Virtualisierungsansätze mit Schwerpunkt Xen

Markus Gerstel

Hauptseminar: Ansätze für Betriebssysteme der Zukunft, WS 2005/06
Betreuer: Dipl.-Inf. Tobias Landes
Technische Universität München

Zusammenfassung Die Virtualisierung ist aus der Informatik nicht mehr wegzudenken. Im Folgenden werden die Anfänge der Virtualisierung von Hardware-Komponenten 1959 bis hin zur heutigen kompletten Systemvirtualisierung mit UML, VMWare und Xen betrachtet. Dabei wird auch auf die Konzepte, deren Umsetzung und Besonderheiten bei der Systemvirtualisierung geachtet. Anschließend wird ein kurzer Blick in die Zukunft von Xen und der x86-Architektur gerichtet.
Im Gegensatz zu einer früheren Arbeit [16] aus dieser Seminarreihe, die sich vor allem mit dem allgemeinen technischen Aspekt der Virtualisierung auseinandersetzt, soll hier das Augenmerk auf die tatsächlichen Implementierungen gerichtet werden.

Inhaltsverzeichnis

1 Geschichte, Definition und Ziele der Virtualisierung

Die Geschichte der Virtualisierung beginnt im Juni 1959 mit der Abhandlung „Time Sharing in Large Fast Computers“ von Christopher Strachey. Seine Idee: Ein 1-CPU-System arbeitet Programme nacheinander ab. Greift ein Programm auf ein Pheripherie-Gerät zu, so wird ein Kontextwechsel durchgeführt, und das nächste Programm wird bis zu einem Pheripherie-Zugriff abgearbeitet. Dieses Konzept - heute bekannt als „Multiprogrammierung“ - erlaubt es die CPU (und damit die damals kostbare Rechenzeit) maximal auszunutzen. Als einfaches Scheduling-Verfahren ist es auch ein Vorläufer moderner Multitasking-Konzepte.

Darüberhinaus kann es nach heutigen Maßstäben auch als erste Virtualisierung einer CPU betrachtet werden: Virtualisierung ist die Technik, Betriebsmittel eines Computers so zu repräsentieren, dass sie von Benutzern und Programmen einfach verwendet werden können, ohne dass die genaue Implementation oder physikalische Eigenschaften des Betriebsmittels bekannt sein müssen. Christopher Strachey erdachte die logische CPU, auf der Programme wie auf einer „normalen“ CPU gestartet werden können. Ein Scheduler ordnet dann diese logische CPU der physikalischen CPU zu. Für den Anwender geschieht dies transparent.

1.1 Von der Multiprogrammierung zur Virtualisierung

Ab 1962 nahm der ATLAS Computer seinen Betrieb in Manchester auf. ATLAS bot nicht mehr nur Multiprogrammierung, sondern zusätzlich den ersten Drucker-Spooler und einen einstufigen virtuellen Speicher[1] mit demand paging.

Durch die Virtualisierung des Hauptspeichers konnten beim ATLAS-Computer vor allem Kosten eingespart werden. Gleichzeitig entstand aber für den Endbenutzer kein großer Mehraufwand, da die Ein- und Auslagerung der Hauptspeicherseiten im Hintergrund ablief.

1.2 Moderne Virtualisierung von Hardware-Komponenten

Bei heutigen Betriebssystemen spielt die Virtualisierung von Hardware-Komponenten auch aus anderen Gründen eine Rolle: Durch *Klassenbildung* kann z.B. ein Anwendungsprogramm auf blockorientierte Geräte unabhängig von dem tatsächlich eingesetzten Gerät zugreifen. Wird eine neue Festplatte installiert, muss **dd** nicht auf die neue Komponente angepasst und neu kompiliert werden.

Identische Vervielfachung der CPU ermöglicht durch die Zuordnung der virtuellen CPUs auf reale CPUs Multitasking und parallele Ausführung.

Die Umlenkung der Druckausgabe auf Dateien beim Spooling wird *Substitution* genannt. So kann die Schnittstelle des realen Druckers benutzt werden, obwohl dieser gerade mit einem Job beschäftigt oder sogar aktuell nicht am Rechner angeschlossen ist. Diese Basisfunktionalitäten, die für unsere heutigen modernen Betriebssysteme selbstverständlich sind, beruhen alle auf dem Konzept der Virtualisierung und Abstraktion.

Die Virtualisierung des Betriebssystems kann sogar von Anwendungssoftware genutzt werden. So kann die Software **Daemon-Tools**[2] über eine SCSI-Schnittstelle dem Betriebssystem CD/DVD-Laufwerke zur Verfügung stellen,

[1] ATLAS hatte 4 Stacks mit je 8 Seiten je 512 Wörter je 6 Bytes Hauptspeicher. Dem standen 4 Trommeln mit je 48 Seiten Hintergrundspeicher gegenüber. Jedes Bit Hauptspeicher kostete etwa das 6-fache eines Bits Hintergrundspeicher.

[2] http://www.daemon-tools.cc

die wiederum mit Image-Dateien „befüllt“ werden können. Für den Endbenutzer bzw. andere Anwendungsprogramme ist dieses virtuelle Laufwerk nicht von einem realen Laufwerk unterscheidbar.

1.3 Von der Virtualisierung zur virtuellen Maschine

Mitte der 60er wurde das M44/44X-Projekt am IBM Watson Research Center durchgeführt. In erster Linie sollte das Verhalten von diversen Timesharing-Konzepten erforscht werden. Als Hauptrechner diente ein IBM 7044 (auch M44 genannt), auf dem mehrere virtuelle Maschinen des Typs IBM 7044 (dann jeweils 44X genannt) liefen. Im Rahmen dieses Projekts wurde unter anderem die FIFO-Anomalie entdeckt und einige Vorurteile gegen den virtuellen Speicher beseitigt.

Mit dem CP-40 und dem VM/370 entwickelte IBM die Technik der virtuellen Maschinen auf der Mainframe-Ebene weiter und ist bis heute ein geschätzter Anbieter von robusten Großrechnern mit Virtualisierungslösungen.

1.4 Von der virtuellen Kopie der realen Maschine zur virtuellen Maschine und zurück

Bis 1977 war eine virtuelle Maschine üblicherweise eine Kopie einer realen Maschine. Die von Kenneth Bowles entwickelte Pseudo-Maschine - kurz p-Maschine genannt - änderte dieses. Sie existierte jedoch nur auf dem Papier. Um Programme ausführen zu können, musste ein Emulator entwickelt werden - ein Programm, das eine p-Maschine auf einem anderen Computer in allen Funktionen nachbildet[3].

Die für die p-Maschine entwickelten Programme wurden portabel indem man solche Emulatoren für verschiedene Plattformen zur Verfügung stellte. Die p-Maschine machte so die Programmiersprache **Pascal**[4] bekannt.

Ähnlich verhält es sich mit der 1991 entwickelten, heute wesentlich verbreiteteren **Java Virtual Machine** (JVM). Die JVM arbeitet direkt mit Java-Bytecode, muss selbst aber auf einem realen Rechner emuliert werden. Mit Java konnten die Vorteile der Maschinencode-Welt mit denen der interpretierten Sprachen verbunden werden: Eine mittlerweile sehr effiziente und schnelle Emulation mit Hilfe von just-in-time-Compilern sowie eine sehr gute Portabilität.

Doch die JVM ging noch einen Schritt weiter als die p-Maschine: 1997 stellte Rockwell-Collins Inc. den JEM1-Mikroprozessor vor. Dieser konnte Java-Bytecode ohne Emulation direkt verarbeiten und stellte so die zur JVM passende „Java Real Machine“ dar.

[3] Emulation wird oft mit Simulation verwechselt. Ein Simulator versucht möglichst genau die internen Zustände des simulierten Systems nachzuahmen, z.B. taktzyklengenaues Verhalten. Ein Emulator arbeitet dagegen ergebnisorientiert. Da interne Vorgänge des emulierten Systems nur soweit wie unbedingt nötig beachtet werden, sind für einige Funktionen eventuell Abkürzungen möglich.

