Christoph Arnold · Michel Rode
Jan Sperling · Andreas Steil

KVM Best Practices

Virtualisierungslösungen für den Enterprise-Bereich

B1 SYSTEMS

dpunkt.verlag

Christoph Arnold konzipiert und realisiert Hochverfügbarkeitslösungen, zu seinen Kernthemen gehören Linux Cluster und Cluster-Dateisysteme. Darüber hinaus beschäftigt er sich seit Jahren mit dem Aufbau von virtualisierten Umgebungen mit Open-Source-Tools. Nach seiner Ausbildung zum Fachinformatiker Systemintegration ist er seit 2009 für die B1 Systems GmbH als Linux-Consultant tätig.

Michel Rode arbeitet seit 3 Jahren bei der B1 Systems GmbH als Linux Consultant und Trainer. Er ist spezialisiert auf Hochverfügbarkeit, Virtualisierung und Mailserver.

Jan Sperling ist auf die Umsetzung von Clustern mit heartbeat und pacemaker spezialisiert. Zu seinen weiteren Schwerpunktthemen gehören Virtualisierung mit Xen und natürlich KVM. Nach seiner Ausbildung zum Fachinformatiker Systemintegration ist er seit 2006 für die B1 Systems GmbH als Linux Consultant tätig.

Andreas Steil befasst sich neben Virtualisierung schwerpunktmäßig mit Netzwerktechnik. Er ist seit 2010 bei der B1 Systems GmbH als Linux Consultant und Trainer beschäftigt. Zuvor arbeitete er 16 Jahre freiberuflich als Netzwerkadministrator und Trainer.

Christoph Arnold · Michel Rode · Jan Sperling · Andreas Steil

KVM Best Practices

Virtualisierungslösungen für den Enterprise-Bereich

dpunkt.verlag

Christoph Arnold · Michel Rode · Jan Sperling · Andreas Steil
info@b1-systems.de

Lektorat: Dr. Michael Barabas
Copy-Editing: Ursula Zimpfer, Herrenberg
Satz: Da-TeX, Leipzig
Herstellung: Nadine Thiele
Umschlaggestaltung: Helmut Kraus, www.exclam.de
Druck und Bindung: M.P. Media-Print Informationstechnologie GmbH, 33100 Paderborn

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

ISBN:
Buch 978-3-89864-737-3
PDF 978-3-86491-116-3
ePub 978-3-86491-117-0

1. Auflage 2012

Ringstraße 19 B
69115 Heidelberg

5 4 3 2 1 0

Vorwort

Dieses Buch befasst sich mit dem Thema Virtualisierung im Allgemeinen und ihrer praktischen Umsetzung mit der Kernel-based Virtual Machine (KVM) im Speziellen. Die Virtualisierung von Rechnersystemen hat in den letzten Jahren einen zentralen Platz in der Informationstechnologie eingenommen. Zwischenzeitlich ist der große Hype zwar zum Thema »Cloud Computing« übergegangen, doch so wie eine Wolke aus kleinsten Wassertröpfchen besteht, basiert praktisch jede Cloud auf virtualisierten Rechnerinstanzen. So bleibt das Thema Virtualisierung hochaktuell, wenn auch mehr im Hintergrund.

KVM benötigt im Gegensatz zu Xen keinen eigenen Kernel, sondern ist seit Kernel-Version 2.6.20 in den Linux-Kernel integriert. KVM kann somit immer in Verbindung mit dem aktuellsten Linux-Kernel genutzt und der Hypervisor mit deutlich geringerem Aufwand weiterentwickelt werden. War lange Zeit ein entscheidendes Argument für Xen und die proprietären Konkurrenten (allen voran VMware) deren erwiesene Praxistauglichkeit, so hat KVM gegenüber den etablierten Lösungen in letzter Zeit enorm aufgeholt und gilt – spätestens seit der Integration in den Vanilla-Kernel von Linux – als stabil und zukunftssicher.

Die Vorteile der Hardwareunterstützung aktueller x86-Prozessoren von Anfang an konsequent nutzend, bietet KVM eine schlanke, performante und – im Zusammenspiel mit dem Hardwareemulator QEMU und der Hypervisor-Abstraktionsschicht `libvirt` – auch eine flexible und weiträumig unterstützte Lösung für die Virtualisierung sowohl im Desktop- als auch im Serverbereich. In diesem Buch wird es vorwiegend um die Servervirtualisierung im Enterprise-Bereich gehen.

Zielgruppe

Dieses Buch richtet sich in erster Linie an versierte Anwender im Enterprise-Bereich, die KVM in ihren Arbeitsalltag integrieren möchten. Es soll ihnen die Möglichkeiten, aber auch die Grenzen von KVM

aufzeigen. Aber auch dem ambitionierten Neueinsteiger kann dieses Buch den Weg in die Welt der Virtualisierung ebnen.

Voraussetzung für das Verständnis dieses Buches sind grundlegende Kenntnisse der Funktionsweise von Rechnerhardware, TCP/IP-Netzwerken und Grundkenntnisse in der Administration Unix-artiger Systeme und der Linux-Konsole.

Entstehungsgeschichte

Bei ihrer täglichen Arbeit für die B1 Systems GmbH mit dem Schwerpunkt auf Virtualisierung und Hochverfügbarkeit in großen IT-Umgebungen sind die Autoren unter anderem mit der Realisierung KVM-basierter Lösungen beschäftigt. Oft stoßen sie dabei auf Fragen und Probleme, bei deren Lösung ein gutes Buch zum Thema KVM hilfreich wäre. Mangels Literatur suchten sie sich ihr Wissen mühsam aus allen möglichen Quellen zusammen. Dieses Buch soll seinen Lesern diese mühevolle Suche abnehmen oder zumindest erleichtern.

Aufbau

Das Buch führt Sie ein in die Grundlagen der Virtualisierung im Allgemeinen und vermittelt Ihnen Grundlagen- und Spezialwissen, das Sie zum erfolgreichen Einsatz KVM-basierter Technologie benötigen.

