mit IT-Sicherheit von Virtualisierung ausgehen. So wurde beispielsweise in der Konzeptstudie *„SubVirt: Implementing maleware with virtual machines“*[9] der Universität Michigan beschrieben, wie es möglich ist, Virtualisierung so zu missbrauchen, dass ein nicht virtualisiertes Betriebssystem von einem Angreifer in eine virtuelle Umgebung verschoben wird, um dann vollständige Kontrolle über dieses System zu erlangen.

Die Vorraussetzung für einen solchen Angriff ist zwar ein temporärer Vollzugriff auf das System, dieser kann aber beispielsweise durch eine Schwachstelle in einem Dienst erreicht werden. Die Veränderungen am System werden jedoch letztendlich unter Umständen nicht offensichtlich für den Nutzer des Systems, womit alle Prozesse auf dem System kompromittiert wären.

1.4 Funktionsweise unterschiedlicher Virtualisierungsansätze

Es gibt eine Reihe verschiedener Ansätze, um Betriebssystemprozesse voneinander isoliert laufen zu lassen. Dies kann mit Funktionen des Hostbetriebssystems erzielt werden, durch komplexe Systemvirtualisierungen einer Hardwarearchitektur oder auch mit Hardware-gestützter Virtualisierung durch den Prozessor.

1.4.1 Kapselung von einzelnen Prozessen

Eine einfache Möglichkeit, auf einem Unix-Betriebssystemprozesse voneinander isoliert laufen zu lassen, ist der Einsatz einer so genannten *Change Root* (chroot) Umgebung. Als *root*[10] bezeichnet man in einem Linux-Betriebssystem, das Stammverzeichnis oder auch das Wurzelverzeichnis eines Verzeichnisbaums. In den Unterverzeichnissen dieses Stammverzeichnisses befinden sich alle nötigen Programme, von den Programmen benötigte Bibliotheken, Schnittstellen zu Geräten und sonstige Dateien des Betriebssystems. Für die Programme und Prozesse existiert kein Verzeichnis unterhalb dieser Wurzel.

Bei der Change Root Umgebung handelt es sich um eine sehr einfache Trennung eines Prozesses vom ursprünglichen System durch eine so genannte *Wrapper Funktion* (siehe Abbildung 1.1). Dies ist genau genommen eine Betriebssystemfunktion, die einen Prozess und somit dessen Kindprozesse in einem Unterverzeichnis der Dateisystemebene ausführt und dort einsperrt. Das heißt, sie hindert diesen Prozess und dessen Kinder daran, auf Dateien unterhalb

[10] engl. Wurzel

des Change Root Verzeichnisses zuzugreifen. Das Verzeichnis wird für den Prozess also zu seinem Wurzelverzeichnis.

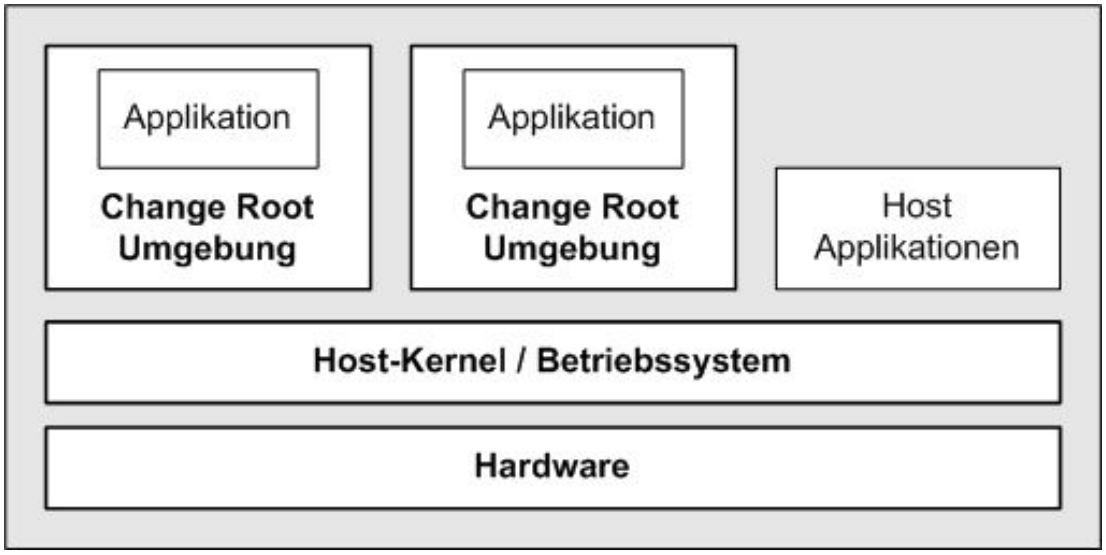

Abbildung 1.1: Change Root Umgebung

In diesem Zusammenhang ist zu erwähnen, dass alle Verzeichnisse und Geräte in einem Unix-Betriebssystem durch eine Datei dargestellt werden. Das Ziel einer Change Root Umgebung ist es also, zu verhindern, dass ein Prozess und dessen Unterprozesse überhaupt die Möglichkeit bekommen, auf Dateien zuzugreifen, die sie nicht direkt benötigen, um ihre Funktion ausführen zu können.

Andererseits muss jedoch darauf geachtet werden, dass dem Prozess alle benötigten Systembibliotheken, Binär- und Konfigurationsdateien und auch Verknüpfungen auf Geräte innerhalb der Change Root Umgebung zur Verfügung gestellt werden, zum Beispiel durch Kopieren der nötigen Dateien und Verzeichnisse in Unterverzeichnisse der Change Root Umgebung.

Die Change Root Funktion selbst liefert jedoch keine Möglichkeiten Einschränkungen der CPU-Verteilung oder der Eingabe/Ausgabe direkt zu implementieren.

1.4.2 Virtualisierung auf Betriebssystemebene

Ein etwas komplexerer Virtualisierungsansatz, unter Einsatz des Hostbetriebsystem-Kernels, ist die Virtualisierung von Prozessen in voneinander getrennten Instanzen auf der Hostbetriebssystemebene. Diese Instanzen können ein nahezu vollständiges Betriebssystem darstellen, das nicht mit einem eigenen Kernel läuft, sondern auf die Ressourcen des Hostbetriebssystem-Kernels zurückgreift. Diese Herangehensweise nutzt zum einen den Ansatz der Change Root Umgebung, erweitert diesen aber um zusätzliche Funktionen, wie der variablen Verteilung von CPU-Zeit oder das Bereitstellen von virtuellen Netzwerkstrukturen und Zugriff auf Geräte, für einzelne Prozesse beziehungsweise Prozessinstanzen. Es wird dafür gesorgt, dass ohne großen manuellen Aufwand eine nahezu vollständige virtuelle Umgebung für diese Instanzen bereitgestellt wird, die alle benötigten Bibliotheken und Kommunikationsschnittstellen von und zu Geräten zur Verfügung hat (siehe Abbildung 1.2).

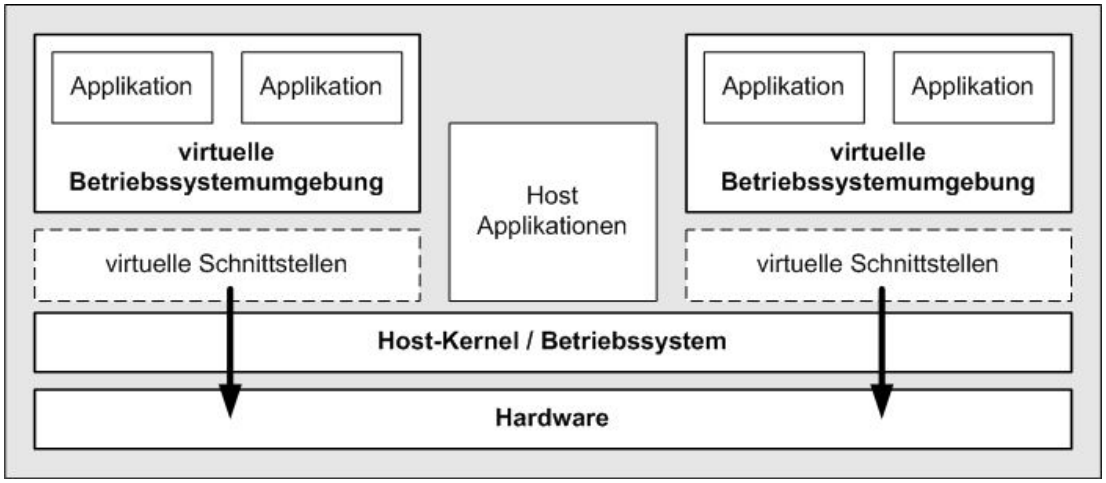

Abbildung 1.2: Virtualisierung auf Betriebssystemebene

Genauso wie bei der Change Root Umgebung kommt, wie bereits erwähnt, lediglich der Kernel des Hosts zum Einsatz, der das vollständige Management der Gäste übernimmt. Daher sind Verfahren, die diesen Ansatz verfolgen, meistens sehr performant, so dass Gäste mit der selben Geschwindigkeit laufen können, wie der Host selbst. Häufig ist auch die Ausnutzung des Systems sehr effektiv, da der Betriebssystem-Kernel lediglich alle vorhandenen Ressourcen auf die laufenden Prozesse verteilen muss. Virtualisierungssysteme, die diesen Ansatz verfolgen, sind *OpenVZ*[11], dessen kommerzielle Variante *Virtuozzo*[12] und *Linux-VServer*[13].

1.4.3 Virtualisierung auf Systemebene

Die zuvor genannten Ansätze zur Virtualisierung auf Betriebsystemebene haben den entscheidenden Nachteil, dass lediglich der Kernel des Hostsystems genutzt werden kann und nur real vorhandene Geräte im System zur Virtualisierung eingesetzt werden können. Soll beispielsweise ein BSD-Gast auf einem Linux-Host

[11] http://openvz.org/
[12] http://www.virtuozzo.com/
[13] http://linux-vserver.org/

betrieben werden, ist dies mit den zuvor vorgestellten Ansätzen nicht möglich. Soll sogar Hardware emuliert und so dem Gastbetriebssystem beispielsweise eine komplett andere Struktur als die real existierende Hardware bereitgestellt werden, ist dies fast gar nicht möglich. Eine vollständige Betriebssystemvirtualisierung kann durch Emulation, Paravirtualisierung oder Hardwarevirtualisierung erfolgen.

1.4.4 Virtualisierung durch Emulation

Emulation bedeutet im Zusammenhang mit Virtualisierung, dass Teile eines Systems oder auch das gesamte System für eine virtuelle Umgebung nachgebildet werden, ohne dass die real darunter liegende Hardware des Hostsystems diese Komponenten besitzt. Durch Emulation kann also sogar die CPU für ein virtuelles System nachgebildet werden. Dieses Verfahren hat jedoch den Nachteil, dass Emulation durch erhöhte Komplexität um ein vielfaches mehr an Ressourcen benötigt. Damit ist dieses Verfahren nur eingeschränkt zur Virtualisierung von Serverbetriebssystemen einsetzbar. Um eine Emulation durchzuführen, ist es nötig das Betriebssystem des Hosts und seine unterliegende Hardware von den virtuellen Umgebungen zu trennen. Diese Trennung wird durch den Einsatz eines so genannten Hypervisor ermöglicht werden, was im Folgenden näher erläutert werden soll.

Ein so genannter *Hypervisor*, auch bekannt als *Virtual Machine Monitor* (VMM), stellt eine Verwaltungsinstanz für virtuelle Maschinen dar, die bei der Virtualisierung auf Systemebene zwischen dem Hostbetriebssystem und den Gästen liegt. Der VMM übernimmt zum einen die Verwaltung der Gastsysteme und stellt diesen vor allem eine virtuelle Architektur zur Verfügung (siehe Abbildung 1.3). Diese virtuelle Architektur kann zum einen durch die

zuvor beschriebene Emulation erstellt werden, also durch Nachbildung von Komponenten, die nicht im Host physikalisch vorhanden sind, aber auch durch Verteilung der physikalischen Komponenten auf mehrere Gäste.

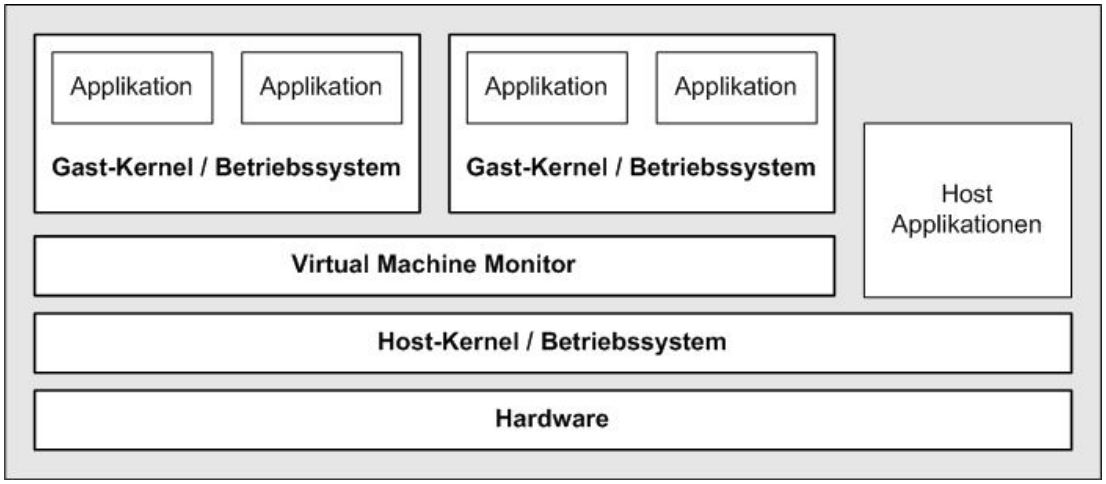

Abbildung 1.3: Virtualisierung auf Systemebene

So ist es möglich, unter einem VMM Gäste zu betreiben, die einen eigenen Betriebssystem-Kernel verwenden, unter dem dann wiederum Applikationen laufen, die direkt den Kernel des Gastsystems verwenden. Durch dieses Konzept ist es nicht nötig, dass das Gastsystem für die virtuelle Umgebung angepasst werden muss. Der Gast läuft also unmodifiziert. Mit diesem Verfahren kann beispielsweise ein unmodifiziertes FreeBSD auf einem Linux-Host betrieben werden.

Durch den Einsatz eines Hypervisor auf Systemebene entstehen jedoch gravierende Nachteile. Zum einen wird die Performance, wie zuvor beschrieben, durch den Prozess der Emulation beeinträchtigt, da Teile der unterliegenden Hardware vollständig durch Software dargestellt werden. Zum anderen müssen beispielsweise Interrupts[14] eines Gast-Kernels an das System durch den

[14]engl. Unterbrechungsanfragen

Hypervisor abgefangen werden, da diese nicht direkt an den physikalischen Prozessor weitergeleitet werden dürfen. Dies würde sonst unter Umständen zum Absturz des Hostsystems führen. Diese Umlenkung von Gastsystem-Interrupts, über den Hypervisor, an den Prozessor stellt wiederum eine hohe Einbuße an Performance dar, da der Hypervisor immer daran arbeiten muss, seine Gäste in bestimmten Bereichen ständig zu überwachen. Der Hypervisor selbst erzeugt also eine Last. Virtualisierungslösungen, die nach diesem Hypervisorkonzept arbeiten sind beispielsweise *VMWare Workstation*[15] und *Microsoft Virtual PC*[16].

1.4.5 Paravirtualisierung

Paravirtualisierung entspricht vom Prinzip her der Virtualisierung durch Emulation, da auch hierbei ein Hypervisor zum Einsatz kommt, der Gäste mit eigenen Betriebssystem-Kernel verwaltet. Jedoch muss bei diesem Verfahren der Kernel eines Gasts zuvor angepasst werden. Dies ist nötig, da bei Paravirtualisierung keine einzelnen Teile der Hosthardware emuliert werden, sondern dem Gast eine vollständige virtuelle Architektur, mit virtuellen Geräten, entsprechend einem physikalischen Rechnersystem zur Verfügung gestellt wird. Dieses virtuelle Rechnersystem ist zwar der real vorhandenen Hosthardware sehr ähnlich, jedoch haben Gäste keinen direkten Zugriff auf diese. Paravirtualisierung ist also eine Mischform aus der Virtualisierung auf Systemebene und Virtualisierung auf Betriebsystemebene.

Die Portierung des Gast-Kernels entspricht einer Anpassung des Gastbetriebssystems auf eine neue reale Hardwarearchitektur. Portierungen dieser Art sind speziell für quelloffene Systeme, wie Linux und BSD, sehr einfach zu

[15] http://www.vmware.com
[16] http://www.microsoft.com/windows/virtualpc

implementieren. Bei nicht quelloffenen Systemen, wie beispielsweise Windows resultiert jedoch der Nachteil, dass Paravirtualisierung nicht angewendet werden kann. Zu den Vertretern der Paravirtualisierung zählen das *Xen*[17] Projekt und das auf Linux-Systeme spezialisierte *User-Mode Linux*[18].

Ein großes Problem, das speziell die zuvor vorgestellten Hypervisorverfahren haben, ist die benötigte Performance des Hypervisor bei der Ressourcenverteilung an die Gäste. Gastbetriebssysteme sind für gewöhnlich nicht für virtuelle Architekturen entwickelt, sondern versuchen die Hardware in einem System direkt anzusprechen. Ein Hypervisor muss deshalb meist sehr aufwendig diese Versuche des Gasts mit dem System zu kommunizieren, abfangen und weiterleiten. Sobald mehrere Gäste im System vorhanden sind, kommt zusätzlich der Aufwand dazu, die verschiedenen Aufrufe der einzelnen Systeme zu isolieren und der Reihe nach abzuarbeiten. Zwar tritt dieses Problem bei Paravirtualisierung nur schwächer auf, da Gäste dort für die virtuelle Hardware angepasst werden, es ist aber auch dort vorhanden. Das Problem selbst ist vor allem auf das Design heutiger 32 Bit Prozessoren zurückzuführen, da diese ursprünglich nicht für Virtualisierung ausgelegt wurden.