[4] Wobei nur das an der Universität von San Diego, Kalifornien (UCSD) verwendete UCSD-Pascal auf der p-Maschine aufsetzte. Das später bekanntere **Turbo Pascal** verwendete keine virtuelle Maschine.

1.5 „Middleware“ oder Software-Virtualisierung

1993 stellte Sun eine Software namens **Wabi** vor. Diese erlaubte es mit einem völlig neuen Virtualisierungsansatz Windows-Programme direkt unter Suns eigenem Betriebssystem Solaris 2.2 sowohl auf der x86- als auch auf der SPARC-Plattform auszuführen: Eine Zwischenschicht sorgte dafür, dass Windows-Systemaufrufe (API-Calls) in X-Windows-Systemaufrufe übersetzt wurden. Während auf der x86-Architektur der restliche Code direkt ausgeführt werden konnte, wurde auf dem SPARC-Prozessor ein x86-Emulator verwendet. Mit Wabi konnte Software erstmals unverändert auf einem anderen Betriebssystem und auf einer anderen Architektur als der ursprünglich vorgesehenen verwendet werden.

Linux war zu diesem Zeitpunkt noch unbedeutend und die Chancen, dass Sun Wabi auch für dieses Betriebssystem bereitstellen würde, entsprechend schlecht. Im Juni 1993 formierte sich das **Wine**-Projekt, um mit einem ähnlichen Ansatz Windows 3.1-Programme auch unter Linux ausführbar zu machen. Im Unterschied zu Wabi besitzt Wine jedoch keine Emulatorschicht[5] und setzt daher eine x86-Architektur voraus.

Mittlerweile ist Wine so weit entwickelt worden, dass selbst Aufrufe in spezielle Bibliotheken (z.B. DirectX) unter Linux schnell ausgeführt werden können. Über diese Schnittstellen können die Anwendungsprogramme auch auf freigegebene Verzeichnisse oder Geräte zugreifen. Eine zusätzliche Speichervirtualisierung verhindert Konflikte um gleiche Speicherräume und sichert das Wirtsbetriebssystem vor unerlaubten Zugriffen. Wine funktioniert mittlerweile auf jedem POSIX-kompatiblen Betriebssystem.

Wine übersetzt nicht 100% der von Windows zur Verfügung gestellten API-Funktionen. Die Unterstützung ist dennoch so gut, dass Firmen ihre Software zusammen mit Wine verkaufen statt sie selbst auf z.B. Linux zu portieren[6].

1.6 Das Betriebssystem im Betriebssystem – User Mode Linux

Im Jahr 1999 sorgte der Programmierer Jeff Dike mit einem Linux-Patch für großes Aufsehen: Er modifizierte den Open-Source-Kernel so, dass dieser auch als unpriviligierter Prozess unter Linux ausgeführt werden konnte. Dementsprechend nannte er seine Entwicklung **User Mode Linux**[7] (UML). Mit UML war es möglich mehrere Instanzen des Linux-Betriebssystems gleichzeitig auf demselben Rechner auszuführen. Inzwischen arbeitet Jeff Dike im Auftrag der Intel

[5] Das rekursive Akronym WINE steht für WINE Is Not an Emulator. Das Wine-Projekt ist unter http://www.winehq.com zu finden.

[6] Beispielsweise bietet die Firma StarFinanz GmbH ihre Homebanking-Software **StarMoney** 5.0 (für Windows) zusammen mit Wine auch für Linux-Systeme an: http://www.starmoney.de/index.php?linux

[7] Nach dem deutschen Wikipedia-Eintrag [23] war der Name des Projekts angeblich ursprünglich Linux-on-Linux (LoL). Dies konnte allerdings nicht durch weitere Quellen bestätigt werden.

Corporation ausschließlich an der Weiterentwicklung von UML.

Kernelprogrammierer konnten nun einen Kernel gleich auf dem Entwicklungssystem mit den aus der normalen Programmwelt bekannten Debugging- und Profilingtools testen. Somit entfiel das umständliche Überspielen auf einen dedizierten Testrechner und auch die Fehlersuche gestaltete sich wesentlich einfacher. Außerdem lässt sich UML lizenzkostenfrei verwenden und liegt im Quellcode vor.

Tatsächlich sind die neuen Möglichkeiten so nützlich, dass UML seit der Kernelversion 2.6 zur Linux-Standardausstattung gehört.

1.6.1 Die UML-Architektur

Der Linux-Kernel ist bereits auf ein breites Spektrum von Hardware-Architekturen portiert worden. Jeff Dike konnte auf der generischen *arch*-Schnittstelle des Kernels aufbauen und passte ihn an eine neue Architektur an: *um* (User Mode). In dieser werden alle hardwarespezifischen Befehle durch Software-Funktionen emuliert, die je nach Bedarf Befehle an den echten Linux-Kernel (*Host-Kernel*) weitergeben. Im UML-System gibt es keine Treiber für Pheripherie-Geräte. Stattdessen werden die Resourcen des Host-Systems verwendet.

Der Host-Kernel behandelt den UML-Kernel genauso wie jeden anderen Prozess. Der UML-Kernel benötigt damit weder Sonderrechte noch muss er als root ausgeführt werden.

Innerhalb der virtuellen UML-Maschine verhält sich der UML-Kernel fast wie ein normales Betriebssystem: Es gibt einen eigenen Scheduler, Rechteverwaltung, Logging und diverse Hintergrundprozesse. Die Prozessverwaltung ist mit dem Host-System gekoppelt: Für jeden Prozess innerhalb der virtuellen UML-Maschine wird ein entsprechender Prozess im Host-System angelegt. Teile dieser UML-Kernelfunktionen sind aktuell noch von einer x86/Linux-Plattform abhängig. Allerdings versucht man derzeit UML auch auf andere verbreitete Plattformen wie x86/Windows, x86/BSD und PPC/Linux zu portieren[8].

1.6.2 Die Virtualisierung in UML am Beispiel von Systemaufrufen

Systemaufrufe von Prozessen in UML-System dürfen natürlich nicht an den Host-Kernel geschickt werden. Im UML-System läuft ein spezieller *Tracing-Thread*, der solche Aufrufe abfangen und an den UML-Kernel weiterleiten kann, wo sie dann bearbeitet werden. Eine Besonderheit im Linux-Kernel verlangt allerdings, dass der Systemaufruf auch beim Host-Kernel ankommt. Der UML-Kernel wandelt den Originalbefehl in einen harmlosen *getpid*-Aufruf und reicht diesen an

[8] Gemeint ist hier das Host-System. Da UML keine Emulationsschicht besitzt, sind Host- und Gast-Architektur zwangsläufig gleich.

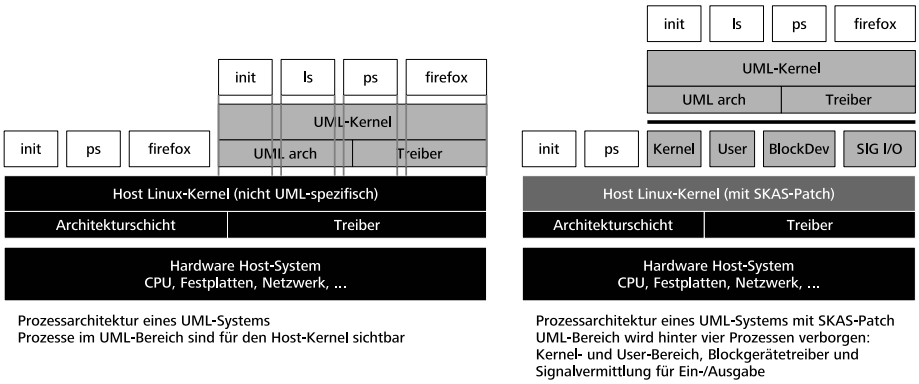


Abbildung 1: UML-Architektur
Nach Vorlage aus [8]

den Host-Kernel weiter.[9] Die Rückgabe wird anschließend verworfen, während der UML-Kernel das eigene Ergebnis an den Prozess zurückgibt, der diesen Eingriff gar nicht bemerkt.