Der Aufbau des Buches im Einzelnen:

1. **Virtualisierung**
   bietet einen Überblick über verschiedene Ansätze zur Virtualisierung und deren Verwendung in aktuellen Virtualisierungsprodukten.
2. **KVM-Architektur**
   stellt die Bausteine vor, die zur Virtualisierung mit KVM benötigt werden.
3. **Installation**
   begleitet den Installationsprozess komplett von der Installation der KVM-Pakete bis zum Aufsetzen der ersten virtuellen Maschine.
4. **libvirt-Tools**
   verschafft Ihnen einen Überblick über alle Werkzeuge, die `libvirt` Ihnen bietet, und deren Einsatzzweck.
5. **Storage**
   behandelt sämtliche Formen von Speicher im KVM-basierten Setup.

6. **Netzwerk**
   leitet Sie beim Vernetzen Ihrer KVM-Instanzen an.
7. **Deployment**
   befasst sich mit den gängigsten Methoden zum »Ausrollen« von KVM im großen Stil.
8. **Backup**
   gibt einen kurzen Einblick in mögliche Backup-Strategien im KVM-Setup.
9. **Migration**
   präsentiert mögliche Migrationspfade von anderen Virtualisierungsprodukten nach KVM sowie von physikalischen Maschinen nach KVM.
10. **Hochverfügbarkeit**
    stellt Lösungen zur Hochverfügbarkeit mit KVM vor. Live-Migration wird ebenso behandelt wie der prinzipielle Aufbau eines Virtual System Cluster.
11. **Troubleshooting**
    bietet Lösungen für die am häufigsten auftretenden Probleme in KVM-Setups an.

Systemvoraussetzungen

Die im Buch beschriebenen Vorgehensweisen wurden auf den zur Entstehungszeit gängigen Enterprise-Distributionen (Red Hat Enterprise Linux, SUSE Linux Enterprise Server und Ubuntu) entwickelt und getestet.

Mit Updates oder Service Packs der Distributionen ändern sich Versionsnummern und enthaltene Funktionalität der verwendeten Software. Abwärtskompatibilität zu früheren Versionen sollte gewahrt sein. Somit bleibt der Inhalt dieses Buches über Versionssprünge hinweg aktuell. Von wichtigen Neuerungen oder Änderungen erfahren Sie, indem Sie die ChangeLogs oder Release-Notes wichtiger Softwarekomponenten im Auge behalten.

Die im Zusammenhang mit KVM wichtigsten Komponenten sind:

- kernel
- qemu-kvm
- libvirtd
- virt-manager

Typografische Konventionen

Folgende typografische Konventionen finden in diesem Buch Verwendung:

- *Kursivschrift*
  für Fachbegriffe und Hervorhebungen
- `Nichtproportionalschrift`
  für Konsolenausgaben, Datei- & Paketnamen, URLs
- Marginalien
  für ergänzende Bemerkungen, Verweise auf weitere Informationen und zur Kenntlichmachung der Distributionsunterschiede

Neu eingeführte Begriffe werden kursiv dargestellt und jeweils bei der ersten Erwähnung erklärt. Ein kurzes Glossar am Ende des Buches erleichtert das Nachschlagen der wichtigsten Begriffe.

Weitere Informationen

Ein komplexes und dynamisches Thema wie Virtualisierung lässt sich mit allen Details unmöglich in einem einzigen Buch komplett abdecken. An entsprechenden Stellen sind daher Links und Hinweise zu weiterführenden und aktuellen Informationen untergebracht.

Danksagungen

Dank an Anke Börnig und Jana Jaeger für den letzten Schliff an der Rohfassung dieses Buches, an Martina Dejmek für die schönen Grafiken, an Christian Berendt, den umsichtigen »Supervisor«, die Testleser und Tester Jeremias Brödel, Uwe Grawert, Karsten Keil, Florian Kellmer, Thomas Korber, André Nähring, Rico Sagner, Florian Sojer, Michael Steinfurth und Philipp Westphal, die in ihrer Freizeit mit Sachverstand und kritischem Auge Fehler aufgespürt und bei deren Beseitigung geholfen haben, an Ursula Zimpfer für ihre aufmerksame und konstruktive Korrektur, an Michael Barabas vom dpunkt.verlag für die jederzeit sehr gute Zusammenarbeit …

… und an alle, die Dank verdient hätten und hier nicht erwähnt werden.

Und nun wünschen wir Ihnen viel Freude beim Lesen!

Inhaltsverzeichnis

1 Virtualisierung

Dieses Kapitel bietet die Basis zum Verständnis der Virtualisierung. Ausgehend von der Begriffserklärung für *Virtualisierung* wird die Funktionsweise von Hypervisoren und virtuellen Maschinen vorgestellt. Darauf aufbauend folgt ein Vergleich zwischen den unterschiedlichen Virtualisierungsformen und ihrer praktischen Ausformungen bei aktuell gängigen Virtualisierungsprodukten. Insbesondere werden hierbei die beiden Virtualisierungslösungen Xen und VMware vorgestellt und separat betrachtet. KVM wird später in einem eigenen Kapitel detailliert dargestellt. Schließlich folgen noch Anmerkungen zum Einsatz und der Bedeutung der Virtualisierung in der gegenwärtigen Informationstechnologie.

1.1 Was ist Virtualisierung?

Virtualisierung gibt es im Bereich der Informatik schon seit Mitte der 60er-Jahre, als vor allem IBM mit Versuchssystemen unter anderem zur virtuellen Speicherverwaltung[1] die Grundlagen für ein hardwareunabhängigeres Design von Rechnersystemen legte. IBM entwickelte in den 90er-Jahren für seine Mainframe-Server (z.B. IBM System z) auch erstmals die Technik der *logischen Partitionierung* (engl. logical partition, LPAR), bei der die einfach vorhandenen Hardwareressourcen (CPUs, Arbeitsspeicher, Festplattenspeicher etc.) in mehrere Systeme – sogenannte *Partitionen* – aufgeteilt wurden. Deshalb spricht man in diesem Zusammenhang auch von *Serverpartitionierung* – im Gegensatz zu Virtualisierung, bei der etwas nachgebildet wird.