1.4.6 Das IA-32 Ringkonzept

Die *Intel Architecture 32* (IA-32) oder auch i386 Architektur, auf der alle heutigen 32 Bit Intel Prozessoren aufbauen, stellt einen Sicherungsmechanismus für Operationen in Form von vier Berechtigungsebenen (privilege levels) oder auch Ringen zur Verfügung.[19] Diese Ringe werden mit den Ziffern 0 bis 3

[17] http://www.cl.cam.ac.uk/Research/SRG/netos/xen/
[18] http://user-mode-linux.sourceforge.net/
[19] Vgl. [5], IA-32 Intel Architecture Software Developer's Manual, Kapitel 6.3.5

bezeichnet, wobei es gilt: je höher die Ziffer, desto weniger Berechtigungen haben Operationen, die in dieser Ebene laufen. Jede Ebene besitzt dazu einen eigenen *Stack*[20], auf dem die Operationen, die der Prozessor ausführen soll, abgelegt und der Reihe nach abgearbeitet werden. Eine Operation, die in einer bestimmten Ebene läuft, hat lediglich einen sehr eingeschränkten Zugriff auf Operationen in einer der darüberliegenden Ebene (siehe Abbildung 1.4).

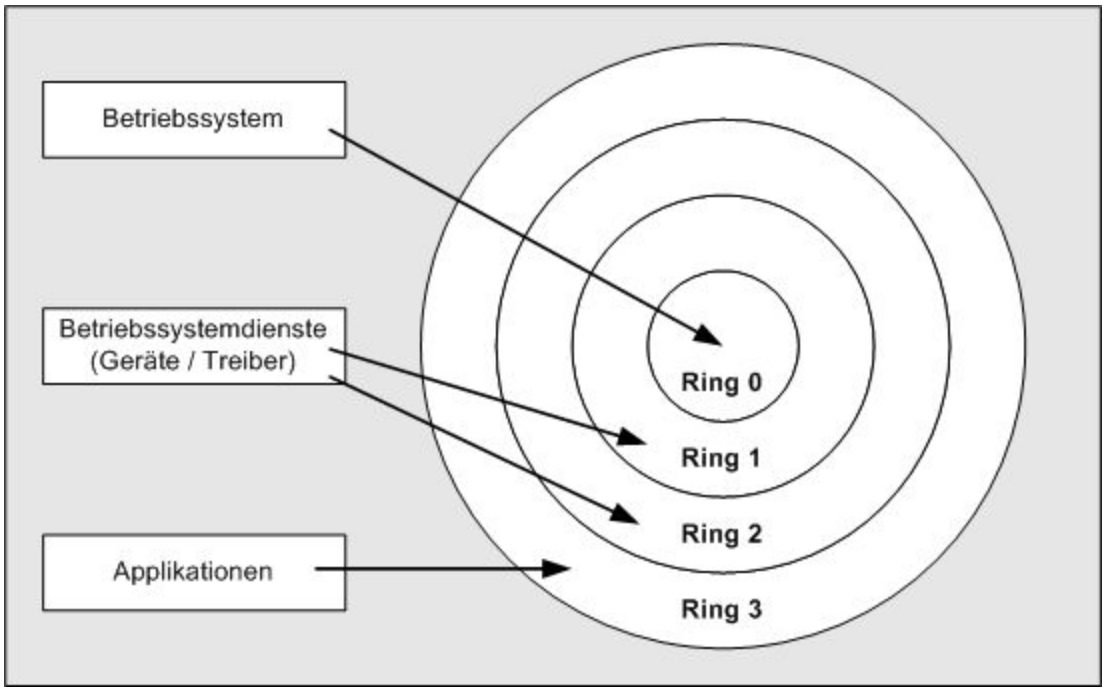

Abbildung 1.4: IA-32 Ring Konzept

Dieses Konzept ist aus dem folgenden Hintergrund entstanden: Um die Verlässlichkeit eines Betriebssystems zu garantieren, ist es nötig, dass Kernel-Operationen privilegiert und gekapselt ablaufen. Weniger privilegierten Operationen, wie beispielsweise die einer normalen Anwendung, dürfen nicht die Operationen des Betriebssystem-Kernel stören oder gar manipulieren.

[20] engl. Stapel

Es ist die Aufgabe des Betriebssystems, dieses Konzept sinnvoll zu nutzen. Die meisten Betriebssysteme sind so entwickelt, dass der Betriebssystem-Kernel im Ring 0 läuft und die Anwendungen im Ring 3. Ring 1 und 2 werden in einigen Betriebssystemen beispielsweise für Gerätetreibern genutzt.

Hieraus ergibt sich für die vollständige Virtualisierung eines Betriebssystems das unter Kapitel 1.4.4 erläuterte Problem des hohen Verwaltungsaufwandes eines Hypervisor. Betriebssysteme sind im allgemeinen so programmiert, dass sie im Ring 0 laufen und vollständige Kontrolle über den Prozessor und die Hardware haben. Ein *Virtual Machine Monitor* muss das Betriebssystem in einer virtuellen Maschine in einen virtuellen Ring 0 verschieben und dafür sorgen, dass Systemaufrufe des Betriebssystem-Kernels wie direkter Zugriff auf Ressourcen oder Interrupts[21] abgefangen werden. Die abgefangenen Aufrufe können dann durch den VMM an die reale Hardware weitergeleitet oder durch virtuelle Funktionen emuliert verarbeitet werden. Wenn mehrere Gastsysteme parallel auf einem Host laufen, muss der VMM zusätzlich dafür garantieren, dass sich die Gäste nicht untereinander behindern und den Systemaufrufen der einzelnen Umgebungen jeweils Ressourcen zugeteilt werden. Dieses Management ist bei vielen Virtualisierungslösungen sehr aufwendig und verschlingt dementsprechend Ressourcen.

[21] engl. Unterbrechungsanforderungen an die CPU

1.4.7 Hardwarevirtualisierung

Wie in Kapiel 1.4.6 gezeigt, bereitet die Architektur der normalen Prozessoren, die auf der IA-32 Architektur basieren, einige Probleme bei der optimalen Nutzung von Virtualisierungsverfahren. Sowohl Intel als auch AMD haben darauf reagiert und Prozessoren entwickelt, die das Problem adressieren.

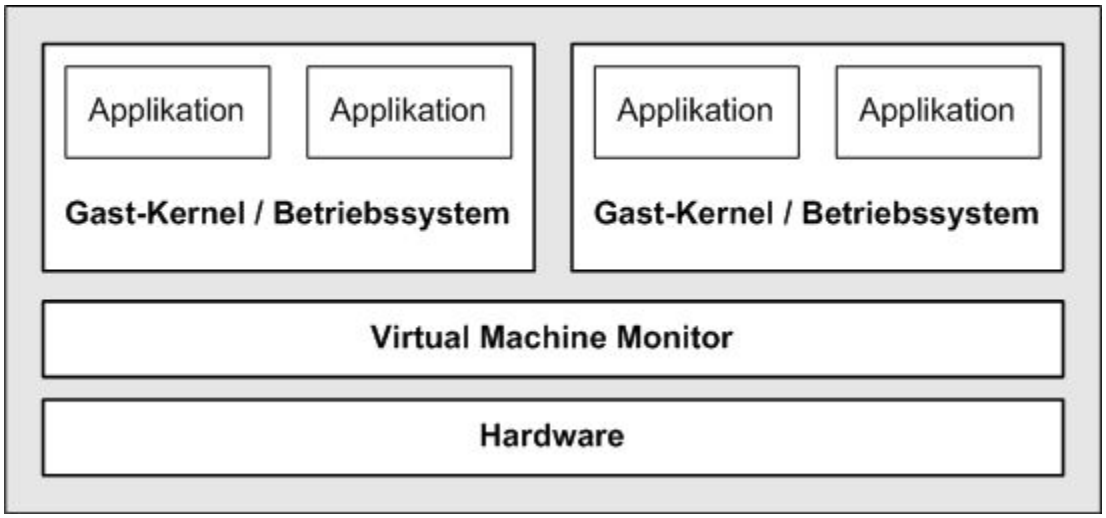

Abbildung 1.5: Virtualisierung auf Prozessorebene

Wie zuvor erwähnt, müssen einige Einbußen an Performance bei einer vollständigen Betriebssystemvirtualisierung hingenommen werden, da der Hypervisor durch das Konzept der vier Berechtigungsebenen einiges an Arbeit aufbringen muss, um den Gast davon abzuhalten direkt auf den Ring 0 zuzugreifen.

Die *Intel Virtualization Technology* (Intel VT)[22], die in neuere Prozessoren integriert wird, stellt dem Hypervisor einen neuen Modus mit dem Namen *Virtual Machine Execution* (VMX) zur Verfügung[23], indem der *Virtual Machine*

[22] http://www.intel.com/technology/computing/vptech/
[23] Vgl. [4], Intel Virtualization Technology Specification for the IA-32 Intel Architecture

Monitor über den so genannten *VMX root mode* das Management der Gäste betreiben kann. Ein ähnliches Prinzip verfolgt AMD mit der *Secure Virtual Machine* (SVM) Architecture, die mit AMD Prozessoren, die die *Pacifica Virtualization Technology* unterstützen, aufgerufen werden können.

Dies wird jeweils erreicht, indem dem VMM ein zusätzlicher *Ring -1* bereitgestellt wird. Somit erhält dieser die Kontrolle über den Ring 0, wo seine Gäste ohne besonders aufwendige Anpassungen nun laufen können.[24] Der Virtual Machine Monitor agiert also in einer Schicht zwischen Hardware und Gastbetriebssystemen (siehe Abbildung 1.5). Das Virtualisierungssystem Xen unterstützt in seiner aktuellen Version sowohl Paravirtualisierung als auch die hier vorgestellte Virtualisierung auf Hardwareebene.

[24]Vgl. [14] The Rings of Power: Intel's VT-x Technology and the Secrets of Virtualization

1.5 Freie und kommerzielle Virtualisierungslösungen

Die vorgestellten Virtualisierungsansätze werden sowohl durch Open Source Lösungen, als auch diverse kommerzielle Virtualisierungslösungen bereitgestellt. Während Virtualisierungslösungen wie Xen[25], User-Mode-Linux[26] und Linux-Vserver[27] seit Beginn ihrer Entwicklung als Open Source Projekte entwickelt wurden, an denen eine Gemeinschaft von freiwilligen Entwicklern und Sponsoren beteiligt waren, werden für den produktiven Einsatz in Firmen meist kommerzielle Virtualisierungsprodukte eingesetzt.

Zu den bekanntesten kommerziellen Virtualisierungssystemen zählen die Produkte *VMWare Workstation*, *VMWare GSX Server* und *VMWare ESX Server*[28], die durch Emulation einen Betrieb von nahezu allen IA-32 basierenden Betriebssystemen ermöglichen. Des weiteren bietet Microsoft mit *MS Virtual Server*[29] und *MS Virtual PC*[30] zwei Virtualisierungslösungen an, die ursprünglich lediglich den Einsatz von Microsoft Betriebssystemen offiziell ermöglichten, inzwischen jedoch auch Linux voll unterstützen. Der Anbieter SWSoft[31] stellt schließlich eine Virtualisierungslösung mit dem Namen *Virtuozzo*[32] zur Verfügung, die für den Einsatz von virtuellen Linux-Servern vor allem im Hosting-Bereich genutzt wird.
Eine interessante Entwicklung, die seit der Veröffentlichung von Xen zu beobachten ist, ist der Trend zur kostenlosen Vergabe von kommerziellen

[25] http://www.cl.cam.ac.uk/Research/SRG/netos/xen/
[26] http://user-mode-linux.sourceforge.net/
[27] http://linux-vserver.org/
[28] http://www.vmware.com/
[29] http://www.microsoft.com/germany/virtualserver/default.mspx
[30] http://www.microsoft.com/windows/virtualpc/default.mspx
[31] http://www.swsoft.com/
[32] http://www.swsoft.com/en/products/virtuozzo/

Virtualisierungsprodukten. Während die kommerziellen Produkte in den vergangenen Jahren eher marktbeherrschend waren, da es keine freien Lösungen mit nahezu gleichem Funktionsumfang und gleicher Performance gab, besteht nun durch Xen ein sehr performantes Paravirtualisierungssystemprodukt, das vollständig frei verfügbar ist. Dies hat eine grundsätzliche Veränderung für den kommerziellen Markt zur Folge. Anbieter wie VMware und Microsoft geben nun zwar ihre Produkte kostenlos ab, nehmen jedoch Geld für Serviceleistungen wie Wartung, Werkzeuge für die Bedienung und Garantie ein.

Der Anbieter SWSoft ging sogar soweit, dass er die Quellen seines kommerziellen Virtuozzo ebenfalls offenlegte und unter dem Projektnamen OpenVZ als Entwicklungsbasis für seine kommerzielle Variante nutzt, die vor allem zusätzliche Werkzeuge wie eine grafische Administrationsoberfläche liefert. Darüberhinaus plant SWSoft über Virtuozzo in Zukunft auch die Virtualisierungslösungen anderer Hersteller zu unterstützten. So sollen die Virtuozzo-Werkzeuge den Einsatz von verschiedenen Implementierungen wie Xen und VMWare ermöglichen.

1.6 Virtualisierungssysteme im Überblick

Nachdem zuvor die Funktionsweise der verschiedenen Virtualisierungsverfahren erläutert wurde, sollen an dieser Stelle einige Virtualisierungssysteme vorgestellt werden. Tabelle 1.1 gibt eine Übersicht über einige Virtualisierungsverfahren.

Tabelle 1.1: Vergleich von Virtualisierungsverfahren

Virtualisierung	Name	Eigenschaften	Unterstützte Gastsysteme
Betriebssystem	Change Root	Prozesse und deren Unterprozesse werden abgeschottet in Unterverzeichnissen des Hosts ausgeführt	Implementiert im Linux-Kernel, kann nur durch den Benutzer 'root' erstellt werden
	Open VZ, Virtuozzo, Linux-VServer	Durch Kernel-Patches erweitertes Verfahren auf Basis von Change Root	Linux-Systeme, die mit dem Host-Kernel arbeiten können
Emulation	VMWare Workstation und GSX-Server	Vollständige Systemvirtualisierung	Windows, Linux, BSD, Solaris
	Microsoft Virtual PC und Virtual Server	Vollständige Systemvirtualisierung	Windows, Linux
Paravirtualisierung	Xen	Gäste mit angepasstem eigenen Kernel	Linux, BSD, Solaris
	User Mode Linux	Gäste mit angepasstem eigenen Kernel	Linux
Hardwarevirtualisierung	Xen	mehrere unmodifizierte Gäste teilen sich eine Hardwareplattform	beliebiges Gastsystem auf Hardware, die die Intel VT oder AMD SVM unterstützt

1.7 Eingrenzung des Themenbereichs der Diplomarbeit

Im weiteren Verlauf wird sich der Themenbereich der Diplomarbeit auf die Virtualisierungslösungen OpenVZ und Xen beschränken. Diese wurden nach Begutachtung und Vergleich anderer vorhandener Open Source Verfahren wie Linux-VServer und User-Mode-Linux ausgewählt. Sie vefügen zum einen über eine gut gepflegte Dokumentation und bieten zum anderen eine einfache Implementierung unter *Fedora Core 5*[33], das als Betriebssystem für die exemplarische Implementierung dienen soll (siehe 3.1). Die Auswahl dieser beiden Verfahren soll es ermöglichen die Vor- und Nachteile der jeweiligen Implementierung zu erörtern, um eine Aussage über die Einsatzmöglichkeiten von Virtualisierungsverfahren in einer Linux-Serverumgebung treffen zu können.

[33] http://fedora.redhat.com/

2 Analyse ausgewählter Virtualisierungsverfahren

Zuvor wurde eine allgemeine Einführung in den Themenbereich der Virtualisierung gegeben und eine Übersicht über verschiedene Ansätze von Virtualisierungsverfahren vorgestellt. Im Folgenden soll nun ein Vergleich zwischen dem auf Betriebssystemebene agierenden Virtualisierungssystem OpenVZ und dem Paravirtualisierungssystem Xen durchgeführt werden. Hierzu wird zunächst eine detaillierte Analyse des Aufbaus und Funktionsumfangs der beiden Virtualisierungsverfahren vorgenommen. Das Ziel der Analyse ist es, die Grundlage für eine exemplarische Implementierung beider Verfahren zu schaffen, um diese dann abschließend für den Einsatz von Benchmarks nutzen zu können.

2.1 OpenVZ

OpenVZ ist eine Virtualisierungssoftware speziell für Linux, die eine Virtualisierung auf Betriebssystemebene ermöglicht, bei der der Kernel des Hostsystems auch das Management und die Ressourcenverteilung für die virtuellen Maschinen übernimmt (siehe Kapitel 1.4.2). OpenVZ ist ein Open Source Projekt, das von der Firma *SWsoft*[1] unterstützt wird und eine Entwicklungsbasis für die kommerzielle Virtualisierungslösung *Virtuozzo*[2] darstellt, welche ebenfalls von SWsoft vertrieben wird. OpenVZ ist unter der *GNU General Public License* (GPL)[3] Version 2 lizenziert.

2.1.1 Implementierung und Funktionsweise

OpenVZ erstellt voneinander isolierte und abgeschirmte virtuelle Umgebungen, so genannte *Virtual Private Server* (VPS), auf einem einzelnen physikalischen Hostsystem zur Verfügung. Die virtuellen Server sind voneinander vollständig getrennt. Sowohl die Prozesse und Prozesskommunikation, als auch Betriebssystemvariablen und das Dateisystem, werden jeder virtuellen Umgebung einzeln zugewiesen. Jeder VPS verhält sich wie ein realer Server, der eigene Benutzer, Prozesse, Bibliotheken, Konfigurationsdateien, Internet Protocol (IP) Adresse, Speicher und ähnliches hat, wobei jedoch kein direkter Zugriff auf die Hardware ermöglicht wird. In einem VPS kann Software ohne Anpassung an die OpenVZ-Umgebung betrieben werden.

Um die genannte Eigenschaften einer abgeschirmten Umgebung zu erhalten, muss der Host-Kernel eine zusätzliche Virtualisierungsschicht zur Verfügung stellen, die zur Zeit noch nicht von dem Standard-Linux-Kernel zur Verfügung

[1] http://www.swsoft.com/
[2] http://www.swsoft.com/virtuozzo
[3] http://www.gnu.org/copyleft/gpl.html

gestellt wird. Somit ist es momentan noch nötig Anpassungen am Kernel des Hostsystems durch einen Patch der Kernel-Quellen durchzuführen. Auf der OpenVZ Homepage werden jedoch auch Kernel-Pakete für diverse Distributionen bereitgestellt, in denen die Erweiterungen implementiert sind.