1.6.3 Der UML-SKAS-Patch

Der originale UML-Ansatz sieht nur einen minimalen Aufwand zur Speichervirtualisierung vor. Dadurch kann böswillige Software möglicherweise aus dem UML-System ausbrechen, indem sie den UML-Kernelspeicher überschreibt. Anschließend könnte sie mit den Zugriffsrechten des UML-Kernels auf das Host-System übergreifen. Ein weiterer Nachteil ist die in Abschnitt 1.6.1 angesprochene Prozess-Kopplung. Werden mehrere UML-Systeme gleichzeitig eingesetzt, steigt die Zahl der Prozesse im Host-System mitunter schnell an, während die Performance entsprechend sinkt. Auch sind Kontextwechsel innerhalb eines UML-Systems besonders teuer, da sie vom Host-Kernel abgesegnet werden müssen. Der sogenannte „**Seperate Kernel Address Space**“-Patch (SKAS) will sich dieser Probleme annehmen.

Durch einen geschützten Kernelspeicher und eine weitere Abstraktionsschicht werden Isolation und Performance der UML-Systeme entscheidend[10] verbessert.

[9] Tatsächlich wird ein UML-Debug-Handler vom Host-Kernel vor der Ausführung des Systemaufrufs ausgeführt. Diese Debug-Schnittstelle wurde speziell für UML in den Linux-Kernel eingebaut. Es wurde aber keine Funktion vorgesehen um den Systemaufruf zu löschen oder die Bearbeitung abzubrechen. Daher muss der ursprüngliche Aufruf überschrieben werden.

[10] Eine Kernel-Kompilierung innerhalb eines UML-Systems soll beispielsweise mit SKAS in der halben Zeit möglich sein.

Allerdings muss der SKAS-Patch auf den Kernel des Host-Systems angewendet werden.

2 Systemvirtualisierung

Die heute weitverbreitete x86-Architektur galt lange Zeit als ineffizient bzw. nicht virtualisierbar. Als Hauptursache hierfür gilt, dass die x86-CPU nicht das Abfangen aller relevanten Befehle[11] und eine anschließende Sonderbehandlung in einer speziellen Routine zuläßt.

2.1 Virtualisierung eines kompletten x86-PCs – VMWare

Dementsprechend sorgte die Firma VMWare für einige Aufregung, als sie der Öffentlichkeit 1999 ihre Software **VMWare Workstation** präsentierte. Mit dieser ließ sich erstmals ein kompletter x86-Computer auf einem anderen x86-System mit ansehnlicher Geschwindigkeit virtualisieren.[12] Die virtuelle Maschine besitzt sogar ein eigenes BIOS nebst eigener virtueller Hardware, die innerhalb gewisser Grenzen ausgewählt werden kann.

2.1.1 Virtualisierung teilweise auf Instruktionsebene

Genaue Details der Implementation von VMWare Workstation werden verständlicherweise als Geschäftsgeheimnis unter Verschluss gehalten. VMWare Inc. hält allerdings ein ausführliches Patent[13] auf die Funktionsweise der Virtualisierung. Aus diesem kann man folgendes Verfahren ablesen: Ein Virtual Machine Monitor (VMM) überprüft parallel zur Laufzeit ständig den Programmcode der virtuellen Maschine bevor dieser ausgeführt wird. Dieses Verfahren wird Scan-Before-Execution (SBE) oder **PreScan** genannt: Wird im Programmcode ein abzufangender Befehl geortet, wird er kurzerhand durch einen Breakpoint ersetzt und die Suche endet dort zunächst. Der PreScan durchsucht bei konditionalen Sprüngen beide Pfade bis zu einem kritischen Befehl weiter - oder alternativ bis zu einer maximalen Suchtiefe. Sprünge, deren Ziel zur Zeit des PreScans noch nicht feststehen, müssen emuliert werden.

[11] Mit relevanten Befehlen sind solche gemeint, die entweder den System-Zustand ändern oder abhängig von dem internen CPU-Zustand andere Auswirkungen zeigen. Zum Beispiel gibt es Ring-abhängige Befehle, deren Bedeutung sich je nach Ausführungsort ändert: z.B. in einem Kernel (Ring 0 = privilegierter Modus bzw. Kernel Mode) oder in einem Prozess (Ring 3 = unprivilegierter Modus bzw. User Mode).

[12] VMWare etablierte sich binnen kürzester Zeit als Technologieführer auf dem Gebiet der x86-Virtualisierung. Gegen Ende 2003 wurde VMWare von der damals angeschlagenen EMC Corporation für 635 Mio. USD aufgekauft. Heute ist EMC Marktführer im Bereich der Enterprise Storage.

[13] Das Patent ist im Volltext auf der Seite des United States Patent and Trademark Office einsehbar. [14]

Dieses Vorgehen kann zwar die Instruktionsschwierigkeiten sehr effizient lösen, führt aber zu einem neuen Problem: Die x86-Architektur erlaubt selbstmodifizierenden Code bzw. auch das Auslesen des eigenen Programmcodes. Zu diesem Zweck wird vom VMM eine Kopie des originalen Codes vorgehalten, um bei Bedarf Zugriffe dorthin umzuleiten.

2.1.2 Die Architektur von VMWare

Die virtuelle Maschine erhält einen eigenen virtualisierten physikalischen Speicher. Deshalb überwacht der VMM sowohl alle Speicherzugriffe als natürlich auch diejenigen auf die entsprechenden Steuerregister. Ein spezielles Modul im Host-System, der **VMDriver** (oder VMX Driver), assistiert bei der Speicherverwaltung und dient als Kommunikationsschnittstelle zwischen der VMWare Applikation (**VMApp**) und dem VMM.

Die VMWare Applikation ist eine im Host-System laufende Anwendung, die den Bildschirminhalt der virtuellen Maschine anzeigt sowie I/O-Anfragen zwischen virtueller Maschine und Host-System weiterleitet[14].

Über die VMWare Applikation kann die Hardware der virtuellen Maschine näher spezifiziert werden. Dadurch können der VM eine beliebige[15] Hauptspeichermenge, bis zu drei Netzwerkkarten[16], ein SCSI-Controller und IDE-Controller zugewiesen werden. An diese wiederum lassen sich virtuelle Festplatten, die jeweils mit einer Datei im Host-System realisiert werden, anschließen. Ebenfalls können auch reale Festplatten, reale Partitionen sowie CD/DVD-Laufwerke, die entweder ein reales Laufwerk kopieren oder ihren Inhalt aus Image-Dateien beziehen, angeschlossen werden. Außerdem lassen sich noch Floppy-Laufwerke, USB-Controller, serielle und parallele Schnittstellen hinzufügen. Am Host-System angeschlossene USB- und SCSI-Geräte können auch direkt an die virtuelle Maschine „durchgereicht“ werden.

Das von VMWare eingesetzte Verfahren funktioniert im allgemeinen mit allen x86-Betriebssystemen, auch wenn diese nicht für einen Betrieb in einer virtuellen Umgebung ausgelegt sind. VMWare stellt jedoch für unterstützte Betriebssysteme ein Softwarepaket zur Verfügung, das in der virtuellen Maschine installiert werden kann. Es beeinhaltet modifizierte Treiber für die meisten Peripherie-Geräte der virtuellen Maschine (Grafikkarte, Netzwerkkarten, etc.), die die Per-

[14] Beispielsweise die Eingabe über die Tastatur oder Kommunikation einer VM über eine reale serielle Schnittstelle nach außen

[15] In 4 MB-Schritten zwischen minimal 4 MB und einem Maximum je nach Ausstattung des Host-Systems

[16] Diese können jeweils im Bridged-Mode an ein Interface des Host-Systems gebunden werden und im Host-Only-Mode ausschließlich mit dem Host-System kommunizieren. Alternativ dazu lassen sie sich in einem NAT-Modus auch transparent für das restliche Netzwerk betreiben. Die Netzwerkkarten erscheinen in der virtuellen Maschine als AMD PCNet-Karten.

formance erheblich steigern. Zusätzlich ist ein sehr nützliches Programm enthalten, das die Bedienbarkeit der virtuellen Maschine erleichtert, indem z.B. Dateien mittels Drag-and-Drop aus der virtuellen Maschine direkt in das Host-System kopiert werden können und umgekehrt.