[1] Bei der virtuellen Speicherverwaltung wird den Prozessen ein vom physikalischen Hauptspeicher unabhängiger Adressraum, der *virtuelle Speicher*, zur Verfügung gestellt. Zugriffe auf den Hauptspeicher erfolgen nicht mehr direkt durch die Prozesse selbst; stattdessen werden die Speicheranforderungen von einem Speichermanager abgefangen und weiterverarbeitet. Dies ermöglicht Speicherschutzmechanismen zur Abtrennung der Adressräume von Prozessen sowie die Nutzung von Massenspeichern zur Auslagerung von Speicherinhalten, die sonst die tatsächlich vorhandene Speicherkapazität überschreiten würden.

Die logische Partitionierung wird auf der Hardwareebene realisiert und hat daher kaum Performance-Einbußen zur Folge. Sie sorgt beispielsweise beim Arbeitsspeicher dafür, dass jeder Partition ein bestimmter Adressierungsbereich ohne Überschneidungen zugewiesen wird. Zur Prozessornutzung hatte in der Anfangsphase der logischen Partitionierung jede Partition ihre eigene Zentraleinheit (CPU); später dann konnte mit der Technik der *Mikro-Partitionierung* ein Prozessor mehrere logische Partitionen verwalten. Diese Techniken wurden im Laufe der Zeit auch in andere Großrechnerserien übernommen. Je nach Großrechnerserie können solche Systeme ihre Ressourcen auf 60 oder mehr logische Partitionen aufteilen. Aber nicht nur die Aufteilung von Rechnerressourcen, auch die Bündelung zu sogenannten *Clustern* hat der Virtualisierung neue Aufgabengebiete erschlossen. Statt teurer Großrechner wird für das Supercomputing immer mehr Standardhardware eingesetzt, deren Rechenleistung durch Virtualisierung zusammengefasst werden kann.

Im Laufe der Zeit haben sich weitere unterschiedliche Konzepte und damit verbundene Technologien – sowohl bei der Hardware als auch bei der Software – herausgebildet. Die Entwicklung hat sich – abgesehen von der Hardwareunterstützung (siehe Abschnitt 1.3.4, S. 10) – jedoch zunehmend in Richtung Software verschoben. Die heutigen Anbieter von Virtualisierungslösungen sind in den meisten Fällen Softwarefirmen.

So unterschiedlich die Lösungen virtueller Systemumgebungen auch sein mögen, sie alle dienen dem einen Zweck: vorhandene Systemressourcen effizient und sicher zu nutzen. Wenn es auch keine allgemeingültige Definition für *Virtualisierung*[2] gibt, so können doch alle diese Lösungen mit der folgenden Definition beschrieben werden:

Definition: Virtualisierung

Als Virtualisierung bezeichnet man Techniken, die Ressourcen eines Rechners aufteilen oder zusammenfassen und dabei das Funktionsverhalten der realen Maschine kapseln.

Unter Ressourcen versteht man im Zusammenhang mit der Virtualisierung Prozessor(en), Hauptspeicher sowie I/O-Ressourcen des Netz-

[2] Der Wortbedeutung nach ließe sich *Virtualisierung* (lat.: virtus = Mannhaftigkeit, Tüchtigkeit, Vermögen, Kraft; Tugend, ...) als *mit der Fähigkeit ausgestattet* oder *mit dem Vermögen ausgestattet* deuten.

werks und der Speichersysteme einschließlich Direct Memory Access (DMA)-Controller, die auch als *Core-Four* bezeichnet werden.

Diese Hardwareressourcen werden von einem Wirtsystem, dem *Host*, bereitgestellt und von der verwendeten Virtualisierungslösung transparent an die virtuellen (Gast-)Systeme verteilt.

Die Aufteilung der Ressourcen nennt man – wie schon bei LPAR – *Partitionierung*, wobei jede Partition ein eigenständiges Subsystem aus den vorhandenen Ressourcen darstellt, das dem virtuellen System zur Verfügung gestellt wird und eine Abstraktion der physikalischen Systembestandteile bildet. In diesem Sinne lässt sich Virtualisierung auch als eine Abstraktionsschicht beschreiben, die sich logisch zwischen Anwendung und Ressourcen einfügt.

1.2 Hypervisor und Virtual Machine Monitor

Möglich wird diese Ressourcenaufteilung durch eine logische Schicht zwischen dem Host- und dem virtuellen Gastsystem durch den sogenannten *Hypervisor*. Beim Hypervisor oder auch *Virtual Machine Monitor* (abgekürzt VMM) handelt es sich um eine Virtualisierungssoftware, die eine Ausführungsumgebung für virtuelle Maschinen schafft und ihre Steuerung ermöglicht. Der Hypervisor ist der Kern der meisten Virtualisierungsprodukte und erlaubt einem einzelnen physischen Rechner den gleichzeitigen Betrieb mehrerer virtueller Systeme. Die Gastinstanzen (engl. guests) teilen sich dann die Hardwareressourcen des Wirts (engl. host).

Für ein virtuelles System stellt sich diese Abstraktionsschicht wie eine normale Systemumgebung dar, auf die es exklusiven Zugriff hat. Der Hypervisor täuscht dem virtuellen System vor, dass es der alleinige Nutzer der entsprechenden Ressourcen ist.

Solch ein System wird – je nach Virtualisierungslösung – »virtuelle Maschine« (abgekürzt VM) oder »Container« genannt und stellt im Falle einer virtuellen Maschine meist eine komplette virtuelle Hardwareumgebung dar. Eine virtuelle Maschine meint in diesem Zusammenhang einen vollständigen Computer, der nicht aus Hardware besteht, sondern über Software abgebildet wird. Eine VM ist die Schnittstelle, die dem Gast (oder auch der Domain) vom Host bereitgestellt wird und auf der der Gast dann läuft.

Auf diesen Subsystemen können ganze Betriebssysteme oder auch nur Teile davon in einer Laufzeitumgebung gestartet werden.[3] Sich als Betriebssystem darstellende virtuelle Maschinen wiederum können voll-

[3] Eine Laufzeitumgebung ist ein durch Software realisiertes Modell eines Computers.

ständig durch Software (z/VM), durch Software mit zusätzlicher Hardwareunterstützung oder allein durch Hardware (LPAR) realisiert werden.