Die Virtual Private Server auf einem Host laufen alle unter dem Kernel des Hostsystems. Dies bietet vor allem einen enormen Geschwindigkeitsvorteil, da nur minimal Performance für das Ressourcenmanagement verloren geht. Dies wird hauptsächlich dadurch erreicht, dass nur der Host-Kernel die Verteilung von physikalischen Ressourcen vornimmt, und diese so optimal ausgenutzt werden können. Sind zum Beispiel mehrere CPUs im System vorhanden, werden nicht einzelne CPUs an einen virtuellen Server abgegeben, sondern die gesamte vorhandene Performance wird auf alle Prozesse in den verschiedenen Umgebungen verteilt. Es ist jedoch möglich, die Anteile an der Gesamtperformance unterschiedlich zu verteilen und so zum Beispiel einem Server mehr CPU-Zeit zuzusichern, als einem anderen. Einem OpenVZ-Gast stehen keine dedizierten CPU- oder Speicherressourcen zur Verfügung, jedoch können Geräte wie Netzwerkkarten auch exklusiv einem Gast zugeteilt werden. OpenVZ stellt hierzu ein Ressourcenmanagement für die Gäste zur Verfügung, das es erlaubt, diese Parameter und Einstellungen sogar zur Laufzeit vorzunehmen (siehe Kapitel 2.1.3).

Virtual Private Server werden über so genannte Templates installiert, wobei jedes Template eine Sammlung von Paketen einer bestimmten Linux-Distribution ist, die dazu dient, einen VPS zu installieren. Somit ist es auch theoretisch möglich auf einem Linux-Host virtuelle Server mit verschiedenen Linux-Distributionen zu betreiben, sofern natürlich der Host-Kernel alle benötigten Funktionen der einzelnen Distribution beherrscht. Dies ist nicht immer garantiert, da zum Beispiel

viele Distributionen mit einem stark veränderten Betriebssystem-Kernel arbeiten, der zusätzliche Funktionen bietet, die nicht im *standard Kernel*[4] implementiert sind.

2.1.2 OpenVZ VPS-Management

Das Management der OpenVZ Virtual Private Server wird mit Verwaltungswerkzeugen im Hostbetriebssystem durchgeführt. Zunächst dienen diese Werkzeuge der allgemeine Einrichtung und Bedienung eines VPS, also dem Installieren, Starten, Stoppen, Pausieren und Löschen eines VPS. Des weiteren ermöglichen diese Werkzeuge aber auch ein umfassendes Ressourcenmanagement, auf das im nachfolgenden Kapitel 2.1.3 noch näher eingegangen wird. Ein Großteil der Standardaufgaben kann über den Befehl *vzctl* durchgeführt werden. Über den Aufruf:

```
vzctl <subcommand> <vps-id> [OPTIONS]
```

kann zum Beispiel ein neuer VPS erstellt werden. Dabei steht *<subcommand>* für eine auszuführende Aktion wie beispielsweise *create* (VPS erstellen), *destroy* (VPS löschen) oder *set* (VPS Ressourcen konfigurieren). Die *<vps-id>* bezeichnet die ID des zu konfigurierenden Gast. Diese ID ist numerisch und kann jeden Wert größer oder gleich 100 an beim Erstellen des VPS zugewiesen bekommen. Natürlich muss diese ID für jeden VPS eindeutig sein. Als *[OPTIONS]* können schließlich Parameter für das *<subcommad>* eingefügt werden.

[4] http://www.kernel.org/

2.1.3 Ressourcenmanagement

Das Ressourcenmanagement von OpenVZ dient dem Verteilen der vorhandenen Systemressourcen auf die virtuellen Umgebungen. Dieses Management in OpenVZ-Umgebungen ist sehr wichtig, da die Auslastung eines OpenVZ-Hosts meist viel höher ist, als die eines Systems, das nur einen nicht virtualiserten Server betreibt. Da OpenVZ seinen Gästen wie zuvor erwähnt keinen direkten Zugriff auf die Hardware erlaubt, und somit beispielsweise die Performance von mehreren Prozessoren in einem System als ein einziger CPU-Ressourcenpool zur Verfügung steht, können diese Ressourcen über das Ressourcenmanagement auf die Gäste nach Belieben verteilt werden. Neben den CPU-Ressourcen gibt es in erster Linie noch die Zuweisung von Festplattenspeicherplatz und Systemspeicher.

Das Ressourcenmanagement erlaubt es, die verfügbaren Ressourcen unter den Gästen aufzuteilen und durch Isolierung dieser Ressourcenanteile einen so genannten *Quality of Service* zu garantieren. Durch die eindeutige Zuweisung von beispielsweise CPU-Zeit oder Systemspeicher wird dafür gesorgt, das eine Überlastung eines einzelnen virtuellen Servers nicht zur Beeinflussung oder gar einem Ausfall anderer Server führt, sondern lediglich diesen einen Server betrifft. Das Ziel von Ressourcenmanagement ist es also, zum einen eine optimale Ausnutzung der vorhandenen Ressourcen und zum anderen vor allem eine Fairness bei der Ressourcenverteilung unter den Gästen zu garantieren.

Das Management bietet auch die Möglichkeit, Informationen über den Zustand der einzelnen Maschinen zu sammeln und auszuliefern. Mit Hilfe dieser Informationen ist es dann möglich, eine Skalierung der Ressourcen vorzunehmen.

OpenVZ setzt drei spezielle Verfahren ein, um die Ressourcen eines Gasts zu verwalten:

- Zweistufiger Fair-CPU-Scheduler
- Zweistufige Disk Quotas
- User Beancounters

Wie zuvor erwähnt sind Änderungen an den Parametern sowohl für einen deaktivierten Gast als auch zur Laufzeit eines Gasts möglich. Diese Komponenten sollen nun im Folgenden näher erläutert werden.

2.1.3.1 CPU-Management

OpenVZ setzt einen so genannten *zweistufigen Fair-CPU-Scheduler*[5] ein. Bei diesem Konzept werden Anteile an *CPU-Zeit*, also der Zeitraum in dem die CPU-Ressourcen einem Gast zur Verfügung gestellt werden, in Form von vorgegebener Gewichtigkeit zur Verfügung gestellt. Dies bedeutet nichts anderes, als dass alle vorhandenen CPU-Ressourcen in verschieden großen Anteilen auf mehrere Gäste verteilt werden können. Die Zuweisung dieses CPU-Anteils wird bei der Erzeugung eines VPS durch einen Standardwert vordefiniert, kann aber jederzeit geändert werden. Über den Befehl *vzcpucheck* können die CPU-Ressourcen des Hosts zunächst in einem numerischen Wert ausgedrückt werden:

```
# vzcpucheck
Current CPU utilization: 6000
Power of the node: 100000
```

Man erhält also eine Angabe über die momentan genutzten Ressourcen (*Current CPU utilization*) und die gesamt verfügbaren Ressourcen des Hostsystems (*Power*

[5] http://fairsched.sourceforge.net/doc/fairsched.txt

of the node). Es ergibt sich hieraus die Möglichkeit einem Gast CPU-Zeit fest zuzusichern, was bedeutet, dass diese CPU-Ressourcen immer diesem Gast exklusiv zur Verfügung stehen. Des weiteren kann man aber auch einen maximalen Anteil an CPU-Zeit definieren, der größer oder gleich dem Anteil an zugesicherter CPU-Zeit sein darf. Beide Parameter werden über den *vzctl* Befehl für einen VPS gesetzt:

```
vzctl set 103 --cpuunits 10000 --cpulimit 20 --save
```

In diesem Beispiel wurde dem VPS 103 also 10% der CPU Zeit zugesichert (10000 von 100000 verfügbaren Einheiten) und ein maximaler Anspruch auf CPU-Zeit von 20% ermöglicht (–cpulimit 20). An dieser Stelle sollte auch erwähnt werden, das man auch dem Hostbetriebssystem auf diese Weise über einen Parameter *–ve0cpuunits* CPU-Zeit fest reservieren kann, um bei einer hohen Auslastung des Hosts immer noch genügend freie Ressourcen für eine Verwaltung der Gäste zu haben.[6]

Der OpenVZ Fair CPU-Scheduler wird als zweistufig bezeichnet, da dieser auf der ersten Ebene zunächst entscheidet, welchem virtuellen Gast er den zuvor beschriebenen Anteil an CPU-Zeit zuweist. Diese Entscheidung wird aufgrund des zugewiesenen Wertes von fest reservierten CPU-Einheiten durchgeführt. Auf der zweiten Ebene wiederum entscheidet der Standard-Linux-Scheduler, welchem Prozess in dieser virtuellen Umgebung er jeweils CPU-Zeit zuweist. Diese Entscheidung basiert daher auf den normalen Linux-Prozessprioritäten, die im System festgelegt sind.

[6] Vgl. [3], OpenVZ User's Guide, Kapitel 6 - Managing CPU Share

2.1.3.2 Festplattenmanagement

Der Festplattenspeicher unter OpenVZ wird durch den Einsatz von *zweistufigen Disk Quotas* den Gästen zugewiesen. Eine *Disk Quota* unter Linux bezeichnet die Begrenzung von vorhandenem Speicherplatz und der Anzahl von maximal verfügbaren *Inodes* für verschiedene Benutzer oder Benutzergruppen.

Ein Inode unter Linux-Dateisystemen ist ein Indexeintrag für Dateien, wobei unter Linux sowohl Verzeichnisse, als auch Geräteschnittstellen als Datei definiert werden, wie zuvor bereits in Kapitel 1.4.1 erläutert wurde. Für den Speicherplatz und die Anzahl der Inodes werden über Quotas so genannte Limits[7] definiert, die den verfügbaren Speicherplatz und die maximale Erzeugung von Dateien begrenzen. Dabei unterscheidet man zwischen *Softlimits*, die für einen begrenzten Zeitraum überschritten werden dürfen und *Hardlimits*, die niemals überschritten werden können.

Wenn beispielsweise eine Disk Quota mit 90MB Softlimit und 100MB Hardlimit auf das Home-Verzeichnis eines Benutzers eingerichtet wird, so darf dieser zwar für einen gewissen Zeitraum etwas mehr als 90 MB Daten ablegen, wobei er gewarnt wird, dass das Soft-Quota-Limit erreicht wurde. Es wird ihm jedoch nicht ermöglicht, die 100MB Grenze zu überschreiten. Das Hardlimit wirkt sich also wie ein Erreichen der maximalen Festplattenkapazität aus.

OpenVZ macht sich so Quotas auf zwei Ebenen zu nutze, um den vorhandenen Speicherplatz für Gäste zur Verfügung zu stellen. Auf der ersten Ebene kann der Administrator des Hostsystems Quotas nutzen um eine virtuelle Festplatte zu erstellen, die eine bestimmte Größe hat und eine definierte Anzahl an Dateien aufnehmen kann. Auf der zweiten Ebene können zusätzlich Quotas innerhalb

[7] engl. Grenzen

des Gasts für Benutzer und Gruppen definiert werden, wie zuvor im Beispiel beschrieben.

Eine Vergrößerung oder Verkleinerung dieser Quotas ist während des Betriebes möglich. Das bedeutet, man kann die Festplatte eines Gasts vergrößern oder verkleinern, ohne diesen neu starten oder gar abschalten zu müssen.

2.1.3.3 IPC-Ressourcenmanagement

Das Management von IPC-Ressourcen (Inter Process Communication)[8] für virtuelle Umgebungen unter OpenVZ wird von so genannten *User Beancounters*[9] *(UBC)* übernommen. Da Gäste in OpenVZ den Kernel der Hostumgebung nutzen, besitzen sie keine eigenen, exklusiv nutzbaren Ressourcen. Um trotzdem eine Verteilung von vorhandenen Ressourcen zu ermöglichen, besitzt jeder virtuelle Server einen eigenen Satz an User Beancounters. Diese dienen dazu dem jeweiligen virtuellen Server bestimmte Ressourcen zu garantieren oder auch einzuschränken, um eine Fairness bei der Verteilung von Ressourcen zwischen mehreren virtuellen Servern zu garantieren. User Beancounter verwalten wie erwähnt, die Ressourcen der *Inter-Prozess-Kommunikation*, zu denen in erster Linie die Eingrenzung des verfügbaren Hauptspeichers, der internen Kernel-Ressourcen zur Steuerung der Kommunikation unter Prozessen und des Netzwerk Puffers gehört. Dazu gehören unter anderem die maximale Anzahl von Prozessen und Threads[10], Netzwerk Sockets[11] und die maximale Anzahl gleichzeitig geöffneter Dateien.

Die UBC-Werte können in der OpenVZ-Hostumgebung mit dem Aufruf:

[8] engl. Interprozesskommunikation
[9] engl. frei übersetzt Erbsenzähler
[10] engl. Ausführungsstränge einer Software
[11] engl. Verbindungspunkte

```
cat /proc/user_beancounters
```

eingesehen werden. Dieser Aufruf liefert eine Liste der IPC-Einstellungen für alle im System aktiven OpenVZ-Gäste. Listing 2.1 zeigt einen exemplarischen Auszug dieser Liste mit den IPC-Einstellungen für einen Gast mit der UID 102.

```
  uid  resource          held    maxheld  barrier      limit  failcnt
  102: kmemsize       1316699    2090077  2752512    2936012        0
       lockedpages          0          0       32         32        0
       privvmpages       4631      15676    49152      53575        0
       shmpages            45         45     8192       8192        0
       dummy                0          0        0          0        0
       numproc             17         32       65         65        0
       physspages        2176       3874        0 2147483647        0
       vmguarpages          0          0     6144 2147483647        0
       oomguarpages      2176       3874     6144 2147483647        0
       numtcpsock           3          4       80         80        0
       numflock             3          4      100        110        0
       numpty               0          1       16         16        0
       numsiginfo           0          2      256        256        0
       tcpsndbuf            0          0   319488     524288        0
       tcprcvbuf            0          0   319488     524288        0
       othersockbuf      6672      13552   132096     336896        0
       dgramrcvbuf          0       8368   132096     132096        0
       numothersock         8         11       80         80        0
       dcachesize      114061     138649  1048576    1097728        0
       numfile            248        610     2048       2048        0
       dummy                0          0        0          0        0
       dummy                0          0        0          0        0
       dummy                0          0        0          0        0
       numiptent           10         16      128        128        0
```

Listing 2.1: Bsp. User Beancounters für einen Gast

Jeder IPC-Ressource in einer OpenVZ-Umgebung werden fünf Werte zugeteilt, die Auskunft über den momentanen Verbrauch (held), den bisher maximalen Verbrauch (maxheld) seit Aktivierung des virtuellen Servers, einem Softlimit (barrier), einem Hardlimit (limit) und einem Fehlerzähler, der das Erreichen des Hardlimits registriert. Diese Angaben können also helfen, Einsicht in

den Ressourcenverbrauch eines virtuellen Servers zu gewähren. Mit Hilfe des *Fehlerzählers* ist es vor allem möglich, eine optimale Skalierung der Ressourcenverteilung durchzuführen. Zeigt der Fehlerzähler mehrere Fehler an, bedeutet dies, dass die zugeteilten Ressourcen für die Anforderungen des Gasts zu bestimmten Zeitpunkten im Betrieb nicht ausgereicht haben. Der Gast hat versucht mehr Ressourcen einzubinden als ihm zugeteilt wurden und es können somit Schlussfolgerungen gezogen werden, ob die betroffene IPC-Ressource erhöht werden muss, der Gast neu konfiguriert werden sollte, so dass er weniger Ressourcen verwendet, oder dass Hostsystem allgemein überlastet ist. Leider ist eine Anpassung der UBC-Ressourcen nicht immer ganz trivial durchzuführen, was im Folgenden anhand des Speichermanagements über die User Beancounters verdeutlicht werden soll.

2.1.3.4 Speichermanagement

Das Speichermanagement in OpenVZ geschieht über die zuvor beschriebenen UBC-Parameter. Speicher wird den Gästen über mehrere Stufen zur Verfügung gestellt. Die folgenden Parameter aus */proc/user_beancounters* (Auszug aus Listing 2.1) sind für das Haupt- und SWAP-Speichermanagement des einen Gasts relevant:

```
resource          held   maxheld  barrier      limit  failcnt
kmemsize       1316699   2090077  2752512    2936012        0
privvmpages       4631     15676    49152      53575        0
physpages         2176      3874        0 2147483647        0
vmguarpages          0         0     6144 2147483647        0
oomguarpages      2176      3874     6144 2147483647        0
```

Listing 2.2: UBC-Parameter für Speichermanagement

Diese UBC-Parameter haben jeweils folgende Funtion in einem OpenVZ-Gastsystem:

- **kmemsize**

  Größe des reservierten Kernel-Speicher in Byte. Dieser Bereich steht immer dem Kernel zur Verfügung und darf nicht durch einen *SWAP*[12]-Prozess für andere Anwendungen in den Hintergrundspeicher verschoben werden.

- **privvmpages**

  Die mögliche Größe des maximal dynamisch zuweisbaren Gesamtspeichers, d.h. Hauptspeicher und SWAP Bereich, sofern dieser vorhanden ist und nicht bereits duch andere Gäste genutzt wird, in Speicherseiten von jeweils 4 Kilobyte (Kb) was 4096 Byte entspricht.

- **physpages**

  Die garantierten physikalischen Speicherseiten mit jeweils 4 Kb, also der reservierte *Random Access Memory* (RAM) Bereich.

- **vmguarpages**

  Der garantiert zuweisbare Speicher (RAM+SWAP) in der Größe von 4Kb, der für den Gast zur Verfügung gestellt werden kann.

- **oomguarpages**

  Garantiert zuweisbarer Gesamtspeicherbereich (RAM+SWAP), im Falle einer Situation, bei der der gesamtverfügbare Speicher vollständig durch Gäste ausgenutzt wurde. Hierdurch wird kein VPS zu einer Abschaltung gezwungen, solange er in einer solchen Situation nicht mehr als die über die Barriere eingegrenzten Speicherseiten verwendet.