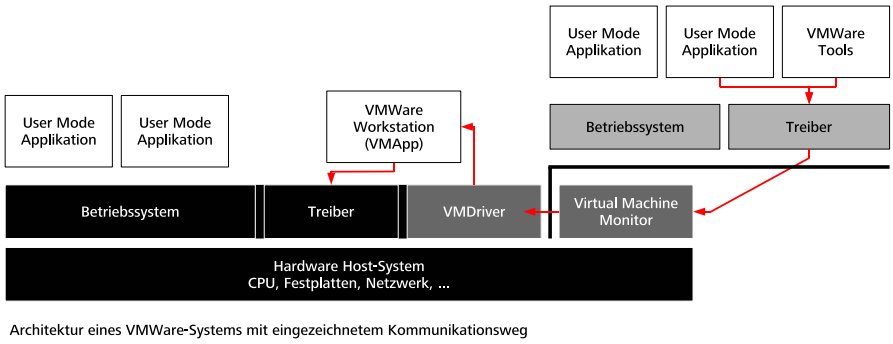


Abbildung 2: VMWare-Architektur
Nach Vorlage aus [11]

2.1.3 Marktabdeckung

VMWare wird aktuell in folgenden relevanten Versionen vertrieben:

- **VMWare Workstation** und **VMWare GSX Server**: Identische Funktionen - wobei der GSX-Server speziell auf den Serverbetrieb ausgelegt ist und mehr Hauptspeicher und Festplattenplatz unterstützt.
- **VMWare ESX Server**: Ersetzt das Host-Betriebssystem und kann so die Performance weiter steigern.
- **VMWare Virtual Center**: Das aktuelle Flaggschiff von VMWare ist für den Einsatz in Cluster-Umgebungen vorgesehen und wird u.a. von IBM zur x86-Virtualisierung in Enterprise-Umgebungen genutzt. Diese Version bietet weitere Funktionen, die im Abschnitt 2.4.1 besprochen werden.
- **VMWare Player**: Die am 20. Oktober 2005 vorgestellte Minimal-version von VMWare Workstation. Der VMWare Player kann zwar bereits existierende virtuelle Maschinen betreiben, aber die vorgegebene Hardware-Konfiguration nicht ändern. Der VMWare Player ist lizenzkostenfrei erhältlich.

2.2 Kooperative Virtualisierung mit Xen

Im Gegensatz zu VMWare, das auf eine aufwendige und zeitraubende vollständige Virtualisierung setzt, verfolgt das an der Universität von Cambridge ent-

wickelte Xen einen anderen Ansatz: Xen stellt den virtuellen Maschinen eine Architektur namens x86/Xen zur Verfügung, die der x86-Architektur sehr ähnlich[17] ist, aber auch Unterstützung für Virtualisierung bietet. Über alle virtuellen Maschinen wacht der sogenannte Xen-*Hypervisor*, ein Kontroll-Instrument das auch als Schnittstelle zwischen Host- und Gast-System dient.

Xen liegen unter anderem folgende Ideen bzw. Design-Prinzipien zugrunde:

- Die Anwendungen innerhalb der virtuellen Maschine müssen ohne Modifikationen lauffähig bleiben.[18]
- Nur Para-Virtualisierung ist fähig eine hohe Leistung mit starker Isolation auch auf unkooperativen[19] Host-Architekturen (x86) zu verbinden.
- Für eine effiziente Virtualisierung sollten die Effekte der Betriebsmittel-Virtualisierung auch auf kooperativen Host-Architekturen vor dem Betriebssystem der virtuellen Maschine nicht verborgen werden.
- Xen soll später bis zu 100 virtuelle Maschinen auf einem Server betreiben können. Diese sollen jeweils selbst mit vernünftiger Performance arbeiten können ohne sich gegenseitig zu stören.

2.2.1 Die x86/Xen-Architektur

Damit Xen die Isolation des Host-Systems (bei Xen auch *Dom0* genannt) gegenüber den virtuellen Maschinen (bei Xen *DomU* genannt) sowie zwischen den einzelnen virtuellen Maschinen garantieren kann, muss erneut eine komplexe Speichervirtualisierung erfolgen.

Bei x86 geschieht der Speicherzugriff über den Translation Lookaside Buffer[20] (TLB) auf Hardware-Ebene, der für möglichst schnelle Zugriffe unbedingt ausgenutzt werden sollte. Leider kann der TLB nicht zwischen Hypervisor- und DomU-Speicherbereich unterscheiden. Würde man für einen Systemaufruf den Codebereich zwischen Hypervisor und DomU wechseln, müsste man den TLB komplett leeren. Dies verzögert die nächsten Speicherzugriffe unnötig. Xen geht dabei folgenden Weg: Das DomU-Betriebssystem soll die Seitentabellen möglichst selbst verwalten. Lesezugriffe werden immer direkt ausgeführt, während Schreibzugriffe vom Hypervisor auf Zulässigkeit geprüft werden. Außerdem werden 64 MB am oberen Speicherende jeder DomU für den Hypervisor reserviert[21]. Damit lassen sich Systemaufrufe[22], die der Hypervisor direkt beantworten kann, ohne

[17] Das Konzept, den virtuellen Maschinen eine der Host-Architektur ähnliche, aber nicht identische Architektur anzubieten, nennt man auch Para-Virtualisierung.

[18] Dies bedeutet, dass alle Features der Architektur, die von Standard-ABIs (Application Binary Interfaces) benötigt werden, virtualisiert werden müssen.

[19] im Sinne der Virtualisierbarkeit

[20] Ein Cache für die letzten verwendeten virtuellen Speicheradressen

[21] Die oberen 64 MB können damit vom Betriebssystem der virtuellen Maschine nicht genutzt werden. Glücklicherweise wird dieser Adressbereich aber von keiner der üblichen Applikationsschnittstellen genutzt.

[22] passenderweise auch *Hypercalls* genannt

Leerung des TLB ausführen. Da nun alle Schreibzugriffe auf die Seitentabellen von Xen geprüft werden müssen, sind Systemaufrufe sehr häufig nötig. Für eine effiziente Ausführung kann das Gast-Betriebssystem viele solcher Schreibanweisungen in einen einzigem Aufruf zusammenfassen. Da der Lesezugriff sofort, der Schreibzugriff jedoch verzögert ausgeführt wird, ist es möglich, dass auf eine Seite zugegriffen werden soll, die der virtuellen Maschine noch gar nicht zugeteilt wurde. Daher muss bei einem Seitenzugriffsfehler zunächst überprüft werden, ob sich noch Schreibanweisungen in der Warteschlange befinden, diese anschließend durchgeführt und dann der Lesezugriff wiederholt werden. Segmenttabellen werden ähnlich behandelt.

Die Virtualisierung der CPU wird bei der x86-Architektur dadurch erleichtert, dass der Prozessor vier verschiedene Zugriffsebenen (*Ringe*) kennt, von denen aber üblicherweise nur zwei (Ring 0 und 3) verwendet werden[23]. Um die Sicherheit des Hypervisors zu gewährleisten, führt Xen Gast-Betriebssysteme immer auf Ring 1 aus. Dies erleichtert das Abfangen und Para-Virtualisieren von ungewünschten Befehlen ungemein, erfordert aber eine Modifikation des Gastsystems. Tatsächlich bleiben durch diese Änderung nur noch zwei zu beachtende Spezialfälle übrig: Systemaufrufe (von Anwendungsprogrammen in DomU) und Seitenfehler. Letztere können beim x86 nur mit einem Umweg über Ring 0 behandelt werden. Für erstere kann das DomU-System per Hypercall einen eigenen Exceptionhandler installieren[24].