Man unterscheidet zwei Typen von Hypervisor-Architekturen. Bei beiden Typen müssen Systemaufrufe eines Gastes, die direkt auf privilegierte Hardware (Speicher/CPU/Interrupts) zugreifen wollen, von der Hypervisor-Schicht abgefangen und interpretiert werden.[4]

Ein *Typ-1-Hypervisor* läuft ohne weitere Software direkt auf der Hardware und arbeitet dadurch recht ressourcenschonend. Der Hypervisor fängt privilegierte Operationen, die exklusiven Zugriff auf die Hardware brauchen, ab und ersetzt sie durch unprivilegierte Operationen. Typ-1-Hypervisoren sind sehr schlank und robust und gelten als die performantesten Servervirtualisierungsprodukte. Die virtuellen Maschinen nutzen die vom Hypervisor bereitgestellten Ressourcen. Allerdings muss der Hypervisor selbst die Treiber für die Hardware mitbringen. Bekannte Vertreter des Typ-1-Hypervisors sind beispielsweise IBM z/VM, Xen, VMware ESX oder Sun Logical Domains.

Ein *Typ-2-Hypervisor* dagegen setzt als Anwendung in Form der Virtualisierungssoftware auf ein vollwertiges (Host-)Betriebssystem auf und kann daher die Gerätetreiber des Betriebssystems, auf dem er läuft, nutzen. Im Wesentlichen funktioniert ein Typ-2-Hypervisor also wie ein Typ-1-Hypervisor, nur dass ein Hostbetriebssystem die Verwaltung der virtuellen Systeme übernimmt. Beispiele für den Typ-2-Hypervisor sind VMware Server/Workstation, Microsoft Virtual PC, QEMU, Parallels Workstation/Desktop.

Bei der x86-Architektur wurden entsprechende Funktionen ursprünglich über die Software des Hypervisors realisiert. Mit der Einführung der hardwareunterstützten Virtualisierung (siehe Abschnitt 1.3.4, S. 10) helfen spezielle Virtualisierungsfunktionen des x86-Prozessors (Intel-VT, AMD-V) bei der Ausführung solcher Systemaufrufe.

1.3 Virtualisierungstechniken

Mit der Zeit haben sich vier grundlegende Techniken zur Virtualisierung herausgebildet, die in den folgenden Abschnitten ausführlicher dargestellt werden: die Virtualisierung auf Betriebssystemebene, auch Prozessvirtualisierung genannt, die Vollvirtualisierung, die Paravirtualisierung und die hardwareunterstützte Virtualisierung.

[4] Näheres dazu findet sich in Abschnitt 1.3.4, Die Ringproblematik, S. 11ff.

1.3.1 Prozessvirtualisierung

Bei der Prozessvirtualisierung werden Prozesse auf Betriebssystemebene über unterschiedliche virtuelle Prozessräume verteilt. Den Prozessen wird dabei vorgespielt, sie könnten eine komplette Rechnerumgebung benutzen, sie sind aber tatsächlich isolierte Userspace-Instanzen (»Container« oder »Jails«) eines darunterliegenden Betriebssystems, weshalb auch der Begriff »Betriebssystemvirtualisierung« gebräuchlich ist. Dies geschieht auf Basis des bereits laufenden Kernels, es wird also kein neuer Kernel gestartet. Den einzelnen Prozessgruppen wird eine virtuelle Laufzeitumgebung innerhalb eines Containers zur Verfügung gestellt. Durch die unterschiedlichen Prozessräume innerhalb der Container entsteht der Eindruck mehrerer unabhängiger Systeme. Es gibt aber nur einen Prozessraum und dessen Kernel, der für eine strikte Trennung zwischen den einzelnen virtuellen Prozessräumen der Container sorgt. Durch die virtuellen Prozessräume lassen sich mehrere virtuelle Systeme logisch voneinander trennen.

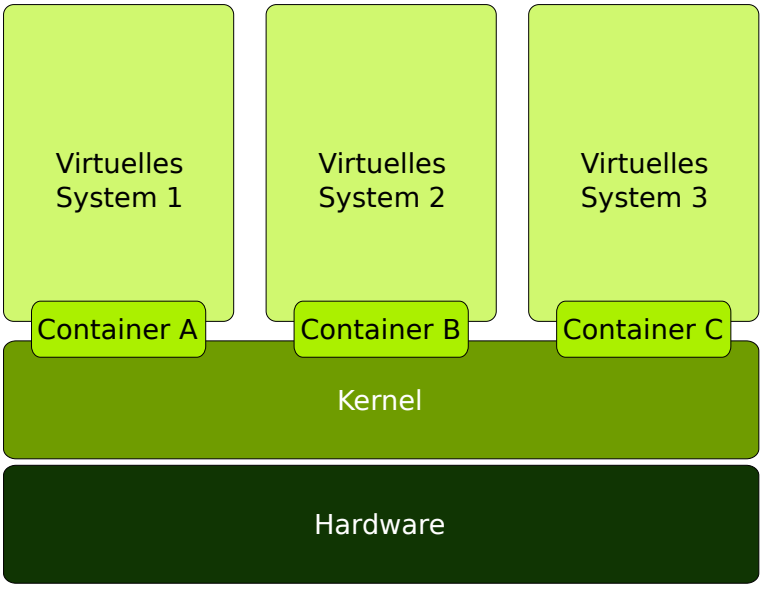

Abb. 1-1
Prozessvirtualisierung – ein Kernel für alle virtuellen Systeme

Da sich diese Art der Virtualisierung an Prozessen und Prozessgruppen orientiert, spricht man auch von Prozessvirtualisierung. Diese Technik wird überwiegend dazu verwendet, Anwendungen so voneinander zu isolieren, dass ein Angreifer, der die Kontrolle über so eine Anwendung erlangt hat, nicht ohne Weiteres Zugriff auf die anderen Container erlangen kann. Solaris Zones und FreeBSD jails sind die klassischen Vertreter für Prozessvirtualisierung, aber auch die Linux-Projekte

VServer und OpenVZ fallen in diese Kategorie. Auch beim User Mode Linux kann man von Prozessvirtualisierung sprechen. Es nimmt aber eine Sonderstellung ein, da dort ein spezieller User-Mode-Kernel unter Kontrolle des Hostkernels läuft.