[12] engl. Austausch

Ist es erforderlich einen Parameter zu erhöhen, so kann dies über den nachfolgenden Aufruf durchgeführt werden:

```
vzctl set <vps-id> --<UBC-Parameter> <barrier>:<limit> --save
```

Dabei müssen jedoch immer eventuell abhängige Parameter mit berücksichtigt werden, um keine Inkonsistenzen zu verursachen. Eine höhere Zuweisung beispielsweise von zugeteilten physikalischen Speicherseiten wird zu keinem Ergebnis führen, wenn die maximal erlaubte Nutzung unter diesem Wert liegt. Der Aufruf des nachfolgenden Befehls überprüft die zuvor durchgeführte Anpassung und gibt gegebenenfalls Vorschläge die Konfiguration zu korrigieren:

```
vzcfgvalidate /etc/sysconfig/vz-scripts/<vps-id>.conf
```

2.1.3.5 Netzwerkvirtualisierung

Virtual Private Server unter OpenVZ haben Zugriff auf jeweils ein oder mehrere virtuelle Netzwerkkarten oder auch physikalische Karten, die einem Gast exklusiv zugeordnet werden. Somit gibt es für jeden virtuellen Sever die Möglichkeit, eine oder mehrere IP-Adressen einzurichten. Die virtuellen Netzwerkkarten unter OpenVZ werden durch das Gerät *vznetdev* bereitgestellt. Dieses virtuelle Gerät stellt eine Netzwerkverbindung zwischen den Gästen und dem Host zur Verfügung. Das Gerät arbeitet nach einem so genannten *Zero-Copy Design*[13], bei dem alle Daten die über das Gerät verschickt werden sollen, direkt an das Gerät adressiert werden und nicht an irgend einer anderen Stelle im System zwischengespeichert werden. Dies ermöglicht zum einen eine sehr hohe Performance, zum anderen kann aber vor allem der Netzwerkverkehr

[13]Vgl. [8], EMP: Zerocopy OSbypass NICdriven Gigabit Ethernet Message Passing

eines Servers, der über diese virtuellen Geräte läuft, nicht von anderen Servern abgefangen werden. Es ist also nicht möglich, aus einem virtuellen Server die Datenpakete eines anderen Servers unmittelbar abzuhören.
Eine virtuelle Netzwerkkarte verhält sich wie eine physikalische Netzwerkkarte im System. So ist es auch möglich, den Netzwerkverkehr eines virtuellen Servers mit einer Firewall wie IPTables[14] zu schützen oder Netzwerk Diagnose Tools wie Wireshark[15] (ehemals Ethereal), Nmap[16] oder Nessus[17] einzusetzen. Das gleiche gilt für das Erstellen und Nutzen von Rounting Tabellen eines VPS. Man kann also verschiedene Gateways und Routen für den VPS definieren oder die *MTU Size*[18] anpassen.

2.1.4 Installation eines OpenVZ-VPS

Ein Virtual Private Server unter OpenVZ kann mit Hilfe so genannter *Templates* installiert werden. Ein Template bezeichnet zunächst eine Sammlung von Programmpaketen, zugehörigen Bibliotheken und Skripte, einer bestimmten Linux-Distribution.[19]

Um die die genannten Programmpakete, benötigte Bibliotheken und Skripte verfügbar zu machen, ist es nötig diese mit allen eventuell abhängigen Paketen auf dem Host zur Verfügung zu stellen. Da alle modernen Linux-Distributionen über einen Paketmanager wie zum Beispiel den *Yellowdog Updater Modified* (yum)[20] verfügen, mit dem die aktuellsten Pakete einer Distribution eingespielt

[14] http://www.netfilter.org/
[15] http://www.wireshark.org/
[16] http://insecure.org/nmap/
[17] http://www.nessus.org/
[18] Maximum Transfer Unit Size - die maximale Paketgröße
[19] Vgl. [3], OpenVZ User's Guide, Kapitel 5 - Managing Templates
[20] http://linux.duke.edu/projects/yum/

werden können, bietet OpenVZ-Werkzeuge an, mit denen sich diese Pakete direkt über den Hostpaketmanager in einen neuen VPS einspielen lassen. Die benötigten Informationen, welche Pakete für den Aufbau eines neuen VPS benötigt bzw. erfordert werden, befinden sich in so genannten *Template-Metadata* Dateien. Diese Dateien werden ebenfalls von OpenVZ zur Verfügung gestellt und können nach Bedarf angepasst werden.

Werden diese Pakete nun in einen VPS installiert, so kann aus diesem VPS ein so genannter *Template-Cache* erstellt werden. Ein Template-Cache bezeichnet einen als Tar-Archiv gepackten, funktionsfähigen VPS, der dazu genutzt werden kann neue VPS zu befüllen. Ein Template-Cache einer Distribution bietet den Vorteil, dass ein neuer VPS sehr schnell erzeugt werden kann, da bei einer Erzeugung zunächst keine Pakte mehr geladen oder installiert werden müssen. Der Template-Cache ist also eine Kopie eines VPS der zu einem neuen VPS aufgebaut wird. Über den Befehl *vzpkgcache* kann ein solcher Template-Cache sehr einfach erzeugt werden, da der Befehl direkt die Template-Metadata nutzt um einen temporären VPS aufzubauen, aus dem dann direkt ein entsprechender Template-Cache erzeugt wird. Ein neuer Virtual Private Server kann schließlich durch den folgenden Aufruf durch des unter Kapitel 2.1.2 beschriebene *vzctl* erstellt werden:

```
vzctl create <vps-id> [--ostemplate <template>] [--config <config>]
```

Hierbei steht *<vps-id>* für die ID des neu zu erzeugenden Servers, *<template>* für den zuvor erwähnten Template-Cache und *<config>* für eine Datei welche vorgegebene Konfigurationswerte, wie beispielsweise die User-Beancounter-Parameter, für den neu zu erstellenden Server enthält.

2.1.5 Spezielle Funktionen

2.1.5.1 Checkpointing

OpenVZ erlaubt es, Checkpointing für die Gäste auf einem Host durchzuführen. Checkpointing ist eine Funktion, die wie das unter Kapitel 1.3.3.5 erläuterte *Snapshot*-Verfahren den Zustand einer virtuellen Maschine einfriert und alle ihre internen Zustände in eine Datei abgespeichert. Es wird also ein Fixpunkt aller aktuellen Zustände eines Gasts erstellt. Von einer Hostkonsole aus kann ein Checkpoint über den folgenden Aufruf den Zustand eines Servers speichern:

```
vzctl chkpnt <vps-id>
```

Die erzeugte Checkpoint-Datei kann dann zu einem späteren Zeitpunkt dazu genutzt werden den Server wieder mit seinen gespeicherten Zuständen in Betrieb zu nehmen. Dies kann entweder auf dem aktuellen Host geschehen oder auf einem anderen physikalischen OpenVZ Host, um dort den virtuellen Server wieder mit seinen gesicherten Zuständen in Betrieb zu nehmen. Eine Wiederherstellung wird als *restore* bezeichnet und kann wie folgt durchgeführt werden:

```
vzctl restore <vps-id>
```

Zusammenfassend kann man sagen, dass durch Checkpointing ein Zustand eines laufenden Servers konserviert wird. Durch das Prinzip des Checkpointing ergeben sich jedoch vor allem weiterführende Möglichkeiten, wie die ebenfalls unter OpenVZ unterstützte *Live Migration* die im Folgenden beschrieben wird.

2.1.5.2 Live Migration

Bei der Live Migration kann, wie in Kapitel 1.3.3.4 erwähnt, eine virtuelle Umgebung während des Betriebes von einem physikalischen Host auf einen anderen übertragen werden.

```
vzmigrate --online <destination-node> <vps-id>
```

Aus Benutzersicht stellt sich eine solche Migration von einem Host auf einen andern im Idealfall als kurze Verzögerung von wenigen Sekunden dar. Dazu muss jedoch auch die Hostumgebung so eingerichtet sein, dass eine Live Migration überhaupt möglich ist, was zum Beispiel bedeutet, dass die beiden Hosts ihre Gäste im selben Subnetz betreiben, und dass die Verbindung zwischen den beiden Hosts performant ist. Des weiteren muss auch der Festplattenspeicher, den der zu verschiebende Gast nutzt, auf beiden Hostumgebungen verfügbar sein.

2.1.6 Virtuozzo

Wie bereits zuvor unter Kapitel 1.5 erwähnt ist OpenVZ ein Ableger des kommerziellen Virtualisierungssystems *Virtuozzo*[21], das zur Verfügung gestellt wurde, um eine Basisplattform für den Test von Erweiterungen in Virtuozzo zu liefern. Virtuozzo ist ein Virtualisierungssystem, das in erster Linie in sogenannten Webhosting-Umgebungen eingesetzt wird, wo Anwendern ein kostengünstiger Virtual Private Server beispielsweise für eigene Webauftritte zur Verfügung gestellt wird. Der Hauptunterschied zwischen Virtuozzo und OpenVZ liegt in den Verwaltungswerkzeugen, die Virtuozzo zusätzlich mitbringt. So ist es beispielsweise möglich alle VPS-Systeme auf einem Server per grafischer Benutzeroberfläche[22]

[21] http://www.swsoft.com/en/products/virtuozzo/
[22] in engl. Graphical User Interface (GUI)

zu administrieren, was speziell die Administration auf Hostservern mit einem hohen Aufkommen von Gästen vereinfacht. Durch das kombinierte Anbieten einer kommerziellen und nicht-kommerziellen Lösung, schafft SWSoft also eine Basis, um die Stabilität von Virtuozzo zu verbessern, da bereits im Voraus Implementierungen von neuen Funktionen umfassend getestet werden können.

2.2 Xen

Xen ist ein Virtualisierungssystem, das von der Systems Research Group der Universität von Cambridge entwickelt wurde. Es entstand aus einem Zweig des XenoServers Projekt[23], dessen Ziel es, ist eine öffentliche Infrastruktur für großflächig verteilte Computersysteme zu erstellen. Xen ist Open Source und wurde unter der *GNU General Public License*[24] veröffentlicht.

Xen wird von verschiedenen namenhaften Herstellern wie XenSource[25], Intel, AMD, IBM, RedHat, Novell, HP und Microsoft unterstützt. Dabei haben sich speziell RedHat mit *RedHat Enterprise Linux*, Novell mit *Suse Linux Enterprise* und XenSource mit *Xen Enterprise* das Ziel gesetzt, Xen für den kommerziellen Gebrauch in Unternehmen anzubieten.

2.2.1 Implementierung und Funktionsweise

Xen unterstützt sowohl Para- als auch Hardwarevirtualisierung. Bei der Paravirtualsierung stellt der Xen Virtual Machine Monitor seinen Gästen Schnittstellen zu einer virtuellen Hardwarearchitektur zur Verfügung, die ähnlich der real vorhandenen Hardware im System ist. Es müssen daher Anpassungen am Kernel des Gastsystems durchgeführt werden, was jedoch durch die Tatsache vereinfacht wird, dass das virtuelle System dem realen sehr ähnlich ist. Der Aufwand für die nötigen Anpassungen bezieht sich so lediglich auf den Kernel des Gasts. Diese Änderungen sind speziell bei Open Source Betriebssystemen relativ einfach zu implementieren, da beispielsweise der offene Quellcode des Linux-Kernels erweitert werden kann, um diesen für die *Xen-Architektur* zu

[23] http://www.cl.cam.ac.uk/Research/SRG/netos/xeno/
[24] http://www.gnu.org/licenses/gpl.html
[25] http://www.xensource.com

übersetzen. Dies funktioniert analog zu der Übersetzung von Kernel-Quellen auf eine Prozessorarchitektur, beispielsweise i386 oder i686. Die *Userland Prozesse*[26] müssen nicht angepasst werden und laufen daher unmodifiziert unter dem angepassten Kernel des Gasts. Xen ist momentan für Linux, BSD und Solaris verfügbar.

2.2.1.1 Hardwarevirtualisierung mit Xen

Xen unterstützt auch Hardwarevirtualisierung, die von neuen Prozessoren mit Intel VT und AMD SVM ermöglicht wird (siehe Kapitel 1.4.7). Wenn Xen als Hypervisor für ein System mit Hardwarevirtualisierung eingesetzt wird, ist es nicht länger nötig den Kernel der Gäste anzupassen, da Xen lediglich die Verteilung der Hardwareressourcen übernehmen muss. So lässt sich sogar ein nicht quelloffenes Betriebssystem wie beispielsweise Microsoft Windows XP als Gast unter Xen betreiben, da wie unter Kapitel 1.4.6 beschrieben, das Gastsystem nun seinen Betriebssystem-Kernel unmodifiziert im Ring 0 betreiben darf, während der Hypervisor im Ring -1 läuft. In den folgenden Abschnitten wird nicht weiter auf den Betrieb von Hostsystemen mit Hardwarevirtualisierung eingegangen, wobei sich die meisten der behandelten Themen auch auf solche Systeme übertragen lassen.

2.2.1.2 Hardwareunterstützung unter Xen

Xen unterstützt momentan, in seiner aktuellen Version 3.0, Prozessoren der x86 Familie mit 32 Bit ab dem Typ P6[27] und aktuelle Prozessoren mit x86 64 Bit Architekturen. Außerdem werden Maschinen mit mehreren Prozessoren,

[26] Applikationen der Gäste auf Benutzerebene

[27] Pentium Pro, Celeron, Pentium II, Pentium III, Pentium IV, Xeon, AMD Athlon, AMD Duron

Hyperthreading und Multi Core Technologie von Xen unterstützt. Bei diesen Systemen kann Xen sogar einzelne oder mehrere CPUs einem Gast zuweisen. Die Unterstützung für PowerPC-Architekturen, die beispielsweise bei Apple Macintosh verwendet wurden, befindet sich momentan noch in der Entwicklung.

Xen kann für seine Gäste unter x86/32 Architekturen bis zu 64GB Hauptspeicher mit Hilfe von Intel PAE[28] verwalten, mit x86/64 Bit Prozessoren sogar bis zu einem Terrabyte. Seit Xen 3.0 wird auch der Zugriff für Gäste auf den *Accelerated Graphics Port* (AGP) ermöglicht, um so unter den Gästen Anwendungen mit Grafikbeschleunigung zu betreiben.

Im System vorhandene Geräte werden nicht direkt von Xen, sondern von dem Managementbetriebssystem in der Domäne 0 eingebunden und verwaltet. Auf dieses wird im Kapitel 2.2.2.1 noch im Detail eingegangen. Gäste können so auf die gesamte Hardware zugreifen, die von dem Betriebssystem in der Domäne 0 unterstützt wird. Alle Betriebssysteme in einer Xen-Umgebung, sowohl das Managementsystem als auch die Gastsysteme, werden in voneinander getrennten Instanzen, den so genannten Domänen, betrieben.

2.2.2 Das Xen-Domänenkonzept

Ein Xen-Virtualisierungssystem besteht aus dem Hypervisor und mehreren Domänen (siehe Abbildung 2.1). Diese Domänen sind in 2 Gruppen gegliedert. Die Domäne 0 stellt die Managementdomäne dar, in die das System beim Start des Hosts bootet. Die Domäne 0 wird auch als privilegierte Domäne bezeichnet, da es nur aus dieser Domäne möglich ist, die nich privilegierten Gastdomänen zu verwalten und Ressourcen zuzuteilen. Die Gastdomänen werden in Xen-

[28] Physical Adressing Extension

Terminologie häufig als Domäne U für *unprivileged*[29] bezeichnet.

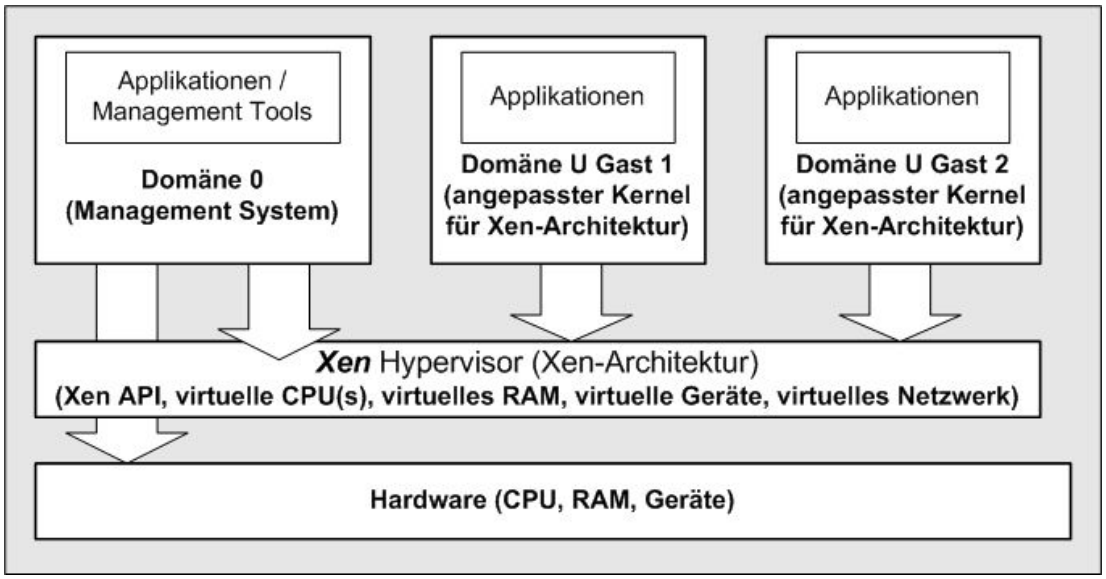

Abbildung 2.1: Xen-Domänenkonzept

2.2.2.1 Domäne 0

Um eine virtuelle Umgebung mit Xen zu erstellen, muss zunächst die Domäne 0 aufgebaut und konfiguriert werden. Die Domäne 0 gliedert sich in 4 Teile:

- mit Xen-Architektur erweiterter Kernel
- Domäne 0 Betriebssystem
- Xen Control Daemon
- Xen Management User Interface

Der Kernel der Xen-Domäne 0 liefert die Funktionalität um überhaupt virtuelle Maschinen betreiben zu können. Da die Xen-Hypervisorerweiterungen momentan

[29] engl. nicht privilegiert

noch nicht zum Standard-Kernel gehören, müssen diese über einen Patch in die Kernel-Quellen für die Domäne 0 kompiliert werden. Viele Linunx-Distributionen wie Fedora, Suse und Debian bieten jedoch auch schon vorgefertigte Kernel-Pakete. Ist der Xen-Kernel auf einem System installiert, kann dieser durch einen Neustart des Systems genutzt werden. Der Xen-Hypervisor-Kernel hat vollen Zugriff auf alle Systemressourcen wie CPU, Speicher und Geräte, da wie zuvor erwähnt die Verteilung der Ressourcen auf die Gäste über die Domäne 0 verwaltet wird und nutzt in einem nicht Hardware-virtualisierten System den Ring 0 des IA-32 Ring Konzeptes (siehe Kapitel 1.4.6).

Die Domäne 0 verhält sich wie ein normales Serverbetriebssystem mit eigenen Bibliotheken, Geräteschnittstellen und Programmen bzw. Prozessen. Die System- und Anwendungsprogramme für die Domäne 0 benötigen, genauso wie die der nicht privilegierten Domänen, keinerlei Anpassungen um unter dem modifizierten Kernel zu laufen. In der Domäne 0 dienen diese Programme und Prozesse sowohl dem Betrieb und dem Management der eigenen Domäne, als auch dem Erstellen und Verwalten der Gäste, sowie der Absicherung der Domäne gegen unbefugten Zugriff.
Des weiteren beinhaltet die Domäne 0 wie erwähnt den *Xen Control Daemon* und das *Xen Management User Interface* zur Verwaltung der virtuellen Umgebung. Diese werden in Kapitel 2.2.3 genauer beschrieben.