Ein- und Ausgabe-Operationen werden über gemeinsame generische Ringpuffer realisiert. Xen unterstützt auch asynchrone Nachrichtenübermittlung zu einem Gast-System, ähnlich den Hardware-Interrupts. Nicht für jede Operation oder jeden Interrupt ist sofort ein Hypercall mit Kontextwechsel notwendig. Das Gast-Betriebssystem kann selbst entscheiden wie oft es unterbrochen werden soll und so auch dynamisch zwischen kleinen Reaktionszeiten oder großem Durchsatz wählen. Die Problematik, dass entweder besondere Geräte[25] jenen Systemen nicht zur Verfügung gestellt werden - wie bei VMWare der Fall - oder fehlerhafte Treiber von diesen Geräten die Systemstabilität oder -isolation beeinträchtigen können, wird in „Safe Hardware Access with the Xen Virtual Machine Monitor“[10] genauer betrachtet.[26]

[23] Vgl. Fußnote 11. Das letzte bekannte Betriebssystem, das Ring 1 oder 2 verwendet hat, ist nach Meinung der Xen-Entwickler OS/2 gewesen.

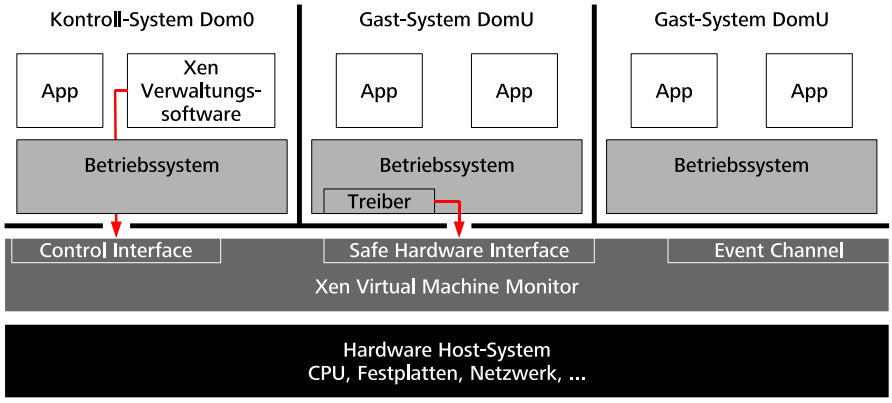
[24] Wobei der Hypervisor natürlich prüfen muss, ob ein gültiger Exceptionhandler gemeldet wird. Diese Prüfung ist sehr einfach: Gültige Exceptionhandler dürfen keinen Programmcode enthalten, der mit Ring 0-Privilegien ausgeführt wird.

[25] Zum Beispiel Fernsehkarten oder andere Geräte, die nicht in eine der Standardklassen eingeordnet werden können.

[26] In „Safe Hardware Access with the Xen Virtual Machine Monitor“[10] wird auch der von Xen verwendete Lösungsansatz beschrieben: Durch *isolated driver domains* (IDDs) werden Treiberfehler eingegrenzt und virtuellen Maschinen dennoch der direkte Zugriff auf Geräte des Host-Systems ermöglicht. Stürzt ein Treiber ab, kann

Xen stellt dem Gast-System drei verschiedene Zeiten zur Verfügung: Die *real time* (= die Zahl von Nanosekunden, seit der Computer gestartet wurde), die *virtual time* (= die nur verstreicht, wenn Xen die Kontrolle an das Gast-System übergeben hat[27]), und die *wall-clock time* (= eine Differenz zur real time, mit Hilfe derer die Uhrzeit in der virtuellen Maschine geändert werden kann, ohne die real time zu beeinflussen). Für die real und virtual time stellt Xen jeweils einen Alarm-Mechanismus zur Verfügung. Die Gast-Systeme sind für die Verwaltung eigener weiterer Alarme selbst zuständig.

Festplattenzugriffe erfolgen über virtuelle blockorientierte Geräte. Da nur Xen selbst den Überblick über die tatsächlichen Speicherorte auf dem physikalischen Datenträger hat, kann es die einzelnen Zugriffe für höhere Performance umsortieren. Falls das Gast-System die ursprüngliche Schreibreihenfolge zwingend benötigt[28], kann es sogenannte *reorder barriers* setzen.



Architektur eines Xen-Systems: Hypervisor/VMM stellt x86/Xen-Architektur zur Verfügung, auf die das Kontroll-Betriebssystem sowie die Gast-Systeme portiert wurden.

Abbildung 3: Xen-Architektur
Nach Vorlage aus [12]

er automatisiert in wenigen Millisekunden neugestartet werden, ohne das Host- oder Gast-System in Mitleidenschaft zu ziehen.

[27] Deshalb richtet sich der Scheduler des Gast-Systems nach der *virtual time*.

[28] z.B. bei write-ahead logs

2.2.2 Der Xen-Hypervisor

Der Hypervisor ist in der Xen-Architektur die oberste Kontrollinstanz. Die Entwickler von Xen entschieden sich diesen möglichst klein[29] zu halten, um Regelwerk und Mechanismen weitgehend zu trennen. So übernimmt er zwar das Scheduling zwischen den Gast-Systemen sowie eine einfache Netzwerkfilter-Funktion und auch die Prüfungen beim Speicherzugriff, kümmert sich aber nicht um die höheren Funktionen - wie die genaue Aufteilung der CPU zwischen den Gast-Systemen, oder welche Pakete im Netzwerk gefiltert werden sollen. Über feste Schnittstellen können weniger priviligierte Programme diese Entscheidungen treffen. Die Kontrollschnittstellen sind allerdings nur vom Dom0-System aus zugänglich. Ein Hintergedanke dazu ist auch, dass außerhalb des Xen-Hauptprojektes hochwertige Kontrollsoftware entwickelt werden kann[30].

Aktuell wird in Xen die Rechenzeit für die Gast-Systeme mit einem *Borrowed Virtual Time* (BVT) Scheduler zugeteilt. BVT wurde 1999 von Kenneth J. Duda and David R. Cheriton entwickelt. Mit diesem Algorithmus kann einerseits ein hoher Arbeitsdurchsatz erreicht werden, andererseits können Gast-Systeme z.B. zur Unterbrechungsbehandlung kurzfristig aufgeweckt werden. Dies ist unter anderem notwendig, um TCP-Timingvorschriften einzuhalten. BVT weicht für die Interruptbehandlung von einem strikt fairen Schedulingverfahren ab, gleicht dies aber über einen größeren Zeitraum wieder aus. Ein Beispielszenario wird in Abbildung 4 dargestellt:

Drei Prozesse werden ausgeführt. P1 möchte 5 Zeiteinheiten lang arbeiten, P2 und P3 jeweils 10. Als Zeitscheibe werden 5 Zeiteinheiten angenommen. Zum Zeitpunkt $t_R = 6$ tritt eine Unterbrechungsanforderung für P1 auf, deren Abarbeitung 1 Zeiteinheit benötigt. Nach einem fairen Zeitscheibenverfahren ist P1 erst zum Zeitpunkt $t_R = 15$ an der Reihe. Im BVT-Verfahren kann P1 für Unterbrechungsanforderungen auf einen „Bonus“ von hier 7 Zeiteinheiten zurückgreifen, und kann daher bereits zum Zeitpunkt $t_R = 10$ die Unterbrechung behandeln.[31]

Da das Scheduling außerhalb des Hypervisors gesteuert wird, ist es mit relativ geringem Aufwand möglich auch mit anderen Scheduling-Algorithmen zu experimentieren.

Der Hypervisor, beziehungsweise die an den Hypervisor angeschlossenen Steuerprogramme, sind auch für eine sinnvolle Performance-Isolation verantwortlich. Tatsächlich kann ein Xen-DomU-System, in dem „asoziale“ Prozesse ablaufen,

[29] Dies verbessert natürlich auch die Skalierbarkeit. Aktuell benötigt Xen pro Gast-System fest 20kB Speicherplatz.

[30] Die Schnittstellen bieten Funktionen, um Gast-Systeme zu erstellen, starten, halten, beenden, löschen, etc. Desweiteren können blockorientierte Geräte verwaltet oder Netzwerkfilter und Bandbreitenbegrenzungen eingerichtet werden. Auch Profiling-Daten stehen zur Verfügung.

[31] Das genaue BVT-Verfahren wird in „Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler“ [6] beschrieben.