Da es nur eine Kernel-Instanz gibt, die bereits Bestandteil des verwendeten Betriebssystems ist, gestaltet sich der Einsatz dieser Virtualisierung sehr einfach. Sie lässt sich gut und ohne großen Aufwand in bestehende Systeme integrieren. Dabei sind keine großen Performance-Einbrüche zu befürchten, da durch die Nutzung einer einzelnen Kernel-Instanz kein Prozessoverhead entsteht. Durch die Verwendung eines Kernels ergibt sich aber der Nachteil, dass alle Container auf eben diesen angewiesen sind. Somit können keine unterschiedlich konfigurierten Kernel-Instanzen oder gar Betriebssysteme laufen. Auch ist es nicht möglich, Laufzeitparameter des Kernels aus den Containern heraus zu verändern. Darüber hinaus ist die Implementierung der Prozessvirtualisierung überaus komplex, da für autarke Systeme eine vollständige Trennung aller Prozesse notwendig ist.

1.3.2 Vollvirtualisierung

Bei der Vollvirtualisierung (engl. full virtualization) wird die logische Schicht zwischen dem Host und dem Gast durch eine Softwarekomponente realisiert, die sich *Hypervisor* oder *Virtual Machine Monitor* (VMM) nennt. Dieser bildet das Bindeglied zwischen dem virtualisierten Gastsystem und dem Hostsystem, indem er die Hardwarezugriffe der virtuellen Maschine über den Host an die physikalische Hardware weiterreicht.

Mittels Vollvirtualisierung gelang es erstmals, beliebigige x86-Betriebssysteme virtuell auf der Intel-Architektur laufen zu lassen. Diese kann somit als *Urahn* der Intel-Virtualisierung bezeichnet werden. Bei der Vollvirtualisierung sind die Gastsysteme und deren Kernel ohne weitere Anpassung in der virtuellen Umgebung lauffähig. Ein Gastsystem benötigt lediglich die passenden Treiber der emulierten Hardware. Dabei ist man im Vergleich zur Prozessvirtualisierung nicht auf das auf dem Host laufende Betriebssystem beschränkt. Diese Freiheit wird durch die Emulation der Hardwarekomponenten erreicht. Im Unterschied zur vollständigen Emulation (siehe Abschnitt 1.3.5, S. 15) werden CPU-Befehle aber ohne weitere Übersetzung an die physikalischen Prozessoren des Hosts durchgereicht.
Als verdeutlichendes Beispiel sei hier der Zugriff auf den Arbeitsspeicher genannt.[5] Der physikalisch vorhandene Arbeitsspeicher wird innerhalb vom Virtual Machine Monitor durch eine *Shadow Page Table*

[5] Diese Erklärung trifft beim Einsatz von *Nested Page Tables* (NPT) nicht mehr zu. Mehr dazu in Abschnitt 1.3.4, S.10

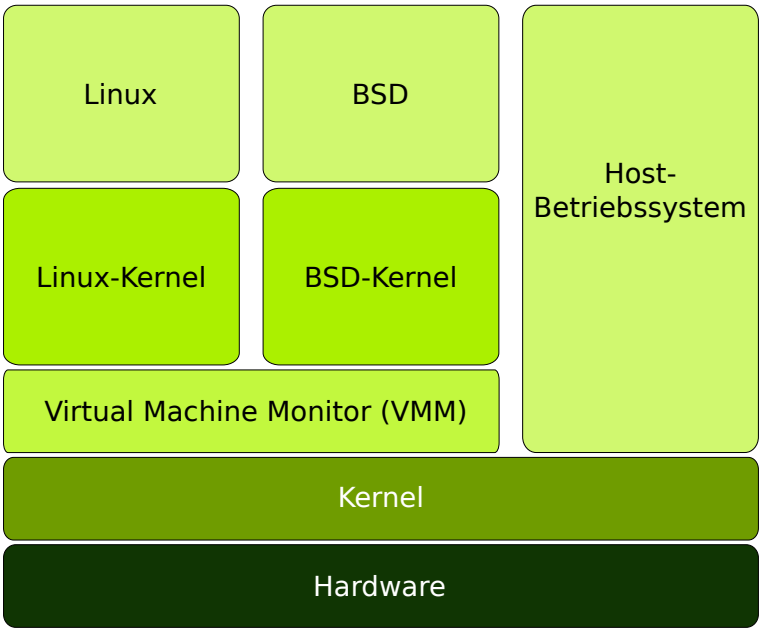

Abb. 1-2
Vollvirtualisierung – der VMM als Bindeglied zwischen Gast- und Host-Kernel

dargestellt. Mit deren Hilfe wird der Arbeitsspeicher einer virtuellen Maschine auf den des Hostsystems abgebildet, ohne dass diese einen direkten Zugriff darauf erhält. Am einfachsten kann man sich die Shadow Page Table als eine Tabelle vorstellen, in der der Speicherbereich X einer virtuellen Maschine dem Bereich Y im realen Arbeitsspeicher zugeordnet wird. Das Gastsystem sieht immer nur den Bereich X und erst der Virtual Machine Monitor setzt Zugriffe auf diesen Bereich in den realen Bereich um. Das Gastsystem kann nicht beurteilen, ob es in einer virtuellen Maschine läuft oder auf realer Hardware, weshalb diese Zugriffe als *transparent* bezeichnet werden. Alle Zugriffe auf reale Hardware werden über den Virtual Machine Monitor gesteuert.

Die Vollvirtualisierung hat den Vorteil, dass – wie bereits erwähnt – keine Anpassungen am zu virtualisierenden System nötig sind. Dies macht diese Virtualisierungslösung sehr einfach anwendbar. Das Prinzip, nach dem bei der Vollvirtualisierung gearbeitet wird, lässt sich auf jede moderne Prozessorarchitektur anwenden. Dadurch dass keinerlei Anpassung auf Betriebssystemebene notwendig ist und eine Umwandlung der Hardwarezugriffe über eine Softwarelösung realisiert wird, hat die Vollvirtualisierung den Nachteil, einen sehr großen Overhead zu erzeugen, der sich entsprechend in der Performance niederschlägt.[6]

[6]Durch den Einsatz von paravirtualisierenden virtio-Treibern (siehe Abschnitt 2.3, S. 31) kann die Performance einer Vollvirtualisierung jedoch enorm gesteigert werden.