2.2.2.2 Domäne U

Der Hauptunterschied zwischen der Domäne 0 und den nicht privilegierten Domänen (*Domäne U*) besteht in der Ansteuerung der Hosthardware. Zwar nutzen nicht privilegierte Domänen einen eigenen Kernel, sie haben jedoch nicht die Möglichkeit wie die Domäne 0 direkt auf die Hardware des Hosts zuzugreifen. Der Kernel einer Domäne U muss zunächst angepasst werden um

auf die paravirtualisierte Xen-Hardwarearchitektur zugreifen zu können, die von der Domäne 0 zur Verfügung gestellt wird (siehe Abbildung 2.1). Wie unter Kapitel 1.4.5 beschrieben muss aus diesem Grund der Kernel einer Domäne U für die Xen-Architektur zunächst übersetzt werden. Wird eine Domäne U dann mit einem modifizierten Kernel gestartet, so kann das Betriebssystem ohne weitere Einschränkungen bzw. Anpassungen genutzt werden. Zwar werden gewisse Ressourcen wie Haupt- und Festplattenspeicher der nicht privilegierten Domäne durch die Domäne 0 zugewiesen und Geräte wie Netzwerkkarten virtuell durch eine Emulation bereitgestellt, die Verwaltung und Nutzung dieser Ressourcen wird jedoch vom Kernel der jeweiligen Domäne U vorgenommen.

2.2.3 Xen-Domänenmanagement

Das Domänenmanagement unter Xen liegt wie zuvor beschrieben in der Domäne 0. Dort befinden sich der *Xen Control Daemon* und das *Xen Management User Interface*.

2.2.3.1 Xen Control Daemon

Der *Xen Control Daemon* ist ein Systemprozess mit dem Namen *xend*, der in erster Linie für das System und Ressourcenmanagement der virtuellen Maschinen und dem Xen-Host zuständig ist. Ein *Daemon* bezeichnet im Zusammenhang mit Unix-basierenden Betriebssystemen einen Prozess, der keine Interaktionen zum Betrieb benötigt, also im Grunde genommen einen eigenständigen Service. Beim Systemstart der Domäne 0 erstellt der Xen Control Daemon eine virtuelle Netzwerkumgebung für die Gäste, reserviert Hauptspeicher und gegebenenfalls dedizierte CPU-Ressourcen für den Host und stellt vor allem Kommunikationsschnittstellen zur Verfügung, um die virtuelle Umgebung zu

verwalten. Diese Kommunikationsschnittstellen werden in erster Linie von dem *Xen Management User Interface*, das im Folgenden näher erläutert wird, genutzt. Es gibt jedoch auch die Möglichkeit einen Webbrowser zur Konfiguration einzusetzen.

2.2.3.2 Xen Management User Interface

Das *Xen Management User Interface* kann in der Domäne 0 über den Befehl *xm* aufgerufen werden und bietet eine Reihe von Funktionen um die Xen-Domänen eines Hosts zu verwalten. Zu diesen Funktionen gehört sowohl das Starten, Pausieren und Stoppen von Domänen, als auch die vollständige Verwaltung von Ressourcen wie virtuelle CPUs, Speicher, Netzwerk und virtuellen Geräten, bis hin zur Erstellung von virtuellen Konsolen, um eine direkte Wartung eines Betriebssystems in einer nicht privilegierten Domäne durchzuführen. Es ist allerdings zu beachten, dass das *Xen Management User Interface* nicht ohne den *Xen Control Daemon* arbeiten kann.[30] Das *Xen Management User Interface* stellt also das Gegenstück zu dem *vzctl* Befehl, der in Kapitel 2.1.2 beschrieben wurde, und den *User Beancounters* (siehe Kapitel 2.1.3.3) aus OpenVZ dar. Die Struktur des xm-Befehls ist wie folgt:

```
xm <subcommand> <domain-id> [OPTIONS]
```

Hier bezeichnet *<subcommand>* einen Befehl wie create (starte), shutdown (stoppe) und console (virtuelle Konsole), die *<domain-id>* den Domänennamen der gewünschten Domäne und *[options]* eventuelle Optionen, die sich auf den *<subcommand>* Befehl beziehen. An dieser Stelle ist anzumerken, dass mit xm

[30] Vgl. [11], xm - Xen management user interface

auch Parameter der Domäne 0, also der Managementdomäne verändert werden können, da diese ja im Grunde auch einen Xen-Gast darstellt.

2.2.3.3 Virtuelle Gerätearchitektur

Wie unter Kapitel 2.2.1 beschrieben, stellt Xen durch Paravirtualisierung seinen Domänen eine virtuelle Hardwarearchitektur zur Verfügung, die auf dem Aufbau der real unterliegenden Hosthardware basiert. Alle virtuellen Geräte unter Xen, also Festplatten, Netzwerkadapter und sonstige werden den Domänen durch eine so genannte *split device driver architecture*[31] zur Verfügung gestellt. Ein virtuelles Gerät wird in dieser Architektur durch zwei, miteinander kooperierende Gerätetreibern erstellt. In der Domäne 0 läuft zunächst ein Treiber, der direkten Zugriff auf die Hardware des Hosts hat und dem zweiten Treiber in einer nicht privilegierten Domäne eine Schnittstelle zu dieser Hardware zur Verfügung stellt. Der Treiber in der nicht privilegierten Domäne wiederum erscheint für den Betriebssystem-Kernel der nicht privilegierten Domäne als virtuelles Gerät, das wie ein physikalisches Gerät angesprochen werden kann. Sobald der nicht privilegierte Kernel eine Anfrage auf das virtuelle Gerät stellt, wird diese von dem nicht privilegierten Treiber über *Shared Memory*[32] an den Treiber in der Domäne 0 gestellt. Dieser prüft nun, ob die Anfrage des nicht privilegierten Kernels nicht zu Problemen bei der Ausführung durch Konflikte oder falsche Adressierung führen kann. Schließlich stellt der Treiber dann die Anfrage direkt an die Hardware bzw. unterbindet diese bei einem Konflikt und kommuniziert an den nicht privilegierten Treiber den Status der gestellten Anfrage, welchen dieser wiederum zurück an den nicht privilegierten Kernel meldet.

[31] Vgl. [1], Xen Interface Manual, Kapitel 8
[32] gemeinsam genutzter Speicher für Interprozesskommunikation

2.2.4 Ressourcenmanagement

Wie zuvor beschrieben wird die Ressourcenzuteilung unter Xen über den *Xen Control Daemon* und das *Xen Management User Interface* durchgeführt. Das Ressourcenmanagement von Xen unterscheidet sich stark von dem das unter OpenVZ zum Einsatz kommt, da Xen seinen Gästen eine virtuelle Hardwarestruktur zu Grunde legt, auf der die Gäste mit eigenen Betriebssystem-Kernel zum Einsatz kommen. Im Folgenden soll auf die Implementierung dieser virtuellen Ressourcen näher eingegangen werden:

2.2.4.1 CPU-Management

Xen setzt für das CPU-Management seiner Gäste einen *Scheduler*[33]-Prozess ein, um CPU-Ressourcen unter dem Aspekt der Fairness auf mehrere Gäste zu verteilen. Fairness spielt in diesem Bereich eine enorm wichtige Rolle, da der Scheduler nicht nur dafür sorgen muss Ressourcen effizient zur Verfügung zu stellen, sondern auch allen aktiven Domänen ihren reservierten Anteil an den Ressourcen zu garantieren. Momentan werden drei Zeitplanungsprogramme von Xen unterstützt: der *Borrowed Virtual Time (BVT) Scheduler*[34], der *Simple Earliest Deadline First (SEDF) Scheduler* und der *Credit Scheduler*, der sich zur Zeit noch in der Entwicklung befindet und später die beiden zuerst genannten Verfahren ersetzen soll.

Zunächst sollte erwähnt sein, dass jedes dieser Scheduler-Verfahren Hostsysteme mit mehreren physikalischen CPUs unterstützt. Physikalische CPUs bezeichnen in diesem Zusammenhang sowohl mehrere CPU-Kerne bzw. Prozessoren in Multi-Prozessor-Maschinen, als auch *Hyperthreading* fähige CPUs, die zwei Threads[35]

[33] engl. Zeitplanungsprogramm
[34] Vgl. [7], Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose schedular
[35] engl. Prozessstränge

parallel ausführen können. Die ersten beiden genannten Scheduler-Verfahren, BVT und SEDF, stellen jeweils einen Algorithmus zur Verfügung, der die Ausführung von Prozesssträngen in einem vordefinierten virtuellen Zeitfenster erlaubt, in dem CPU-Zeit zur Verfügung gestellt wird. Dies bedeutet prinzipiell, dass jeder Prozesstrang sich einen Anteil auf CPU-Zeit im Voraus, also vor seiner Durchführung, reserviert und bei seiner Ausführung die Möglichkeit hat, die reservierte CPU-Zeit zu nutzen. Auf Xen bezogen bedeutet dies, dass die Ressourcen jeder realen CPU im System anteilig für eine Domäne zur Verfügung gestellt werden können. Sind mehrere CPUs im System vorhanden, hat jede Domäne jedoch einen festen Anteil an den ihr zugewiesenen CPUs. Ist eine CPU ausgelastet und eine andere nicht, so ist es mit dem BVT und SEDF-Scheduler nicht möglich eine dynamische Zuordung der phyiskalischen CPU-Ressourcen vorzunehmen. Zwar ist es möglich eine oder mehrere komplette CPUs einer Domäne zuzuweisen, dies ist jedoch für ein Hostsystem mit vielen Domänen unpraktikabel, da somit erstens die Anzahl der Domänen auf die Anzahl der vorhandenen CPUs beschränkt ist, und zweitens keine optimale Ausnutzung aller vorhandenen Ressourcen garantiert werden kann.

Aus diesem Grund wird momentan ein „*Credit Scheduler* Verfahren“ entwickelt, das eine Lastverteilung über mehrere CPUs durchführen kann. Hierbei werden den Domänen so genannte VCPUs (Virtual CPUs) zugeteilt, die keine feste Verbindung mit den physikalischen CPUs im System haben. Benötigt eine Domäne CPU-Zeit, so wird ihre virtuelle CPU einer freien physikalischen CPU im System zugewiesen. Dies geschieht natürlich wieder unter dem Aspekt der Fairness, so dass alle aktiven Domänen ihren reservierten Anteil an CPU-Zeit erhalten.

2.2.4.2 Festplattenmanagement

Wie unter Kapitel 2.2.3.3 beschrieben werden Geräte unter Xen über die geteilte Treiberarchitektur angesprochen. Festplatten für eine nicht privilegierte Domäne können über verschiedene Treiberschnittstellen zur Verfügung gestellt werden. Xen bietet die Möglichkeit sowohl lokale Speicher wie physikalische Festplatten, Partitionen oder einfache Dateien, wie auch Netzwerkspeicher wie NFS[36]- oder SAN[37]-Speicher als virtuelle Festplatten für einen Gast zu nutzen. Eingebunden wird dieser Speicherplatz mit Hilfe des privilegierten Treibers über die Konfigurationsdatei der jeweiligen Domäne, wobei es natürlich möglich ist mehrere Speicherquellen für eine Domäne mit mehr als einer Festplatte einzubinden. Der unpriviliegierte Treiber stellt so seinem zugehörigen Kernel eine Schnittstelle zu den Speicherbereichen in Form eines SCSI[38]-Gerätes zur Verfügung. Dies entspricht einer virtuellen Festplatte. Das eigentliche Management dieses Speicherplatzes, also sowohl die Verwaltung und Nutung des Dateisystems, als auch die Inode und Quota Verwaltung (siehe Kapitel 2.1.3.2), muss der Kernel der nicht privilegierten Domäne übernehmen. Lediglich der Umfang des zur Verfügung gestellten Speichers ist von der Domäne 0 abhängig.

2.2.4.3 Speichermanagement

Da Gäste in Xen einen eigenen Kernel zur Verwaltung einsetzen, übernimmt dieser ebenso wie beim Festplattenmanagement das Management seines zugewiesenen virtuellen Hauptspeichers in seiner Domäne. Zur Einrichtung einer Domäne ist es zunächst nur nötig die maximale Größe des Speichers beim Start der Domäne anzugeben. Über das *Xen Management User Interface* kann dieser jedoch noch zur Laufzeit vergrößert oder verkleinert werden und es ist sogar möglich über

[36]Network File System
[37]Storage Area Network
[38]Small Computer System Interface

einen so genannten *balloon Treiber* in der nicht privilegierten Domäne eine dynamische Anpassung der Größe des Hauptspeichers vorzunehmen.

Der für Xen-Domänen virtuelle Hauptspeicher wird als *Pseudo-Physical Memory* bezeichnet.[39] Jede Domäne bekommt bei diesem Konzept über zwei Tabellen einen Speicherbereich zugewiesen. Die eine Tabelle stellt nur der Domäne eine Speichertabelle zur Verfügung, die wie in einem realen System den Speicherbereich von Null bis zur maximalen Größe beinhaltet. Über diese Tabelle kann also eine Domäne ihren *Pseudo-Physical Memory* adressieren, der sich in Wirklichkeit über den gesamten physikalischen Speicherplatz verteilen kann. Die zweite Tabelle steht dem Xen-Hypervisor zur Verfügung. Dort stehen die Verweise des jeweiligen *Pseudo-Physical Memory* einer Domäne zum physikalisch adressierten Speicher. Wenn also der Kernel einer Domäne Speicher nutzen will, führt der Hypervisor zunächst einen Abgleich seiner Speichertabelle mit der Speichertabelle der Domäne durch, um den physikalischen Speicher für die Operation zu finden. Des weiteren überprüft der Hypervisor, ob der Speicher wirklich der Domäne zugeordnet ist und gibt diesen dann dementsprechend frei oder verweigert den Zugriff. Somit wird auch verhindert aus einer Domäne den Speicher einer anderen Domäne auszulesen oder gar zu beeinflussen.

2.2.4.4 Netzwerkvirtualisierung

Wie auch das Festplattenmanagement wird die Virtualisierung der Netzwerkschnittstellen über die geteilte Gerätetreiberarchitektur (siehe Kapitel 2.2.3.3) realisiert, wobei der Treiber in der privilegierten Domäne als virtuelle Netzwerk-Switch betrieben wird, an den sich Treiber der nicht privilegierten Domänen anhängen können. Die Treiber in nicht privilegierten Domänen bilden also somit virtuelle Netzwerkkarten. Dadurch können wiederum in jeder

[39] Vgl. [1], Xen Interface Manual, Kapitel 3

Domäne U Funktionen wie eine Firewall und Routing für vorhandene virtuelle Netzwerkkarten eingerichtet werden. Der Domäne 0 Treiber kann durch die Emulation des virtuellen Netzwerk-Switch eine einzelne physikalisch vorhandene Netzwerkkarte mehreren nicht privilegierten Domänen zur Verfügung stellen, wobei es natürlich auch möglich ist eine Netzwerkkarte einem Gast exklusiv zuzuweisen.

Das Ressourcenmanagement der Netzwerkvirtualisierung wird über die Domäne 0 gesteuert. Diese kann zusätzlich über Firewallregeln die Herkunft von Netzwerkpaketen validieren, indem über den privilegierten Treiber gesendete Pakete überprüft werden und somit auch eine Überwachung des Netzwerkverkehrs druchführen. Weiterhin ist es so auch möglich über die Domäne 0 zusätzliche Routing-Tabellen für nicht privilegierte Gäste zu definieren.

Die Netzwerkvirtualisierung unter Xen läuft also in zwei Ebenen, der nicht privilegierten Domäne und zusätzlich in der Domäne 0, ab. Ein Vorteil, der sich hieraus ergibt ist eine feinere Konfiguration von virtuellen Netzwerken, da die Domäne 0 definieren kann, welche nicht privilegierten Domänen auf welche Netzwerkbereiche Zugriff haben. Beispielsweise kann man eine Konfiguration so gestalten, das ein Gast die anderen Gäste auf dem Host sieht, jedoch nichts vom externen Netzwerk ausserhalb des Hostsystems.

2.2.5 Installation einer Domäne U

[40] Die Installation eines Gastbetriebssystems in einer nicht privilegierten Domäne ist in mehreren Schritten durchzuführen. Jeder nicht privilegierte Gast in einer Xen-Umgebung besitzt eine eigene Konfigurationsdatei mit Grundeinstellungen.

[40] instdom0

Diese sollte zunächst für einen neuen Gast angelegt werden. Da Xen-Gäste den Großteil der Ressourcenverwaltung selbst übernehmen, umfassen diese Einstellungen in erster Linie allgemeine Konfigurationen und die Bereitstellung von Ressourcen wie:

- Festlegung Gastname
- Pfadangabe des zu nutzenden Domäne U Kernel
- Größe des zu reserviernden Hauptspeichers
- Pfadangabe zu virtuellen Festplatten oder Partitionen
- Pfadangabe zur „Root Partition“
- Bereitstellung von virtuellen Netzwerkadaptern

Im Folgenden Schritt kann mit der Installation von Programmpaketen in die Gastumgebung begonnen werden. Zur Zeit werden für Xen nur wenige Werkzeuge für eine automatisierte Installation einer nicht privilegierten Domäne angeboten. Daher soll hier eine weitestgehend manuelle Vorgehensweise erläutert werden. Wie auch schon bei der Installation von OpenVZ-Gästen erläutert (siehe Kapitel 2.1.4) können aktuelle Installationspakete einer Linux-Distribution für einen Gast am einfachsten über einen Paketmanager, wie den *Yellowdog Updater Modified* (yum), geladen und auch installiert werden. Viele Paketmanager bieten zudem die Möglichkeit eine Installation für eine Change Root Umgebung durchzuführen. Hierbei werden alle benötigten und abhängigen Dateien, dazugehörige Bibliotheken, Pfade und Skripte unterhalb eines Verzeichnis installiert. Als Resultat erhält man beispielsweise eine vollständige Linux-Installation auf einer Festplattenpartition, die zuvor temporär unter */mnt/chroot* eingehangen wurde.