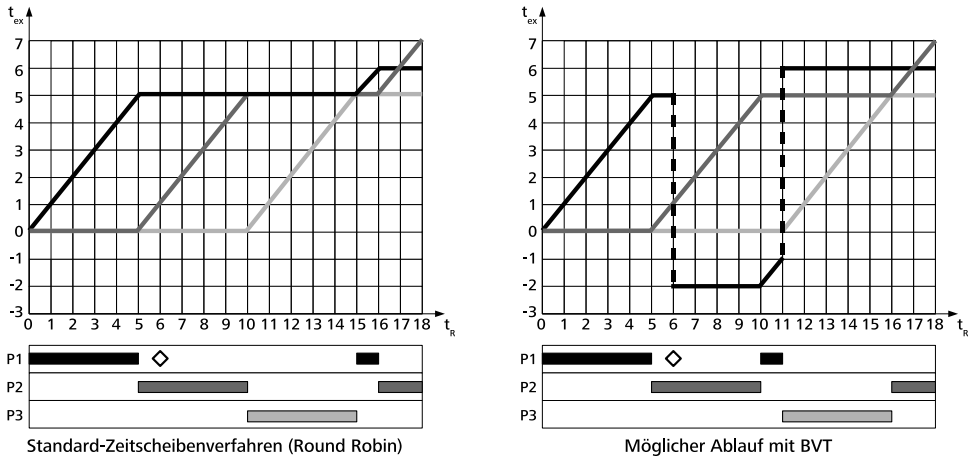


Abbildung 4: Schedulingverfahren im Vergleich

andere Gast-Systeme nur minimal beeinflussen. Die Entwickler von Xen führten einen Test mit vier Gast-Systemen durch: Zwei Benchmark-Umgebungen sowie ein System, in dem eine möglichst hohe Festplattenauslastung generiert wird. Dazu kommt noch ein System in dem einerseits ständig neue Prozesse gestartet werden, andererseits ein zweiter Prozess ständig versucht eine maximale Menge an Speicherplatz zu allokieren und anschließend alles wieder freigibt. Die beiden Benchmark-Umgebungen wurden durch das Verhalten der anderen beiden Gast-Systeme nur um 2% bis 4% gegenüber einem Vergleichstest abgebremst. Dieser Unterschied kann nach Ansicht der Xen-Entwickler auf Cache-Effekte sowie die zusätzlich nötigen Kontextwechsel zurückgeführt werden. Bei einem Kontrollversuch mit allen Prozessen direkt unter einem Linux-System konnten die Stör-Prozesse schnell die Oberhand gewinnen und bremsten die Benchmark-Prozesse vollständig aus.

2.3 Performancevergleich Linux - Xen - VMWare - UML

In Abbildung 5 sind die Resultate mehrerer Benchmarks zu finden. Jeder davon wurde auf einem normalen Linux-Rechner unter Xen, VMWare und UML jeweils siebenmal durchgeführt. Der angegebene Wert ist der Median aller Versuche, relativ skaliert zu dem Ergebnis der direkten Ausführung auf dem Linux-Rechner.

Der erste Test (SPEC INT2000) besteht aus reinen Rechenaufgaben, die wenig I/O oder Interaktion mit dem Betriebssystem verlangen. Der Overhead bewegt sich daher auch bei allen Virtualisierungslösungen in einem angemessenen Rahmen.

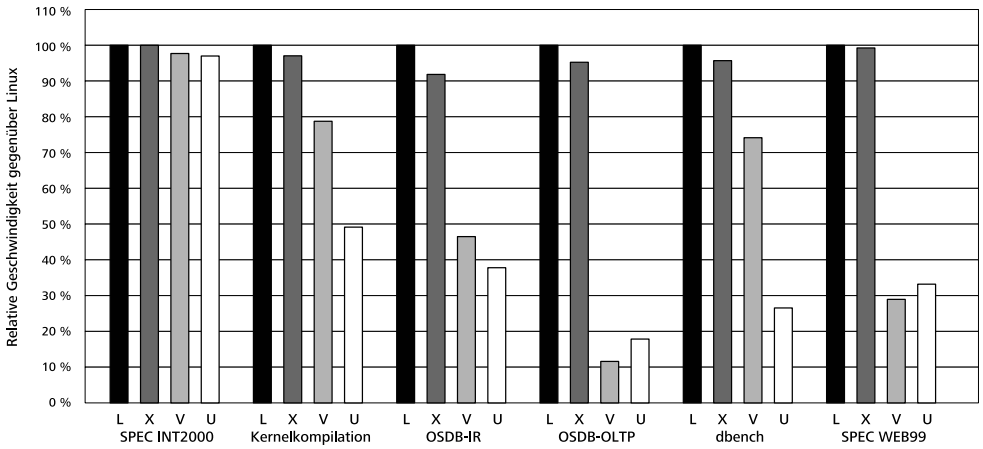


Abbildung 5: Relative Geschwindigkeiten von Linux (L), XenoLinux (X), VMWare Workstation 3.2 (V) und User Mode Linux (U).
Daten entnommen aus [12]

Der zweite Test ist die Kompilation eines Linux 2.4.21-Kernels auf einem lokalen ext3-Dateisystem mit gcc 2.96. Dieser Test gestaltet sich bereits aufwendiger, da wesentlich mehr CPU-Zeit für I/O, Scheduling und Speicherverwaltung benötigt wird. Da bei UML Systemaufrufe sehr teuer sind fällt hier die Performance entsprechend.

Die nächsten beiden Tests wurden mit einer PostgreSQL 7.1.3-Datenbank sowie der Open Source Database Benchmark suite (OSDB) durchgeführt. Seperat wurden parallele Abfragen (IR) und transaktionsbasierte Updates (OLTP) geprüft. Der OLTP-Test verlangt grundsätzlich viele synchrone Festplattenoperationen, was zu vielen Kontextwechseln führt. Sowohl VMWare als auch UML bleiben hier weit hinter dem Linuxsystem zurück. Offensichtlich stellen diese synchronen Operationen für VMWare ein besonderes Hindernis dar, da hier die Leistung von VMWare im Vergleich zu den anderen Benchmarks deutlich einbricht.

Der vorletzte Test (dbench) simuliert einen Fileserver, auf den viele Windows 95 Computer gleichzeitig zugreifen. Es wurden etwa 90.000 Dateisystemoperationen durchgeführt. Wieder stellen sich die Systemaufrufe als Flaschenhals der UML-Architektur heraus.

Bei SPEC WEB99 handelt es sich um einen komplexeren Benchmark, der das ganze System auf Tauglichkeit als Webserver prüft. Es wurden sowohl statische als auch dynamische Seitenaufrufe getestet, gleichzeitig wurden mittels HTTP POST Uploads durchgeführt, und der Server musste entsprechende Logs führen. Daher flossen in diesen Test gleichermaßen die generelle Systemperformance, I/O und Netzwerkgeschwindigkeit ein. Zusätzlich verlangt der SPEC WEB99,

dass der Server auf alle Anfragen in einer gewissen Maximalzeit reagiert und eine Mindestbandbreite garantiert. VMWare und UML sind auch hier wieder die Schlusslichter, nicht zuletzt weil sie bei der Netzwerkgeschwindigkeit weit hinter einem echten Linuxsystem zurückbleiben[32].

In diesen Benchmarks über das gesamte System schneidet die getestete Xen-Version überraschend gut ab. Es handelte sich um eine Vorversion von Xen 1.0. Interessant wäre ein aktueller Test mit Xen 2.0.7 oder der aktellen 3-beta, und den jeweils neuen VMWare- und UML-Versionen. Leider verbieten die neueren VMWare-Lizenzbedingungen das Veröffentlichen von Benchmarkergebnissen, daher bleibt die Durchführung zunächst dem interessierten Leser selbst überlassen.

Auch die in „Xen and the Art of Virtualization“[12] durchgeführten Micro-Benchmarks, die jeweils nur einen einzelnen Systemaufruf oder ein einzelnes I/O-Gerät testen, zeigen ein beachtliches Potential für Xen auf. Interessanterweise sind die Ausführungsgeschwindigkeit und Resourcenverteilung von Xen so gut, dass sogar Anomalien auftreten: Vier PostgreSQL-Instanzen auf einem Linux-SMP-System gleichzeitig sind etwa 25-30% langsamer als vier PostgreSQL-Instanzen in vier Gast-Systemen unter Xen. Hier kommt vermutlich ein SMP-Skalierungsproblem bei PostgreSQL zum Tragen, welches mit Hilfe von Xen umgangen werden kann.