Die Vollvirtualisierung ist weit verbreitet mit entsprechend bekannten Produkten aller derzeit relevanten Virtualisierungsanbieter: So nutzen neben VMware beispielsweise die VirtualBox von Oracle oder auch Microsofts Virtual PC eine Vollvirtualisierung mit Hardwareunterstützung. Auch Xen gehört in diese Reihe, wobei Xen zusätzlich noch Paravirtualisierung beherrscht.

1.3.3 Paravirtualisierung

Bei der Paravirtualisierung wird die Implementierung der Virtualisierung in einen host- und einen gastspezifischen Teil aufgeteilt. Dabei wird vom Hostsystem eine Schnittstelle bereitgestellt, die vom Gastbetriebssystem unterstützt werden muss.[7] Durch die entsprechende Implementierung dieser definierten Schnittstellen ist es dem Gast möglich, alle privilegierten Aufgaben aktiv an den Virtual Machine Monitor (Hypervisor) durchzureichen. So entfällt dessen Aufgabe, den Gast zu überwachen. In den virtuellen Maschinen kommen dazu Frontend-Treiber zum Einsatz, die jeweils mithilfe eines korrespondierenden Backend-Treibers in einem privilegierten System die Hardware direkt ansprechen (siehe Abb. 1-3). Die Kommunikation zwischen Front- und Backend findet über einen gemeinsam genutzten Speicherbereich, den *Event-Channel*, statt.

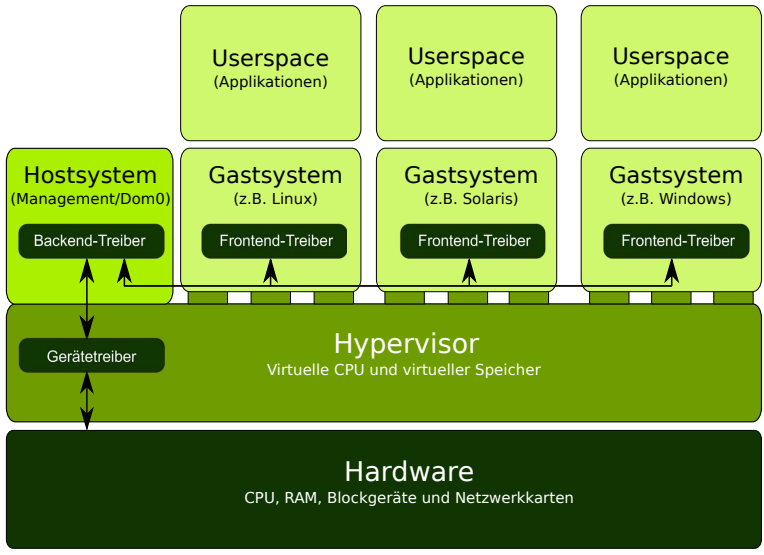

Abb. 1-3
Paravirtualisierung – angepasste Gastsysteme

[7] *Para* (griechisch: neben, bei, ...) bedeutet in diesem Zusammenhang so viel wie *miteinander*.

Die Schnittstelle zwischen den Host- und Gastsystemen wird durch eine Erweiterung des CPU-Befehlssatzes in Form sogenannter *Hypercalls* realisiert. Dazu muss der Kernel des Gastbetriebssystems modifiziert werden: Alle Bereiche des Kernelcodes, die privilegierte Befehle aufrufen, müssen so umgeschrieben werden, dass diese als Hypercalls die Funktion des Hypervisors nutzen. Hypercalls sind Funktionsaufrufe, die an den Hypervisor weitergeleitet werden, und funktionieren analog zu Systemaufrufen: So wie Systemaufrufe es Prozessen im Userspace gestatten, privilegierte Operationen über den Kernel aufzurufen, ermöglichen Hypercalls es dem Gastkernel, privilegierte Operationen über den Hypervisor aufzurufen.[8] Der Virtual Machine Monitor verschiebt sich vom Userspace in den Kernelspace und wird im Zusammenhang mit der Paravirtualisierung meist *Hypervisor* genannt. Durch die Verschiebung in den Kernelspace wird das Hostbetriebssystem selbst zu einem virtuellen System, das lediglich mehr Privilegien als die Gastsysteme hat. Auch die im System vorhandenen Ressourcen lassen sich dadurch hardwarenah partitionieren. Die partitionierte Hardware kann durch das angepasste Gastsystem mithilfe der neuen Befehlssätze angesprochen werden. Dadurch entfällt die performancelastige Emulation der entsprechenden Hardware. Das Gastsystem ist sich durch die Anpassung auch darüber bewusst, in einer virtuellen Umgebung zu laufen, und der Hypervisor muss diese nicht überwachen, sondern stellt nur eine Laufzeitumgebung bereit, die es dem Gast erlaubt, auf die partitionierte Hardware zuzugreifen.

Dieses System bezeichnet man bei Xen als Dom0.

Somit wird, im Vergleich zum Virtual Machine Monitor bei der Vollvirtualisierung, der Aufgabenbereich umgekehrt. Der Virtual Machine Monitor sorgt nicht mehr dafür, dass privilegierte Operationen abgefangen und auf das Hostsystem umgeleitet werden, sondern der Gast meldet solch eine Operation beim Hypervisor an und kann dank des erweiterten Befehlssatzes direkt auf die entsprechenden Komponenten zugreifen. Dafür sind die schon erwähnten Hypercalls nötig. Durch die Befehlssatzerweiterung wird quasi eine neue Architektur definiert; so entsteht z.B. aus der x86-Architektur mit der entsprechenden Befehlssatzerweiterung für Xen die Architektur x86/xen.