Dieses Vorgehen kann nun ebenso dazu dienen einen nicht privilegierten Xen-Gast zu installieren. Dazu müssen zunächst eine *Root-Partition* für den Gast und gegebenenfalls auch Partitionen unterhalb dieser Root-Partition angelegt und formatiert werden. Diese müssen im Folgenden Schritt inkrementell in ein Verzeichnis unterhalb des Hostsystems eingehangen werden, zum Beispiel in */mnt/xenUroot*. Inkrementell bedeutet hier: zuerst wird die Root-Partition unter */mnt/xenUroot* eingehangen, dann gegebenenfalls weitere wie eine Var-Partition unter */mnt/xenUroot/var*. Ist dies geschehen, kann über den Paktemanager eine Installation der Betriebssystempakete erfolgen. Schließlich können die Partitionen des neuen Gastes wieder aus dem Hostsystem ausgehangen werden. Im abschließenden Schritt kann nun der neue Gast über den folgenden Befehl gestartet werden:

```
xm create -c <gastname>
```

Dieser Aufruf des *Xen Management User Interface* startet den nicht privilegierten Gast und öffnet direkt eine Konsole, die den Bootvorgang anzeigt und bei erfolgreichem Start eine Eingabeaufforderung des Gastsystems liefert.

2.2.6 Spezielle Funktionen

Xen bietet wie auch schon OpenVZ spezielle Funktionen in der Verwaltung von Hostsystemen. So ist es, wie auch schon für OpenVZ unter Kapitel 2.1.5 ausgeführt, möglich einen Snapshot oder eine Live Migration der Xen Umgebung durchzuführen. Über den Aufruf des nachfolgenden Befehls ist es beispielsweise möglich eine Live Migration eines Systems durchzuführen:

```
xm migrate --live <gastname> <zielhost>
```

Dies setzt natürlich, wie schon für OpenVZ, voraus, dass sich zum einen der Zielhost im selben Netzwerk wie der Quellhost befindet, und dass eventuell benötigte Ressourcen wie Festplattenspeicher für den Gast auf dem Zielhost erreichbar bzw. verfügbar sind.

2.2.7 Erweiterungen für Xen

Neben den direkt implementierten Funktionen unter Xen gibt es auch eine Reihe von Projekten, die sich mit der Entwicklung von Erweiterungen für Xen befassen. So bietet beispielsweise *XenSource*[41] eine kommerzielle Variante von Xen mit dem Namen *XenEnterprise* an, die speziell für den Einsatz in Unternehmen ausgerichtet ist. XenEnterprise soll beispielsweise wie auch Virtuozzo eine Reihe von erweiterten Managementwerkzeugen liefern, die den Einsatz und die Überwachung von Xen-Hostsystemen und Gästen erleichtern sollen.

Ein anderes freies Projekt liefert momentan mit *EisXen*[42] eine Linux-Distribution zur Verfügung, die es sich zum Ziel gesetzt hat, ein möglichst kleines Linux-Betriebssystem mit Xen-Implementierung zur Verfügung zu stellen, das einen hohen Grad an Sicherheit bietet und sich einfach verwalten lässt. So bietet EisXen ein automatisches Setup eines Domäne 0 Servers, in der die privilegierte Domäne weitestgehend gegen Angriffe von außen abgesichert ist, da nur zum Betrieb notwendige Pakete der Domäne 0 bereitgestellt werden. Alle in irgendeiner Form kritischen Prozesse sollen auf nicht privilegierte Domänen verteilt werden.

Das *XenMan*-Projekt[43] liefert ein freies Konfigurationswerkzeug, das eine grafische Verwaltung von virtuellen Maschinen ermöglicht. Somit richtet sich

[41] http://www.xensource.com/
[42] http://www.xen-buch.de/geteisxen.html
[43] https://sourceforge.net/projects/xenman/

XenMan sowohl an Administratoren, als auch unerfahrene Benutzer, die einen möglichst einfachen und komfortablen Weg für die Installation von Xen-Gästen nutzen möchten. Das Projekt befindet sich jedoch noch am Anfang und bietet daher momentan noch eingeschränkte Funktionalität.

Insgesamt ist also festzuhalten, dass sich eine ganze Reihe von, sowohl Open Source als auch kommerziellen, Projekten um die Virtualisierungslöung Xen gebildet haben, um den Umgang mit Xen zu erweiten und zu vereinfachen.

3 Exemplarische Implementierung und Benchmarks

Zuvor wurde der Aufbau und die Funktionsweise der Virtualisierungssysteme OpenVZ und Xen ausführlich dargestellt. Im Folgenden soll nun eine exemplarische Implementierung des jeweiligen Verfahrens vorgestellt werden. Für jede Implementierung wird zunächst die zugehörige Planung und das Konzept beschrieben. Darauf aufbauend wird die Einrichtung und Konfiguration mit anschließender Inbetriebnahme der jeweiligen Umgebung dargestellt. Die Implementierung soll dazu dienen eine Analyse der Performance von beiden Virtualisierungslösungen durch Benchmarks durchzuführen. Diese sollen eine abschließende Bewertung der beiden Verfahren in Bezug auf Eignung, Implementierungsaufwand und Performance liefern.

3.1 Grundlagen der Implementierung

Da es das Ziel der Implementierung ist, die beiden Virtualisierungsverfahren möglichst aussagekräftig vergleichen zu können, müssen zunächst einige Rahmenbedingungen getroffen werden. Um eine verwertbare Aussage durch Benchmarks zu erhalten, ist es zunächst notwendig die Implementierung für beide Verfahren weitestgehend einheitlich zu gestalten. Als Betriebssystembasis für die Host- und Gästeumgebung wurde *Fedora Core 5* ausgewählt, da es zum einen sowohl von OpenVZ als auch von Xen voll unterstützt wird und zum anderen als offizielle Test- und Basisplattform für zukünftige Implementierungen des kommerziellen *Red Hat Enterprise Linux 5*[1] genutzt wird.

Für den Aufbau einer Testumgebung wurde das folgende Hardwaresystem zur Verfügung gestellt:

- IBM xSeries 345
- 2x 2,8 Ghz Intel XEON Prozessor
- 2GB PC2100 DDR RAM
- Dual Ultra320 SCSI controller
- 2x 36GB HDD (RAID 1)
- Dual Gigabit Ethernet (100MBit genutzt)

Um eine Testumgebung zu erstellen, die sowohl Aussagekraft in Bezug auf eine optimale Ressourcenausnutung als auch den nötigen Implementierungsaufwand darstellen kann, wurde jeweils für OpenVZ und Xen ein Hostsystem mit drei Gästen eingerichtet. Um die Ressourcenausnutzung analysieren zu können, sollen

[1] http://www.redhat.com/rhel/

die Benchmarkverfahren die allgemeine Performance eines Gasts im Bezug auf CPU-Leistung, Interprozesskommunikation, Festplattenzugriff und über die Performace des virtuellen Netzwerkes wiedergeben. Auf den Gästen wurden dementsprechend, neben den standardmäßig installierten Programmpaketen der Fedora Core 5 Distribution, jeweils die *GNU Compiler Collection*[2] (gcc) und der *Apache Webserver*[3] eingerichtet. Auf deren genauen Bezug zu den Benchmarkverfahren wird in Kapitel 3.4 eingegangen.

Der Host und die Gäste wurden zusammen auf einer Festplatte in jeweils seperaten Festplattenpartition installiert. Dabei wurde der Speicherplatz der Gäste mit dem *Logical Volume Manager* (LVM) (siehe Kapitel 3.1.0.1) zur Verfügung gestellt, um eine möglichst hohe Skalierbarkeit des Speicherplatzes für die Gastumgebungungen zu ermöglichen. Des weiteren wurde in diesem Bezug das *third extended filesystem* (ext3) genutzt. Das ext3-Dateisystem bietet zwar nicht das performanteste[4] Dateisystem, es bildet jedoch den Standard als Dateisystem unter diversen Linux-Distributionen, eingeschlossen Fedora Core 5.

[2] http://gcc.gnu.org/
[3] http://httpd.apache.org/
[4] Vgl. http://linuxgazette.net/102/piszcz.html und
http://www.linux-magazin.de/Artikel/ausgabe/2002/01/jfs/jfs.html

3.1.0.1 Der Logical Volume Manager (LVM)

Die Wahl für Partition der Gäste, LVM zu nutzen, liegt hingegen auf der Hand. Da die Menge der Gäste in einer virtuellen Umgebung häufig variiert, weil neue Gäste hinzukommen und alte entfernt werden, ist es wichtig, den vorhandenen Speicherplatz für die Gäste möglichst dynamisch zu gestalten. Speziell LVM2[5] ermöglicht es Plattenplatz, in Form von logischen Volumen, während der Laufzeit dynamisch zu vergrößern, verkleinern, hinzuzufügen und zu entfernen. LVM stellt dabei eine logische Schicht zwischen Festplatte und Dateisystem zur Verfügung.

Bei LVM wird ein Teil einer Festplatte oder auch die gesamte Festplatte als LVM Laufwerk durch Partitionierung als LVM Bereich gekennzeichnet. Dieser Teil stellt ein so genanntes *Physical Volume* dar. Ein oder mehrere physikalische Volumen können zu einer Volumengruppe *Volume Group* zusammengefasst werden (siehe Abbildung 3.1). So ist es beispielsweise möglich den gesamten Speicherplatz mehrerer Festplatten zu einer großen Partition zusammenzufassen. Diese Zusammenfassung zu einer Partition geschieht durch das Erstellen von sogenannen *Logical Volumes* in der Volumengruppe. Diese logischen Volumen können nun wie eine normale Partition mit einem Dateisystem, im Fall der Implementierung ext3, formatiert und unter dem Betriebssystem eingehangen werden.

LVM bietet nicht ausschließlich Vorteile. Es steigt vor allem die Fehlerwahrscheinlichkeit, sobald mehrere physikalische Volumen im Einsatz sind. So ist es zu empfehlen LVM durch einem *Redundant Array of Inexpensive Disks* (RAID) Verbund abzusichern, da unter dem Einsatz von mehreren physikalischen Volumen Datenverlust auftreten kann, sobald nur eine Platte in dem Verbund ausfällt. Da eventuell logische Volumen über zwei oder mehr physikalische

[5] http://sourceware.org/lvm2/

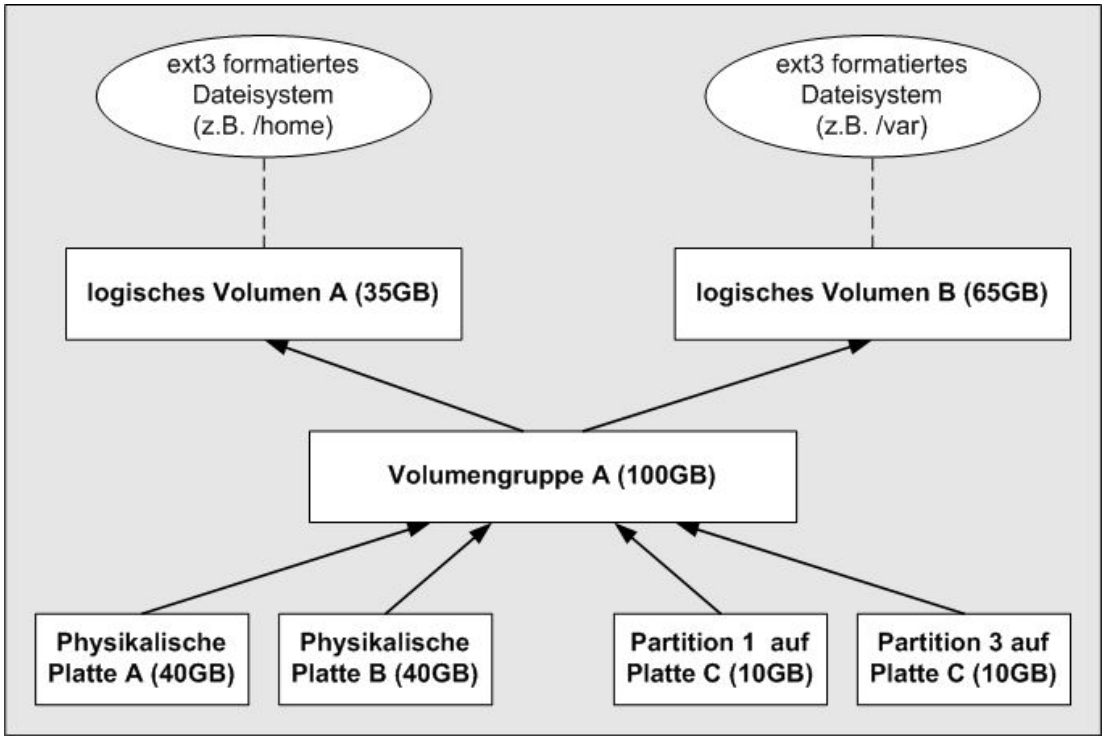

Abbildung 3.1: Logical Volume Management

Volumen verteilt sind, wäre ein Datenverlust um so verheerender.

Die Vorteile von LVM überwiegen jedoch und lassen sich wie folgt zusammenfassen:

- nahezu kein Geschwindigkeitsverlust
- keine Einschränkung von Partitionen durch die Größe einzelner Festplatten
- bis zu 256 Logical Volumes
- dynamische Größenänderung der Volumengruppe durch Einhängen von Physical Volumes
- dynamisches Erstellen / Löschen von Partitionen

Sowohl für OpenVZ als auch für Xen ist der Einsatz von LVM praktikabel. Da in einer OpenVZ-Umgebung alle Gäste im Normalfall unterhalb eines Verzeichnisses liegen (/vz/private/<vps-id>), bietet es sich an für dieses Verzeichnis LVM zu nutzen, da somit beispielsweise der Speicherplatz bei Bedarf sehr einfach vergrößert werden kann. Einen noch größeren Vorteil bietet LVM bei Xen, da hier für jede virtuelle Festplatte ein eigenes logisches Volumen verwendet werden kann. Dies vereinfacht den Wartungsaufwand enorm, da bei Bedarf innerhalb kürzester Zeit neue virtuelle Festplatten erstellt und nicht mehr benötigte gelöscht werden können.

3.2 Die OpenVZ-Testumgebung

3.2.1 Installation der OpenVZ-Umgebung

Die Installation der OpenVZ-Virtualisierungsumgebung gliedert sich in zwei Abschnitte. Zunächst muss das Hostsystems mit dem OpenVZ-Kernel und OpenVZ-Werkzeugen eingerichtet werden. Des weiteren muss dort ebenfalls der Template-Cache aufgebaut werden mit dem die Einrichtung der OpenVZ-Gäste ermöglicht wird. Im zweiten Abschnitt können schließlich die OpenVZ-Gäste eingerichtet werden.

3.2.1.1 Einrichtung OpenVZ-Host

Nachdem der Host mit einer Standardinstallation des Fedora Core 5 Betriebssystems installiert wurde, müssen für die Einrichtung von OpenVZ zunächst alle notwendigen Programmpakete geladen und installiert werden. Diese werden sowohl im Quellcode als auch Distributionen wie Fedora Core 5 direkt auf der Homepage des OpenVZ Projekts[6] zur Verfügung gestellt. Die Pakete sind in drei Guppen aufgeteilt. Die Gruppe *Kernel* enthält den modifizierten Linux-OpenVZ-Kernel mit den Erweiterungen für die Virtualisierung.

[6] http://openvz.org/download/

Die Gruppen *Utilities* und *Templates* enthalten jeweils Werkzeuge, Skripte und Konfigurationsdateien für die Einrichtung und Bedienung der OpenVZ-Umgebung und den Template-Caches zur Erstellung von Virtual Private Servern. Für diese drei Gruppen wurden die folgenden Pakete für die Installation der Testumgebung verwendet:

- **Kernel**
  kernel-smp-2.6.16-1.2111_FC5.026test012.i686.rpm
- **Utilities**
  vzctl-3.0.11-1.i386.rpm
  vzctl-lib-3.0.11-1.i386.rpm
  vzquota-3.0.8-1.i386.rpm
- **Templates**
  vzpkg-2.7.0-18.noarch.rpm
  vzyum-2.4.0-11.noarch.rpm
  vzrpm44-4.4.1-22.5.i386.rpm
  vzrpm44-python-4.4.1-22.5.i386.rpm
  vztmpl-fedora-core-5-2.0-2.i386.rpm

Nach der Installation der Pakete durch den *Red Hat Package Manager* (RPM) über den nachfolgenden Aufruf steht die OpenVZ-Umgebung bereits weitestgehend für die Installation von Gästen bereit:

```
rpm -ivh <paketname>
```

Listing 3.1 soll eine kurze Übersicht über die wichtigsten OpenVZ-Konfigurationsdateien geben[7].

[7] Vgl. [3], OpenVZ User's Guide, Kapitel 9 - Configuring OpenVZ

```
/etc/sysconfig/vz           - globale OpenVZ Konfigurtion
/etc/sysconfig/vz-scripts/ - Konfigurationsdateien der Gäste
/etc/sysctl.conf            - Kernel Parameter
```

Listing 3.1: Die wichtigsten OpenVZ-Dateisystempfade

Während die Datei /etc/sysconfig/vz zunächst keine zwingenden Anpassungen erfordert, muss die Datei /etc/sysctl.conf um die Parameter[8] im Listing 3.2 ergänzt werden und mit dem Systemaufruf *sysctl -p* übernommen werden.

```
# Weterleitung von Netzwerkpaketen erlauben
net.ipv4.ip_forward = 1

# Abschaltung des Address Resolution Protocol (ARP)
# ARP dient der Zuordnung von Internetadessen zu Hardwareadressen
net.ipv4.conf.default.proxy_arp = 0

# Aktivierung von "source route verification"
net.ipv4.conf.all.rp_filter = 1

# Aktivierung des magic-sysrq key
# dient der direkten Ausführung von Kernel Befehlen
kernel.sysrq = 1

# Deaktivierung des Internet Control Message Protocol (ICMP)
# für alle Netzwerkadapter
net.ipv4.conf.default.send_redirects = 1
net.ipv4.conf.all.send_redirects = 0
```

Listing 3.2: Konfiguration der /etc/sysctl.conf

Ist diese Basiskonfiguration durchgeführt, kann das System mit dem OpenVZ-Kernel neu gestartet werden. Abschließend sollte nun der Template-Cache (siehe Kapitel 2.1.3.5) für Fedora Core 5 aktualisiert werden. Ein Aufruf des Befehls *vzpkgcache* erstellt für alle installierten Template-Metadaten einen Template-Cache. In diesem Fall sind die Metadaten für Standardinstallation von Fedora Core 5 durch das Paket vztmpl-fedora-core-5-2.0-2.i386.rpm installiert worden. Wie in Kapitel 2.1.3.5 beschrieben werden für den Template-Cache aktuelle Pa-

[8] Vgl. [3], OpenVZ User's Guide, Kapitel 3 - Installing OpenVZ Software

kete für die entsprechende Distribution über den Paketmanager *yum* geladen und in einem temporären Virtual Private Server istalliert. Dieser wird schließlich in eine Template-Datei mit dem Namen *fedora-core-5-i386-default.tar.gz* archiviert. Im System installierte Templates lassen sich über den Aufruf *vzlist* anzeigen.