2.4 Zusammenfassung

Der von Xen gewählte Ansatz der Para-Virtualisierung ist offensichtlich sehr effizient. Leider setzt er gewisse Anpassungen im Gast-System voraus. Bei freier Software, wie Linux oder den verschiedenen BSD-Varianten, ist dies kein Problem. Allerdings ist es sehr unwahrscheinlich, dass z.B. Microsoft für das auf Desktop-Computern immer noch mit Abstand wichtigste Betriebssystem Windows die notwendigen Modifikationen durchführt. Die Entwickler der ersten Version von Xen konnten zwar in Kooperation mit Microsoft Research Cambridge eine Version von Windows XP auf x86/Xen portieren, aber ob Microsoft diesen Weg weiterverfolgt ist fraglich - nicht zuletzt auch weil Microsoft mit MS Virtual PC eine eigene Virtualisierungslösung anbietet.

2.4.1 Einsatzgebiete von Xen

In den Bereichen, in denen Linux- oder BSD-Systeme als Gast-Systeme interessant sind, kann Xen jedoch punkten. Mittlerweile hat vor allem Linux in den Bereichen File- und Webserver einen großen Marktanteil. So ist Xen zum Beispiel

[32] Bei einer Gigabitanbindung kann Xen mit einer MTU von 1500 den gleichen Datendurchsatz wie ein Linux-System erreichen, VMWare kann aber nur etwa 50% und UML sogar nur etwa 20% des Datendurchsatzes erreichen. Der Grund hierfür liegt in einer speziellen Technik von Xen („page-flipping“), die unnötiges Kopieren der I/O-Daten vermeidet. [12]

für den wachsenden Markt der Virtual-Rootserver interessant: Viele Kunden, die einander nicht unbedingt immer freundlich gesonnen sind, können somit individuell ihr eigenes System vollständig administrieren, bleiben aber völlig voneinander isoliert.

Aber sogar für einen Einzelserver kann eine Dienstetrennung auf verschiedenen virtuellen Servern Sinn machen. Wird zum Beispiel ein Dienst wie SMTP-Mail durch eine Sicherheitslücke kompromittiert oder durch einen Denial-of-Service-Angriff auf Dienstebene lahmgelegt, ist nur eine einzelne Maschine betroffen und kann für sich deaktiviert, neu aufgesetzt oder in einen alten Stand zurückgesetzt werden. Die restlichen Dienste auf der Maschine (z.B. der interne IMAP-Server) bleiben während dieser Zeit erreichbar. Der geringe Overhead durch Xen und die Lizenzkostenfreiheit machen dies eventuell in Zukunft generell praktikabel.

In einem anderen Marktsegment mit Clustering und nichtlokaler Datenspeicherung kann Xen noch ein anderes wohlerprobtes Feature ausspielen: Xen erlaubt es virtuelle Maschinen während der Ausführung nahtlos von einem physikalischen Rechner auf einen anderen zu transportieren. Dazu wird in mehreren Phasen im Hintergrund der Speicher von der ursprünglichen Maschine auf die Zielmaschine kopiert und anschließend die Kontrolle an den anderen Rechner übergeben. Tests mit dieser Technik erlaubten einen Quake 3-Server für den Spieler unbemerkt im Hintergrund zu migrieren - mit einer totalen Ausfallzeit von 60ms. Das VMWare Virtual Center bietet eine ähnliche Funktionalität, in der eine Menge an virtuellen Maschinen auf physikalischen Rechnern nach Belieben hin- und herverschoben werden kann. Wartungsarbeiten können so unkompliziert vorgenommen werden. Auch ein Loadbalancing auf der Ebene von virtuellen Maschinen wird so möglich. In jedem Fall wird wohl auch dieser Bereich in der Zukunft eine sehr große Rolle spielen.

2.4.2 Xen und LVM - Hochverfügbarkeit für den kleinen Geldbeutel

Aber auch für den normalen Endanwender oder Server-Betreiber kann Xen interessant werden: Im Linux-Kernel ist aktuell das Logical Volume Management (LVM) enthalten. LVM ermöglicht es einen Bereich auf der Festplatte dynamisch zu verwalten, ähnlich wie die dynamischen Datenträger unter Windows. So können zur Laufzeit Partitionen angelegt, gelöscht, verschoben, vergrößert und verkleinert werden. Aber es ist auch möglich Schnappschüsse von Partitionen anzulegen. Dies wird über eine Copy-on-Write (CoW) Strategie ermöglicht, in der LVM nur neu geschriebene Blocks in einen eigenen Speicherbereich schreibt. Kombiniert man nun Xen mit LVM, lässt sich eine Live-Backup-Strategie nutzen:

1. Man hält die virtuelle Maschine mit Xen an
2. und lässt Xen offene Dateien auf die Festplatte schreiben.
3. Anschließend nimmt man einen Schnappschuss des Gast-Dateisystems - dies geschieht in wenigen Sekunden.
4. Nun speichert man mit Xen den gesamten Maschinenstatus in eine Datei

5. und lässt anschließend die virtuelle Maschine weiterlaufen.
6. Jetzt kann man im Hintergrund den Schnappschuss des Dateisystems zusammen mit dem Xen-Status über einen beliebigen Weg in beliebiger Zeit auf einen anderen Rechner sichern.
7. Zuletzt entfernt man beide Schnappschüsse wieder; LVM stellt die ursprüngliche Festplattenordnung wieder her und kann von vorne beginnen.

Mit solch einer Strategie kann ein laufender Server mit wenigen Sekunden bis Minuten Ausfallzeit gesichert werden. Dies ist im Prinzip für jede Art von Server interessant, denn es werden alle Dateien gesichert - auch die, die zum Backupzeitpunkt geöffnet und in Benutzung waren. Und der Server muss selbst für eine Wiederherstellung nicht neugestartet werden - er läuft einfach ab dem Backup-Zeitpunkt weiter.

VMWare Workstation besitzt bereits eine Snapshot-Funktion, die auch virtuelle Laufwerke beeinhaltet. Allerdings muss der Snapshot manuell ausgelöst werden, und kann nicht isoliert auf eine eigene Maschine transferiert werden. In Xen kann die ganze Arbeit dagegen per Cron-Script automatisiert werden.

3 Ausblick

Aktuell bietet keine Virtualisierungslösung eine komplette Unterstützung für SMP und Hyperthreading in Gast-Systemen. Xen 3 soll Unterstützung für SMP und die x86_64-Architektur bieten. Außerdem planen die Entwickler ebenfalls die Unterstützung des Intel Itanium IA64 bzw. in weiterer Zukunft auch von ARM und PPC-Host-Systemen. Ferner soll Xen 3 besseren Support für Clustering sowie bessere Managementsoftware bieten. Als neues Feature werden CoW-Dateisysteme anvisiert, welche es erlauben würden mit einem Standardimage zu starten und die einzelnen Gast-Dateisysteme nur noch als Differenz-Dateien zu verwalten. Damit kann bei einer großen Zahl von nur geringfügig unterschiedlichen Gast-Systemen viel Festplattenplatz gespart werden. Langfristig sollen diese CoW-Dateisysteme auch die Live-Migration von VMs mit lokalem Datenspeicher ermöglichen. Xen 3 wird auch ein verbessertes Quote of Service-System bieten, mit dem für Gast-Systeme Limits und Garantien für z.B. CPU-Auslastung und Netzwerkbandbreite eingerichtet werden können. Nach den Angaben der Entwickler erscheint Xen 3 etwa im Dezember 2005.