Damit das Gastsystem in einer paravirtuellen Umgebung laufen kann, muss es erst einmal portiert werden, um die entsprechenden Befehlssätze zu beherrschen. Diese Anpassung benötigt wenige tausend Zeilen zusätzlichen Programmcode. Allerdings erzeugt man dadurch nicht mehr quelloffene Systeme (engl. closed source) und ist auf die Unterstützung des Herstellers angewiesen. Da aber nicht jeder Hersteller

[8] Um die Zahl der dabei entstehenden aufwendigen Kontextwechsel zu reduzieren, können diese Funktionsaufrufe gruppiert werden.

entsprechende Änderungen vornehmen kann oder will, können nicht alle Betriebssysteme rein paravirtualisiert betrieben werden.[9]

Paravirtuelle Lösungen haben durch den Wegfall der Hardwareemulation den Vorteil einer hohen Performance. Dadurch, dass Gastsysteme auf die zu verwendende Lösung portiert werden müssen, um die nötige Unterstützung für die Befehlssatzerweiterungen zu bieten, muss man bei proprietären Systemen auf die Unterstützung des entsprechenden Herstellers zurückgreifen.

Auch wenn die Idee hinter der Paravirtualisierung schon älter ist, ist die entsprechende Umsetzung auf der x86-Architektur noch relativ jung. Am bekanntesten dürfte die Realisierung mittels Xen sein. Ob Xen in Zukunft das paravirtualisierte Modell zugunsten der hardwareunterstützten Virtualisierung aufgeben wird, kann momentan noch nicht beurteilt werden.

1.3.4 Hardwareunterstützte Virtualisierung

Was auf anderen Architekturen schon seit langer Zeit fest implementiert ist, wurde auf der x86-Architektur erst in den letzten Jahren umgesetzt. Die Rede ist von der Implementierung der Virtualisierung in der CPU, der *hardwareunterstützten Virtualisierung* (engl. hardware assisted full virtualization, auch *Native Virtualisierung*). Eines der bekannteren Beispiele ist das schon eingangs erwähnte *Locigal Partitioning* (LPAR) auf der System-p- und System-z-Architektur von IBM. Bei LPAR wird eher eine Art Paravirtualisierung auf Hardwareebene realisiert, was sie zu der momentan performantesten Virtualisierungslösung macht. LPAR ist allerdings nicht auf x86-Systemen verfügbar. Für diese wurde eine Hardwareunterstützung für die Vollvirtualisierung jeweils eigenständig von den Firmen Intel und AMD entwickelt. Die Entwicklung von Intel trug dabei den Codenamen *Vanderpool* und wird jetzt *Intel Virtualization Technology for x86*, kurz VT-x, genannt. AMD entwickelte seine Lösung, die jetzt als *AMD Virtualization*, kurz AMD-V, bekannt ist, unter dem Codenamen *Pacifica*. Beide Hersteller präsentierten ihre Entwicklungen erstmals 2005. Die zwei Lösungen sind – trotz vieler Analogien – nicht zueinander kompatibel, d.h. eine Virtualisierungssoftware muss für AMD-V eine andere Unterstützung anbieten als für Intels VT-x.

[9] Umgehen lässt sich dieses Problem mit Prozessoren, die eine Hardwareunterstützung für die Virtualisierung bieten, in Kombination mit speziellen Treibern: Damit können unmodifizierte Betriebssysteme (z.B. Microsoft Windows), die keine Portierung anbieten, mit Xen oder VMware ESX Server paravirtualisiert laufen.

Eine Hardwareunterstützung verringert zwar den Prozess-Overhead der Virtualisierung selbst; dennoch ist ohne weitere Unterstützung mit großen Performance-Einbußen zu rechnen, die auf I/O-Einschränkungen zurückzuführen sind, da die Treiber für das Gastsystem emuliert werden müssen und somit den Flaschenhals bilden. Dieses Problem lässt sich entweder durch den Einsatz paravirtualisierter Treiber oder durch eine spezielle Hardwareunterstützung für die I/O-Virtualisierung umgehen. Besonders die Hardwareunterstützung hat in den letzten Jahren große Fortschritte gemacht. So sind in den letzten Jahren unter anderem Funktionen wie PCI Passthrough, Single Root I/O Virtualization (SR-IOV) und Hardware-Assisted Paging (HAP), auch als *Nested Page Tables* (NTP) bekannt, hinzugekommen. Bei PCI Passthrough, oder auch *I/O Memory Mapping Unit* (IOMMU), wird es den Gästen möglich, physikalische Geräte direkt zu verwenden. Auf Intel-Systemen wird diese Funktion VT-d (Virtualization Technology for Directed I/O) genannt und benötigt sowohl die Unterstützung durch die CPU als auch durch den Chipsatz. Auf AMD-Systemen heißt diese Funktion AMD-Vi und ist seit Version 3 Bestandteil vom Hypertransportprotokoll und benötigt ebenfalls unterstützende Prozessoren. Mit IOMMU kann jedoch das Gerät nur von einem Gast gleichzeitig verwendet werden. Diese Beschränkung wird durch SR-IOV umgangen, das eine Erweiterung des PCI Express Standard ist. Es ist auf Intel-Systemen auch Bestandteil der *Virtualization Technology for Connectivity* (VT-c).

Bei *Nested Page Tables* (NPT) wird den Gästen direkter Zugriff auf den physikalischen Speicher gewährt, sodass keine Shadow Page Table mehr notwendig ist. Dadurch entfällt ein Großteil des ansonsten notwendigen Virtualisierungs-Overheads. Bei Intel wurde dieses Feature *Extended Page Tables* (EPT) genannt und ist auf der Nehalem-Architektur verfügbar. Bei AMD wurde diese Technik als Rapid Virtualization Indexing (RVI) mit Prozessoren der Barcelona-Reihe eingeführt.

Allen Ansätzen gemeinsam ist die Rolle des Hypervisors, der immer mit der höchsten Privilegienstufe (siehe Abschnitt 1.3.4 »Die Ringproblematik«) ausgeführt wird und stets die virtuellen Betriebssysteminstanzen und deren Zugriff auf die Ressourcen kontrolliert.