3.2.1.2 Einrichtung OpenVZ-Gast

Aus dem erstellten Template lässt sich nun mit dem nachfolgenden Schritten in wenigen Minuten ein neuer OpenVZ VPS einrichten und installieren:

```
vzctl create <vps-id> --ostemplate fedora-core-5-i386-default \
--config vps.basic
vzctl set <vps-id> --hostname <name> --save
vzctl set <vps-id> --ipadd <ip> --save
vzctl set <vps-id> --userpasswd root:<password>
vzctl set <vps-id> --onboot yes --save
```

Die Bedeutung dieser Aufrufe ist relativ einfach. Es wird ein neuer VPS mit einer vorgegebenen ID (hier *101, 102 und 103*) aus dem zuvor erstellten Template *fedora-core-5-i386-default* mit den vordefinierten UBC-Parametern aus emph/etc/sysconfig/vz/vps.basic.conf erzeugt. Die gesamte Konfiguration für den neuen VPS wird im selben Schritt unter /etc/sysconfig/vz/<vps-id>.conf abgespeichert. Der Server erhält einen frei wählbaren Namen *–hostname <name>* und IP-Adresse *<ip>*. Des weiteren sollte ein Passwort für den VPS-Systembenutzer root gesetzt werden *–userpasswd root:<password>*. Schließlich wird veranlasst, dass der VPS bei jedem Neustart des Hostsystems ebenfalls gestartet wird. Der Parameter *–save* dient jeweils dazu, dass Änderungen direkt übernommen werden. Weitere optionale Einstellungen für die Einrichtung eins OpenVZ-VPS lassen sich aus der *OpenVZ User's Guide*[3] entnehmen. Die benannten Aufrufe wurden dementsprechend für jeden zu erstellenden VPS-Server ausgeführt.

3.2.2 Inbetriebnahme

Über den nachfolgenden Befehle können abschließend die VPS-Gäste gestartet werden:

```
vzctl start <vps-id>
```

Ein Login auf dem Gast sollte nun zunächst über OpenVZ mit dem nachfolgenden Aufruf erfolgen um eventuelle Einstellungen des Servers anzupassen und Dienste wie den Apache-Webserver zu konfigurieren und aktivieren.

```
vzctl enter <vps-id>
```

Sollte es nötig sein weitere Programmpakete einzuspielen, kann dies über den nachfolgenden Befehl durchgeführt werden:

```
vzyum <vps-id> <yum-parameter>
```

Die genaue Syntax für mögliche Parameter von *yum* lässt sich aus den entsprechenden Linux Manual Seiten über den Aufruf *man yum* entnehmen.

3.3 Die Xen-Testumgebung

3.3.1 Installation der Xen-Umgebung

Da das zur Verfügung gestellte Testsystem nicht über die CPU-Erweiterungen für Hardwarevirtualisierung verfügt, wurde die Xen-Implementierung ausschließlich im Modus der Paravirtualisierung betrieben. Dementsprechend konnten die Gastsysteme nur mit einem modifiziertem Kernel betrieben werden. Da sowohl die Xen-Werkzeuge als auch Kernel-Erweiterungen Teil der Fedora Core 5 Distribution sind, wurden diese direkt für die Testumgebung genutzt.

Ebenso wie für den Aufbau der OpenVZ-Virtualisierungsumgebung gibt es zwei wesentliche Schritte um eine Xen-Umgebung zu erstellen. Zunächst muss das Hostbetriebssystem installiert werden, das am Ende die Domäne 0 bildet. Für die Implementierung der Testumgebung wurde hier ebenfalls eine Standard Fedora Core 5 Betriebssysteminstallation durchgeführt. Es muss lediglich darauf geachtet werden, im System genügend freien Festplattenspeicher für die Gäste zu reservieren. Dies wurde durch die Zuweisung von Speicher in einer dedizierten Volumengruppe für die nicht privilegierten Gäste implementiert.

3.3.1.1 Einrichtung Domäne 0

Sobald das Betriebssystem installiert ist, kann damit begonnen werden dieses zu einer Domäne 0 umzukonfigurieren. Der Aufwand hierzu ist zunächst relativ gering, da, wie zuvor erwähnt, für Fedora Core 5 alle notwendigen Pakete direkt zur Verfügung gestellt werden. Es handelt sich um die folgenden Pakete, die über den Paketmanager *yum*, der auch dafür sorgt, dass eventuelle Abhängigkeiten berücksichtigt werden, geladen und installiert werden können.

- **Kernel**
  kernel-xen0-2.6.17-1.2174_FC5.i686.rpm
  kernel-xenU-2.6.17-1.2174_FC5.i686.rpm
- **Werkzeuge und Bibliotheken** xen-3.0.2-3.FC5.i386.rpm
  bridge-utils-1.0.6-1.2.i386.rpm
  libvirt-0.1.1-1.FC5.i386
  libvirt-python-0.1.1-1.FC5.i386.rpm

Über den nachfolgenden Aufruf können die Pakete installiert werden:

```
yum install <paketname>
```

Die beiden Kernel-Pakete liefern den Kernel für die Domäne 0 und für die nicht privilegierten Domänen. Der xenU-Kernel sollte sowohl in jeder DomäneU als auch in der Domäne 0 installiert werden, da dieser in der Domäne 0 genutzt wird einen Gast zu starten, aber auch zusätzlich nötige Kernel-Systembibliotheken in der nicht privilegierten Domäne zur Verfügung stellen muss.

Das Xen-Paket umfasst in erster Linie den Xen Control Daemon, das Xen Management User Interface, Konfigurationsdateien und weitere Werkzeuge. Des weiteren werden *libvirt*-Pakete benötigt, die eine Schnittstelle für die Kommunikation zwischen Xen und den Domänen ermöglichen. Die *bridge-utils*-Pakete stellen letztendlich die Komponenten für die Netzwerkvirtualisierung zur Verfügung.

Nachdem die Installation durchgeführt wurde sollte zunächst der Xen Management Daemon konfiguriert[9] werden. Die zugehörige Konfigurationsdatei befindet sich unter:

[9] Vgl. [2], Xen User Manual, Kapitel 4.1.2 - Configuring Xend

```
/etc/xen/xend-config.sxp
```

Das Listing 3.3 zeigt eine Minimalkonfiguration mit den wichtigsten Parametern. So sollte zunächst eine Auswahl getroffen werden, welche CPUs im System von der Domäne 0 genutzt werden dürfen (0 bedeutet alle vorhandenen) und wieviel Hauptspeicher der Domäne 0 permanent zugewiesen werden. Des weiteren wird hier auch die Konfiguration für die Netzwerkvirtualisierung festgelegt. So sollte definiert werden welche physikalische Netwerkkarte (*<device>*) im Hostsystem genutzt wird, um einen virtuellen Switch mit dem Namen *xen-br0* anzulegen (siehe Kapitel 2.2.4.4). Eine Übersicht über alle vorhandenen Parameter liefern die *Linux Manual-Seiten* über den Aufruf *man xend-config.sxp* und das *Xen User Manual*[2].

```
-*- sh -*-

#
# Xend Konfigurations Datei.
#

# Zugewiesene CPUs
(dom0-cpus 0)

# Minimale Speicherzuteilung zur Domäne 0
(dom0-min-mem 256)


# Einstellungen für Bridge Netzwerk
(network-script 'network-bridge bridge=xen-br0 netdev=<device>')
(vif-script 'vif-bridge')
```

Listing 3.3: Konfiguration von xend für Testumgebung

Sobald die Konfiguration von xend abgeschlossen ist, kann das Hostsystem mit dem Domäne 0 Kernel neu gestartet werden.

3.3.1.2 Einrichtung Domäne U

Ist das Hostsystem eingerichtet und Xen gestartet, kann damit begonnen werden Xen-Gäste zu erstellen. Dazu ist es zunächst nötig, eine Konfiguration des Gasts zu erstellen. Die Konfigurationsdateien für die virtuellen Maschinen werden, wie die für den Xen Management Daemon, unter */etc/xen/* abgelegt. Dort befinden sich unter anderem auch einige Beispiele, wobei die Konfiguration relativ simpel ist.[10] Das nachfolgende Listing 3.4 zeigt die Konfiguration eines Gasts der für die Testumgebung eingerichtet wurde:

```
## Konfiguration für Virtuelle Machine xenU_1
kernel ="/boot/vmlinuz-2.6.17-1.2174_FC5xenU"
root = "/dev/sda1 ro"
memory = 256
name = "xenU_1"
disk = ['phy:VolGroup00/xenU_1.1,sda1,w',
        'phy:VolGroup00/xenU_1.2,sda2,w']

#
# Netzwerk Konfiguration
#
vif = [ 'mac=00:00:00:00:00:01, bridge=xen-br0' ]
ip="192.168.0.2"
netmask="255.255.255.0"
gateway="192.168.0.1"
hostname="xenU_1"
```

Listing 3.4: Konfiguration von xend für Testumgebung

Die Parameter im ersten Teil dieser Konfiguration legen zunächst die allgemeine Konfiguration des Gasts *xenU_1* fest. Zunächst wird definiert welcher Kernel für den Gast genutzt werden soll, wobei zu beachten ist, dass die Pfadangabe sich hier noch auf eine Kernel-Datei im Hostsystem bezieht, die dort zuvor installiert wurde (siehe. Kapitel 3.3.1.1). Des weiteren wird hier festgelegt welche virtuelle Festplatte später das Root-Dateisystem beinhalten wird, und wieviel Speicher dem Gast permanent zugewiesen wird. Schließlich wird definiert mit welchem

[10] Vgl. [2], Xen User Manual, Kapitel 5.1 - Configuration Files

Namen der Gast unter Xen verwaltet wird und welcher Speicherplatz dem Gast in Form von virtuellen Festplatten bereitgestellt wird. In diesem Fall werden die beiden logischen Volumen *xenU_1.1* und *xenU_1.2* aus der Volumengruppe *VolGroup00* dem Gast jeweils als SCSI-Gerät *sda1* und *sda2* zur Verfügung gestellt.

Der zweite Teil der Konfiguration bezieht sich ausschließlich auf die Einrichtung des virtuellen Netzwerkes. Zunächst wird festgelegt wie viele Adapter dem Host bereitgestellt werden sollen. Die weiteren Einstellungen beziehen sich dann jeweils auf einen Adapter. So muss festgelegt werden, welche Hardwareadresse dem Adapter zugewiesen werden soll und an welchen virtuellen Switch dieser angeschlossen wird. Des weiteren kann hier IP, Netzmaske, Gateway und Hostname definiert werden.

Sobald die Konfiguration abgeschlossen ist, kann mit der Installation des Gasts begonnen werden. Für das Beispiel muss hierzu zunächst die LVM Volumengruppe mit den beiden logischen Volumen angelegt werden. [11] Im nächsten Schritt müssen die beiden logischen Volumen mit einem Dateisystem formatiert werden. In diesem Beispiel soll xenU_1.1 als Root-Dateisystem mit ext3-Formatierung und xenU_1.2 als Swap[12]-Speicherbereich für den Gast dienen:

```
mkfs.ext3 /dev/VolGroup00/xenU_1.1
mkswap /dev/VolGroup00/xenU_1.2
```

[11] Vgl. http://www.tldp.org/HOWTO/LVM-HOWTO/ - Kapitel 11
[12] engl. Umlagerung

Um das Volumen xenU_1.1 nun mit einem Gastbetriebssystem zu bespielen, muss dieses zunächst im Hostsystem eingehangen werden:

```
mkdir /tmp/xenU_1
mount /dev/VolGroup00/xenU_1.1 /tmp/xenU_1
```

Da der Paketmanager *yum* zwar eine Installation in diesem Installationsverzeichnis durchführen kann, aber für gewisse Pakete Zugriff auf Geräte benötigt, ist es notwendig diese manuell anzulegen bzw. einzubinden:

```
mkdir /tmp/xenU_1.1/dev
for i in console null zero random urandom ; do /sbin/MAKEDEV -d \
/tmp/xenU_1.1/dev -x $i ; done

mkdir /tmp/xenU_1.1/proc
mount -t proc none /tmp/xenU_1.1/proc
```

Darüberhinaus müssen die virtuellen Festplatten dem Gast über eine sogenannte fstab-Datei bekannt gemacht werden. Diese muss für das dargestellte Beispiel den nachfolgenden Inhalt aufweisen und in das Verzeichnis /tmp/xenU_1.1/etc kopiert werden:

```
/dev/sda1       /           ext3    defaults        1 1
/dev/sda3       swap        swap    defaults        0 0
none            /dev/pts    devpts  gid=5,mode=620  0 0
none            /dev/shm    tmpfs   defaults        0 0
none            /proc       proc    defaults        0 0
none            /sys        sysfs   defaults        0 0
```

Die Datei definiert die Eigenschaften der beiden virtuellen Festplatten und gibt weitere systemspezifische Parameter vor.

Nun kann mit der eigentlichen Betriebssysteminstallation begonnen werden. Über nachfolgenden Aufruf wird eine vollständige Fedora Core 5 Web-Serverumgebung in das Verzeichnis /tmp/xenU_1.1, also in das Root-Dateisystem des Gasts, installiert. Der zweite Aufruf ist notwendig, um die Bibliotheken des nicht privilegiertem Kernel in der virtuellen Umgebung zur Verfügung zu stellen:

```
yum --installroot=/tmp/xenU_1.1 groupinstall -y web-server
yum --installroot=/tmp/xenU_1.1 install -y kernel-xenU-2.6.17-1.2174
```

Zum Abschluss ist es möglich über den chroot-Befehl in das Gastsystem zu wechseln, um dort eventuelle Einstellungen wie beispielsweise das Setzen eines Passworts für den Root-Benutzer oder die Festlegung der Netzwerkeinstellungen vorzunehmen:

```
chroot /tmp/xenU_1.1
```

Damit ist die Installation des Gastes abgeschlossen.

3.3.2 Inbetriebnahme

Die Inbetriebnahme des Gastes erfolgt abschließend über den Aufruf:

```
xm create -c xenU_1
```

Der Aufruf startet das System und liefert direkt eine virtuelle Konsole, auf der der Startvorgang verfolgt werden kann. Das System hat von hier an die Eigenschaften einer vollständigen Betriebssysteminstallation.

3.4 Benchmarks

Nach Abschluss der Installation war es möglich die beiden Testumgebungen mit Hilfe von Benchmarkverfahren auf ihre Performance und Ressourcenverteilung hin zu testen. Der Begriff Benchmark bezeichnet in diesem Zusammenhang einen Maßstab für den Vergleich von Leistungen. Im Zuge der Analyse die unter Kapitel 2 vorgestellt wurde, ergaben sich die folgenden Schwerpunkte die durch einen Benchmark der virtuellen Maschinen abgedeckt werden müssen:

- CPU-Auslastung
- Interprozesskommunikation
- Festplattenzugriff
- Netzwerkperformance

In diesem Zusammenhang wurden zwei Benchmarkverfahren ausgewählt, die einen Vergleich dieser Leistungsmerkmale ermöglichen. Eine allgemeine Übersicht von diversen verfügbaren Benchmarkverfahren findet sich unter http://lbs.sourceforge.net/.

3.4.1 Unixbench

Die Auswahl des *Unixbench 4.1.0*[13] Benchmark ergab sich aus einer Reihe von Tests mit diversen Benchmarkprogrammen, die für Linux-Serverumgebungen verfügbar sind. Unixbench ist eine sogenannte Benchmark-Suite, die eine Sammlung von Testverfahren beinhaltet, um die unterschiedlichsten Aspekte eines Servers auf die Performance hin zu testen. So werden sowohl Tests durchgeführt, die einen

[13]http://www.tux.org/pub/tux/benchmarks/System/unixbench/

Vergleich der CPU-Performance ermöglichen, als auch die Interprozesskommunikation analysieren und schließlich eine Angabe über den Datendurchsatz beim Kopieren, Lesen und Schreiben von Dateien ermöglichen.

3.4.1.1 Unixbench Benchmarkergebnisse

Im Folgenden werden die Ergebnisse des Unixbench 4.1.0 Benchmarkverfahren aufgezeigt und erläutert. Hierzu ist zunächst anzumerken, das Unixbench die Möglichkeit bietet Ergebnisse eines Testlaufs als Basiswert für einen Vergleich zu anderen Testläufen zu nutzen. Dies wurde für die hier beschriebenen Tests dazu genutzt zunächst einen Benchmark über die nicht virtualisierte Betriebssystemumgebung durchzuführen. Durch Einsetzen der gemessenen Ergebnisse als Basiswert, stellten sich diese als normierter Wert 10 dar. Alle nachfolgenden Tests für die Gäste in virtuellen Umgebungen beziehen sich also auf den Basiswert 10, der für das nicht virtualisierte Hostsystem ermittelt wurde. Da sich durch die Durchführung von mehreren Testläufen zeigte, dass die Testergebnisse geringen Schwankungen unterliegen, wurden die Ergebnisse durch Abrundung normalisiert.

Die in Abbildung 3.2 dargestellten Ergebnisse zeigen, dass Gäste sowohl unter OpenVZ als auch Xen mit nahezu ungemindeter CPU-Leistung betrieben werden können. Sowohl durch Virtualisierung auf Betriebssystemebene als auch Paravirtualisierung steht die volle CPU-Leistung zur Verfügung und wird nicht durch das Virtualisierungssystem gemindert.