3.1 x86 im Wandel – Pacifica und VT/Vanderpool

Eine Entwicklung, die garantiert nicht nur Xen, sondern auch die anderen vorhandenen Virtualisierungslösungen beeinflussen wird, ist AMD **Pacifica**[33] und Intel **VT/Vanderpool**[34]. Diese Erweiterung des x86-Prozessors wird vor allem

[33] Mittlerweile von AMD auch „Secure Virtual Machine Architecture“ (SVM Architecture) genannt; ab 2006 vorraussichtlich in allen neuen AMD-Prozessoren enthalten.
[34] Mit breiter Verfügbarkeit ist ebenfalls 2006 zu rechnen.

die Rechtestruktur ändern: Zusätzlich zum Ring 0-3 wird es jeweils einen „root mode“ und einen „non-root mode“ geben. Der Xen-Hypervisor kann dann im Ring 0, root mode ablaufen, und das Gast-System in Ring 0-3 non-root mode. Dem Gast-System wird dann kein Zugriff auf die Bereiche des Hypervisors gewährt, und der Prozessor selbst wird umfangreiche Möglichkeiten bieten kritische Befehle abzufangen. Damit wird die Notwendigkeit entfallen, Gast-Systeme für Xen zu modifizieren. Xen 3 wird die Möglichkeiten der nächsten Prozessorgeneration bereits weitgehend unterstützen.

4 Anhang - Virtualisierungslösungen im Überblick

	Hersteller	Technik	Architektur	Lizenz
Wine	WineHQ	API-Virtualisierung	x86	GPL
UML	UML Core Team	Virtualisierung	x86	GPL
VMWare	VMWare Inc.	Virtualisierung mit PreScan	x86, x86_64	Proprietär & Freeware
Xen 1/2	Xen Team	Paravirtualisierung mit Portierung	x86 Gast: x86/Xen	GPL
Xen 3+	Xen Team	Paravirtualisierung mit Portierung	x86, x86_64 Gast: x86/Xen	GPL

	Host-BS	Gast-BS	Erweiterbar mit Hardware
Wine	Linux	Linux mit WinAPI	-
UML	Linux	Linux um-arch	Unterstützung des Host-Systems vorausgesetzt
VMWare	Windows, Linux	Windows, Linux, μnos, BSD, nahezu beliebig	VM erhält eigenes HWSet, sonst nur USB und SCSI
Xen 1/2	Linux, NetBSD	Linux, BSD, Plan 9	Nahezu beliebig
Xen 3+	Linux, NetBSD	Linux, BSD, Plan 9, Windows XP	Nahezu beliebig

	Anpassungen notwendig	Geschwindigkeit
Wine	Keine	schnell
UML	Keine	ohne SKAS: langsam mit SKAS: mittel
VMWare	Keine	ohne VMTools: mittel mit VMTools: schnell
Xen 1/2	Portierung des Gast-Systems auf x86/Xen	sehr schnell
Xen 3+	Portierung auf x86/Xen oder neuere CPU	sehr schnell

Tabellen 1-3: Virtualisierungslösungen im Überblick

Literatur

1. Prof. David Aspinall. *The Birth of Computing in Manchester - Part Two.* http://www.ukuug.org/events/linux2001/papers/html/DAspinall.html, 2001. Talk given at the UKUUG, Linux Developers' Conference. Stand vom 15.Okt.2005.
2. axv. *Kostenloser VMware-Player für vorgefertigte virtuelle Rechner.* http://www.heise.de/newsticker/meldung/65156. Meldung vom 20.Okt.2005 13:29.
3. S. Hand J. G. Hansen E. Jul C. Limpach I. Pratt C. Clark, K. Fraser and A. Warfield. *Live Migration of Virtual Machines.* http://www.cl.cam.ac.uk/netos/papers/2005-migration-nsdi-pre.pdf. Pre-print of paper to be published at NSDI 2005.
4. Peter J. Denning. *Performance Modeling: Experimental Computer Science at its Best.* http://cne.gmu.edu/pjd/PUBS/ecs.pdf. ACM President's Letter for the Communiations of ACM, November 1981.
5. Jeff Dike. *A user-mode port of the Linux kernel.* http://user-mode-linux.sourceforge.net/als2000/, 2000.
6. Kenneth J. Duda and David R. Cheriton. Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. In *Symposium on Operating Systems Principles*, pages 261–276, 1999.
7. David Greve and Matthew Wilding. *The World's First Java Processor.* http://hokiepokie.org/docs/EETimes.ps. als „Stack-based Java a back-to-future step“ in der Electronic Engineering Times vom 12. Januar 1998 erschienen.
8. Roland Gruber. *User-mode Linux.* http://www13.informatik.tu-muenchen.de/lehre/seminare/WS0405/hauptsem/Ausarbeitung05.pdf, 2005. Hauptseminar Ansätze für Betriebssysteme der Zukunft WS2004/05.
9. Wine HQ. *Wine History.* http://www.winehq.com/site/history. Stand vom 17.Okt.2005 23:04.
10. R. Neugebauer I. Pratt A. Warfield K. Fraser, S. Hand and M. Williamson. *Safe Hardware Access with the Xen Virtual Machine Monitor.* http://www.cl.cam.ac.uk/netos/papers/2004-oasis-ngio.pdf. Published at the OASIS ASPLOS 2004 workshop.
11. Bruno Ohl. *Der VMware-Ansatz: Virtualisierung von Rechnern.* http://wwwspies.in.tum.de/lehre/seminare/WS0203/hauptsem/Vortrag9_VM_Ausarbeitung_Verbessert.pdf, 2002. Hauptseminar Ansätze für Betriebssysteme der Zukunft WS2002/03.
12. K. Fraser S. Hand T. Harris A. Ho R. Neugebauer I. Pratt P. Barham, B. Dragovic and A. Warfield. *Xen and the Art of Virtualization.* http://www.cl.cam.ac.uk/netos/papers/2003-xensosp.pdf. Proceedings of the nineteenth ACM symposium on Operating systems principles, pp 164-177, Bolton Landing, NY, USA, 2003.
13. Ian Pratt. *Xen Virtualization.* http://www.cl.cam.ac.uk/netos/papers/xen-lwe2005-short.ppt. Präsentation LinuxWorld 2005, Stand vom 23.Okt.2005.
14. M. Rosenblum Scott W. Devine, E. Bugnion. *Virtualization system including a virtual machine monitor for a computer with a segmented architecture.* http://patft.uspto.gov/netahtml/srchnum.htm. United States Patent and Trademark Office, United States Patent: 6,397,242.
15. searchVB.com. *Multiprogramming.* http://searchvb.techtarget.com/sDefinition/0,,sid8_gci212615,00.html. Stand vom 10.Aug.2005.

16. Christian Surauer. *Ansätze zur Virtualisierung von Rechnern*. http://wwwspies.in.tum.de/lehre/seminare/SS02/hauptsem/Vortrag1_VirtualisierungFinal.pdf, 2002. Hauptseminar Ansätze für Betriebssysteme der Zukunft SS2002.
17. The Xen Team. *Xen 3.0 Roadmap*. http://www.cl.cam.ac.uk/Research/SRG/netos/xen/roadmap.html. Stand vom 08.09.2005.
18. User Mode Linux Core Team. *User Mode Linux HOWTO*. http://user-mode-linux.sourceforge.net/UserModeLinux-HOWTO.html. Stand vom 13.Okt.2005 18:30.
19. Wikipedia. *Java Programming Language*. http://en.wikipedia.org/wiki/Java_programming_language. Stand vom 14.Okt.2005 06:49.
20. Wikipedia. *Java Virtual Machine*. http://en.wikipedia.org/wiki/Java_virtual_machine. Stand vom 17.Okt.2005 09:56.
21. Wikipedia. *Multiprogramming*. http://en.wikipedia.org/wiki/Multiprogramming#Multiprogramming. Stand vom 15.Okt.2005 11:47.
22. Wikipedia. *UCSD p-System*. http://en.wikipedia.org/wiki/UCSD_p-System. Stand vom 2.Okt.2005 05:54.
23. Wikipedia. *User Mode Linux*. http://de.wikipedia.org/wiki/User_Mode_Linux. Stand vom 10.Okt.2005 09:47.
24. Wikipedia. *Virtualization*. http://en.wikipedia.org/wiki/Virtualization. Stand vom 22.Okt.2005 17:41.
25. Christof Windeck. *IBM steckt virtuelle Arbeitsplatz-Rechner in Blade-Server*. http://www.heise.de/newsticker/meldung/65137. Meldung vom 20.Okt.2005 11:05.