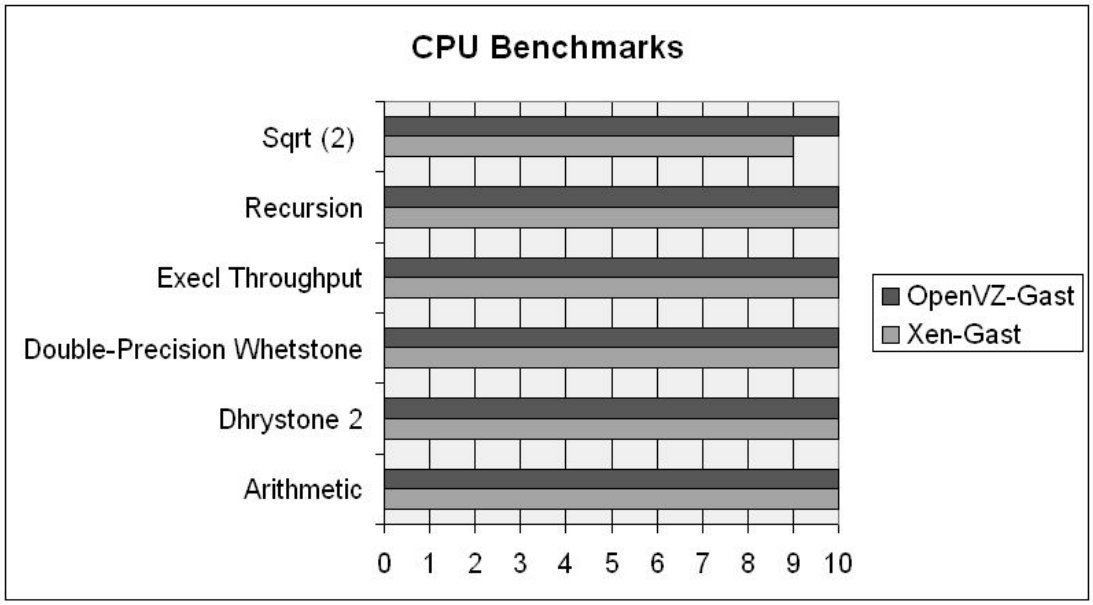

Abbildung 3.2: CPU Benchmarks

Dies gilt jedoch nicht für die Benchmarkergebnisse der getesteten Interprozesskommunikation, die in Abbildung 3.3 dargestellt sind. Hier wird deutlich, dass die IPC-Leistung von OpenVZ deutlich höher als die von Xen ist, da Gäste in OpenVZ direkt mit dem Host-Kernel kommunizieren können. In Xen sind die Abweichungen vermutlich darauf zurückzuführen, dass Prozesse in Gästen zunächst mit dem Gast-Kernel kommunizieren, der dann erst über die virtualisierte Xen-Architektur die Kommunikation zu anderen Prozessen ermöglicht. OpenVZ bietet demgegenüber nahezu keine Verluste an Performance.

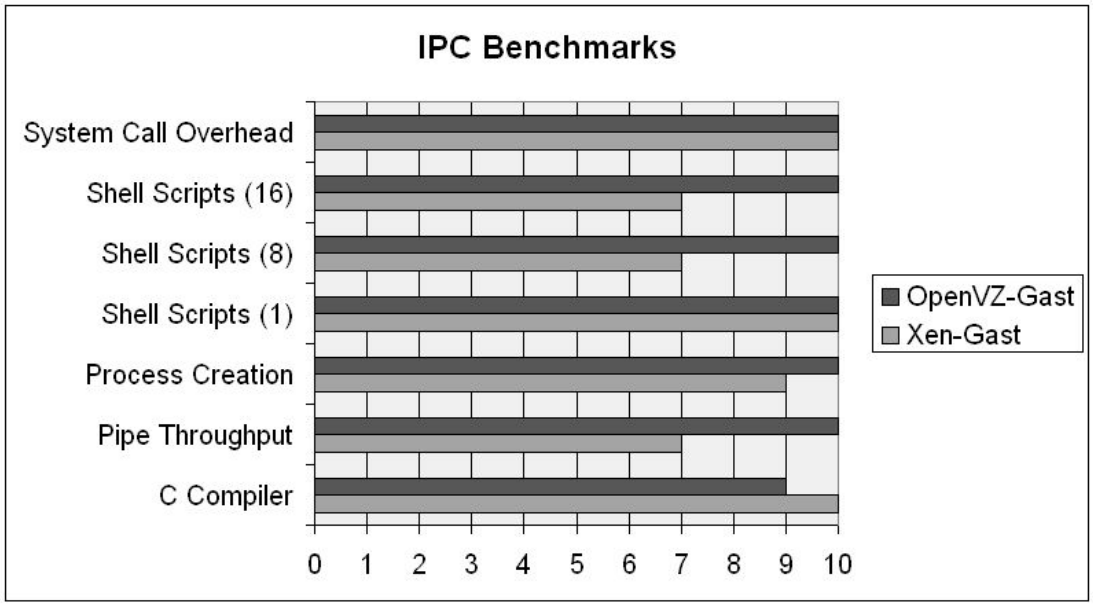

Abbildung 3.3: IPC Benchmarks

Im abschließenden Dateisystem Benchmark verdeutlichen sich diese Ergebnisse, die in Abbildung 3.4 dargestellt sind. Hier fällt zunächst auf, dass sowohl Xen als auch OpenVZ einen Verlust an Performance im Vergleich zu einem reinen Hostsystem aufweisen. Jedoch ist der Verlust bei Xen in einigen Bereichen bei weitem höher als bei OpenVZ. Während OpenVZ-Gäste direkt auf den Festplattenbereich zugreifen, der direkt vom Host-Kernel eingebunden und verwaltet wird, muss dieser Bereich unter Xen wieder über die virtualisierte Architektur über den eigenen Kernel angesprochen werden, was offensichtlich einen höheren Verlust von Performance zur Folge hat.

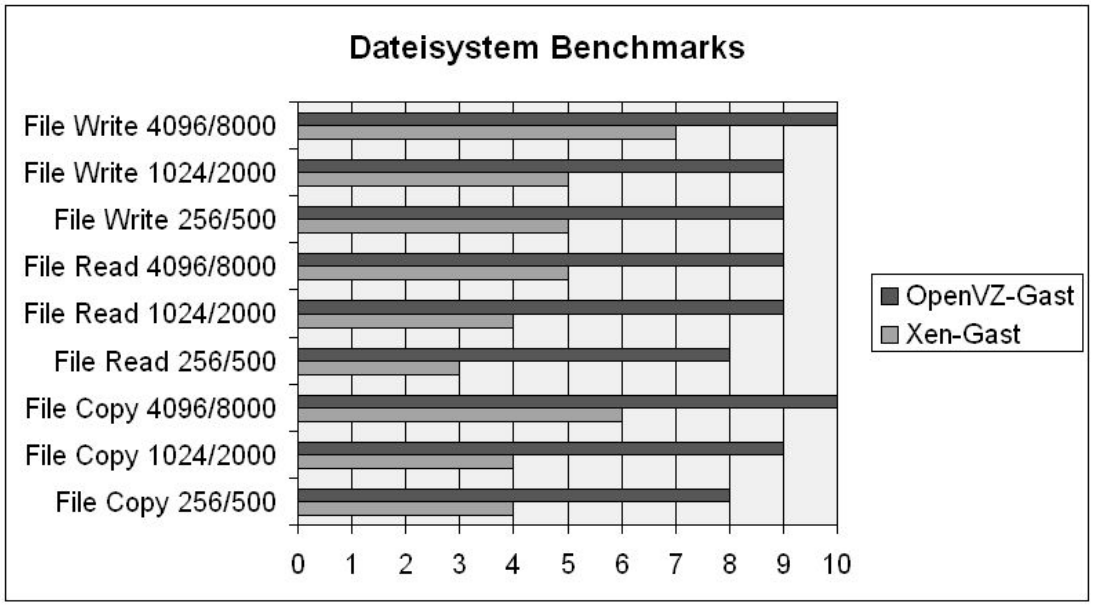

Abbildung 3.4: Dateisystem Benchmarks

3.4.2 Siege

Der *Siege*[14] Benchmark ist ein Werkzeug, das ursprünglich zur Entwicklung von Webseiten genutzt wurde, um die Performance von *Hypertext Transfer Protocol* (HTTP) Anfragen auf eine Webseite zu messen. Siege liefert eine Aussage darüber, mit welcher maximalen Geschwindigkeit Anfragen von einem Webserver beantwortet werden können und erzeugt somit sowohl eine Netzwerkauslastung als auch Arbeitsaufwand für den Webserver.

3.4.2.1 Siege Benchmarkergebnisse

Im Folgenden werden nun die Ergebnisse des Siege Benchmarkverfahrens für die Ermittlung der Netzwerkperformance von virtuellen Maschinen dargestellt. Hierzu wurde jeweils ein Lasttest über 30 Minuten durchgeführt, der jeweils

[14]http://www.joedog.org/JoeDog/Siege

gegen einen nicht virtualisierten Host und 3 parallel laufenden virtuelle Gästen, Anfragen an einen Apache-Webserver stellt.

Tabelle 3.1: Netzwerk-Benchmarks

	Host	**Xen-Gäste (3 parallel)**	**OpenVZ-Gäste (3 parallel)**
Summe Transaktionen	1426657	1592317	1614994
Summe übertragene Datenmenge (MB)	122,45	136,77	137,07
Summe durchschnittliche Antwortzeit (sec)	0,02	0,02	0,05
Summe Transaktionsrate (trans/sec)	792,67	884,68	897,05
Summe durchschnittliche Übertragungsrate (MB/sec)	0,07	0,07	0,07

Tabelle 3.1 stellt die Ergebnisse des Siege Benchmarks dar und verdeutlicht, dass die Implementierung der virtuellen Netzwerkstruktur in keiner Hinsicht der physikalischen nachsteht. Es konnte sogar gezeigt werden, dass die Gastsysteme eine höhere Transaktionsrate als das nicht virtualisierte Hostsystem liefern. Dies ist vermutlich auf die effizientere Ausnutzung von 3 parallel betriebenen Webserver-Prozessen zurückzuführen.

4 Bewertung und Fazit

4.1 Bewertung

Die aus der Analyse hervorgehende exemplarische Implementierung in einer Testumgebung ermöglichte es Benchmarkverfahren durchzuführen, die eine Bewertung der Betriebssystem- und Paravirtualisierungskonzepte erlauben. Es stellt sich heraus, dass OpenVZ und Xen jeweils enorme Vorteile in ihren Virtualisierungsverfahren für eine Linux-Serverumgebung bieten können. Im allgemeinen konnte jedoch festgestellt werden, dass der Performance eines eingesetzten Virtualisierungsverfahrens, die Komplexität der jeweiligen Implementierung und somit der Implementierungsaufwand gegenübersteht.

So bietet OpenVZ eine optimale Ausnutzung von Ressourcen mit nahezu keinem Verlust durch Prozesse, die die Umgebung verwalten müssen. Dem gegenüber steht jedoch die fehlende Möglichkeit für virtuelle Maschinen eigene Betriebssystem-Kernel zu verwenden. Dies beschränkt die Nutzung von OpenVZ ausschließlich auf eine Linux-Serverumgebung mit einem einzigen Kernel als Verwaltungsinstanz. Des weiteren ist es nicht möglich einem Gast physikalische Ressourcen, abgesehen von Netzwerkkarten, exklusiv zuzuweisen. Als enormen Vorteil ist momentan jedoch noch die Möglichkeit einer simplen, einfachen und schnellen Installation von Gästen zu erwähnen, die es erlaubt innerhalb von wenigen Minuten einen neuen Gast zu erstellen. Somit eignet sich OpenVZ

speziell für den Einsatz in Umgebungen mit ähnlicher bzw. identischer Struktur der Gäste, wie beispielsweise eine Webserver-Farm.

Dem gegenüber bietet Xen genau die Funktionalität an, unterschiedliche Betriebssysteme auf einem Hostsystem zu betreiben. Da Xen im weitesten Sinne eine vollständige virtuelle Hardwarearchitektur zur Verfügung stellt, ist es möglich physikalische Ressourcen wie eine CPU oder Speicherbereiche explizit einer virtuellen Maschine zuzuweisen. Dies ermöglicht den Aufbau von komplexen Umgebungen mit einer Vielzahl unterschiedlicher Gastsysteme. Daraus resultiert jedoch dementsprechend auch ein höherer Administrationsaufwand bei der Installation von Gästen, da es momentan nur wenige Werkzeuge gibt, die eine voll automatisierte Installation von Gastsystemen ermöglichen. Als Nachteil ist zusätzlich hervorzuheben, dass zum einen, trotz der Paravirtualisierung bei weitem mehr Ressourcen für das Management verloren gehen als bei einer OpenVZ-Implementierung. Des weiteren ist es nötig Anpassungen am Kernel der Gastsysteme unter Xen durchzuführen, wenn Xen als Paravirtualisierungssystem betrieben wird. Somit beschränkt sich der Einsatz von Xen auf Gastbetriebssysteme, die die Xen-Architektur unterstützen oder Anpassungen am Kernel zulassen. Dies wird zwar mit der Implementierung von Xen als Hypervisor einer Hardwarevirtualisierungsumgebung hinfällig. Es bleibt jedoch abzuwarten, wie weit die Performancenachteile durch diese Form der Implementierung kompensiert werden.

4.2 Fazit

Virtualisierung hat für die Informationstechnik in den letzten Jahren an immer größerer Bedeutung gewonnen, da in vielen modernen Rechnersystemen ein Ressourcenpool zur Verfügung steht, der häufig ungenutzt bleibt. Servervirtualisierung kann dazu dienen diese Ressourcen effizient auszuschöpfen. Dank einer großen Auswahl an unterschiedlichsten Virtualisierungsansätzen ist es möglich, in den verschiedensten Bereichen der IT eine geeignete Form von Virtualisierung zu nutzen.

Im Rahmen dieser Diplomarbeit wurden die unterschiedlichen Virtualisierungsverfahren zunächst ausführlich vorgestellt und eine Übersicht über vorhandene Virtualisierungslösungen erstellt. Aufbauend auf diesen Erkenntnissen wurden die beiden Open Soruce Implementierungen OpenVZ und Xen ausgewählt, um eine detaillierte Analyse über deren Funktionalität als Linux-Virtualisierungssystem darzustellen.

Die Analyse zeigte die genaue Implementierung und Funktionsweise der beiden Verfahren auf und lieferte die Basis für eine jeweilige exemplarische Implementierung. Diese konnte zunächst einen Überblick über den Implementierungsaufwand der Verfahren liefern. Mit der Durchführung von Benchmarks wurde es möglich eine Bewertung über die Performance der beiden Virtualisierungslösungen zu liefern. Aus der detaillierten Analyse, dem nötigen Implementierungsaufwand und den Benchmarkergebnissen, war es möglich eine Bewertung im Bezug auf die Eignung von OpenVZ und Xen für den Einsatz in einer Serverumgebung zu schließen.

Im Vergleich der beiden Verfahren stellte sich heraus, dass OpenVZ einen extrem effizienten Virtualisierungsansatz für den Einsatz auf einer Linux-Betriebssystemebene liefern kann. Dieser bietet eine Performance ähnlich der eines eigenständigen physikalischen Systems ohne spürbarem Verlust durch die Implementierung selbst. Dahingegen bietet Xen die Funktionalität einer sehr umfangreichen Ressourcenzuweisung, bei der ein Großteil der virtuellen Komponenten Gästen exklusiv zur Verfügung gestellt werden kann und somit eine hohe Skalierbarkeit garantiert wird.

Zusammenfassend kann also gesagt werden, dass die Analyse und Bewertung der verschiedenen Implementierungen zeigte, dass es kein Virtualisierungsverfahren gibt, das für jeden Anwendungsfall optimal ist. Es ist wichtig zunächst eine umfassende Begutachtung für jede Anwendung durchzuführen, um eine Lösung auszuwählen, die am ehesten den jeweiligen Anforderungen entspricht.

A Literaturverzeichnis

[1] *Xen Interface Manual*. University of Cambridge, 2002-2005. http://www.cl.cam.ac.uk/Research/SRG/netos/xen/readmes/interface/interface.html (19.11.2005).

[2] *Xen User Manual*. University of Cambridge, 2002-2005. http://www.cl.cam.ac.uk/Research/SRG/netos/xen/readmes/user/user.html (19.11.2005).

[3] *OpenVZ User's Guide - Version 2.7.0-8*. SWsoft, Inc., 2005. http://download.openvz.org/doc/OpenVZ-Users-Guide.pdf (26.04.2006).

[4] *Intel Virtualization Technology Specification for the IA-32 Intel Architecture*. Intel Corporation, April 2005. ftp://download.intel.com/technology/computing/vptech/C97063-002.pdf (26.03.2006).

[5] *IA-32 Intel Architecture Software Developer's Manual*, Band 1 - Basic Architecture. Intel Corporation, März 2006. http://www.intel.com/design/Pentium4/documentation.htm (05.06.2006).

[6] R. J. Creasy. *The Origin of the VM/370 Time-Sharing System*, Band 25 von *IBM Journal of Research and Development*. IBM, September 1981. http://www.research.ibm.com/journal/rd/255/ibmrd2505M.pdf (11.06.2006).

[7] Kenneth J. Duda und David R. Cheriton. *Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose schedular*. Dezember 1999.
http://www.cse.ucsc.edu/∼sbrandt/courses/Winter00/290S/bvt.pdf (17.08.2006).

[8] Dhabaleswar Panda Piyush Shivam, Pete Wyckoff. *EMP: Zerocopy OSbypass NICdriven Gigabit Ethernet Message Passing*. Association for Computing Machinery, November 2001.
http://www.sc2001.org/papers/pap.pap315.pdf (06.08.2006).

[9] Yi-Min Wang Chad Verbowski Helen J. Wang Jacob R. Lorch Samuel T. King, Peter M. Chen. *SubVirt: Implementing malware with virtual machines*. University of Michigan, Microsoft Research, Mai 2006.
http://www.eecs.umich.edu/Rio/papers/king06.pdf (17.05.2006).

[10] Ludger Schmitz. *Virtualisierung: Rechenleistung aus dem großen Topf*. computerwoche.de, Januar 2005.
http://www.computerwoche.de/produkte_technik/software/553021/ (28.11.2005).

[11] Reiner Sailer Sean Dague, Daniel Stekloff. *xm - Xen management user interface*. Linux man page, Juni 2006.

[12] Prof. Dr.-Ing. Wilhelm Meier und Dipl.-Inf. Thorsten Kockler. *Mehrarbeit für CPUs*. Linux Magazin, April 2006.

[13] Gerald J. Popek und Robert P. Goldberg. *Formal requirements for virtualizable third generation architectures*, Band 17 von *Communications of the ACM*. ACM Press - New York, NY, USA, 1974.
http://portal.acm.org/citation.cfm?doid=361011.361073 (11.06.2006).

[14] Alan Zeichick. *The Rings of Power: Intel's VT-x Technology and the Secrets of Virtualization.* Intel Software Network - DevX, Dezember 2005. http://www.devx.com/Intel/Article/30125 (07.06.2006).

B Erklärung

Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Leverkusen, den 25. August 2006

Björn Groß-Hohnacker