Die Ringproblematik

x86-kompatible CPUs enthalten vier unterschiedliche Privilegienstufen für den Speicherzugriff und den Umfang des nutzbaren CPU-Befehlssatzes. Diese Privilegienstufen werden *Domains* genannt und als Ringe (Ring 0 bis Ring 3) dargestellt (siehe Abb. 1-4).

Der innerste Ring 0 verfügt im sogenannten *Supervisor Mode* über alle Rechte der CPU. Privilegierte Prozessoranweisungen, die den Prozessorzustand verändern und die direkte Zugriffe auf die Register, den Arbeitsspeicher und die übrige Hardware ausführen, können nur in Ring 0 abgearbeitet werden.

So läuft auf Ring 0 normalerweise der Kernel eines Betriebssystems, weshalb er auch als *Kernelspace* bezeichnet wird. Nach außen hin werden die Rechte immer weiter eingeschränkt. Während also der Betriebssystem-Kernel privilegierte Prozessoranweisungen im Ring 0 verwenden darf, sind die Anwendungen auf den Benutzermodus – den *Userspace* – von Ring 3 beschränkt.[10] Die Kommunikation zwischen den Ringen erfolgt über sogenannte *Gates*, definierte Schnittstellen, die Aufrufe und deren Rückmeldungen weiterreichen.

Ein Prozess kann grundsätzlich nur innerhalb eines einzelnen Rings ausgeführt werden. Er kann sich nicht selbst in eine andere Privilegienstufe versetzen.

Will nun ein Prozess eines weniger privilegierten Rings eine privilegierte Operation ausführen, muss der Prozess eine *Exception* erzeugen, die diese Operation abfängt und in einem höher privilegierten Ring ausführt.

Der Prozess veranlasst dazu mittels eines *Systemaufrufs* (engl. system call) einen *Kontextwechsel*, bei dem er die Kontrolle über den Prozessor vorübergehend an den Kernel übergibt. Nachdem die Anfrage abgearbeitet wurde, gibt der Kernel die Kontrolle über den Prozessor wieder an den Prozess des Benutzermodus zurück. Der kann dann im Programm dort weitermachen, wo er vor dem Kontextwechsel gestoppt wurde.

Der Userspace-Prozess verlässt dabei niemals den nichtprivilegierten Ring und kann so auch nicht Gefahr laufen, andere Prozesse oder gar die Stabilität des Systemkerns selbst zu gefährden, da nur vertrauenswürdiger Code aus dem Kernel im privilegierten Modus ausgeführt wird.

Zweck dieser Ringarchitektur ist es, die Stabilität und Sicherheit des Systems zu gewährleisten, indem Prozesse auf ihren erlaubten Kontext beschränkt und so davon abgehalten werden, sich gegenseitig unerwünscht zu beeinflussen. Vor allem die Prozesse des Betriebssystem-Kernels selbst bleiben so vor Anwendungsprogrammen im Benutzer-

[10] Die Ringe 1 und 2 finden kaum Verwendung und spielen in diesem Zusammenhang keine Rolle. Ring 1 und Ring 2 wurden bei der x86-Architektur gelegentlich zum Priorisieren von Gerätetreibern eingesetzt. Die meisten aktuellen Prozessorarchitekturen unterscheiden nur zwei Ringe, weshalb die gängigen Betriebssysteme aus Portabilitätsgründen auf die Nutzung von Ring 1 und Ring 2 verzichten.

modus geschützt. Es wird stets nur vertrauenswürdiger Code aus dem Kernel im privilegierten Modus ausgeführt.[11]

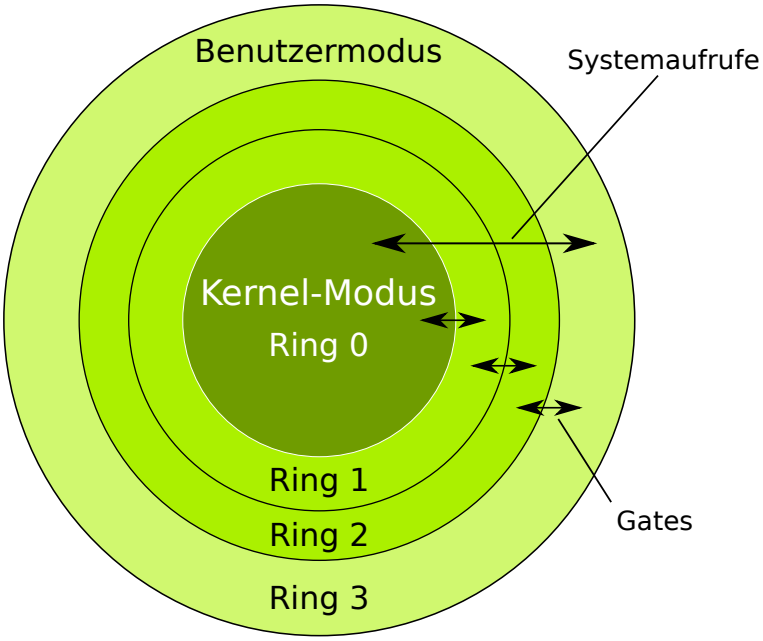

Abb. 1-4
Ringarchitektur – Systemaufrufe müssen vom Hypervisor im Ring 0 abgefangen werden.

Bei nicht virtualisierten Systemen arbeiten der Kernel (und die Hardwaretreiber) also im Ring 0 und die Anwendungen im weniger privilegierten Ring 3. Wenn eine Anwendung einen privilegierten Befehl ausführen will, muss der Befehl an den Kernel in Ring 0 übergeben werden.

Daraus ergibt sich bei der Virtualisierung aber die sogenannte »Ringproblematik«: Sollen mehrere Gastsysteme unmodifiziert laufen, müssten sich diese alle Ring 0 teilen, was bei gleichzeitigen Zugriffen unmöglich wäre.

Für die Virtualisierung auf x86-Prozessoren wird zur Steuerung der virtuellen Maschinen folglich eine eigene Virtualisierungsschicht unterhalb der Gastsysteme benötigt, die diese Zugriffe steuert. Dazu läuft der Virtual Machine Monitor im Ring 0. Ring 0 steht dadurch dem Betriebssystem einer virtuellen Maschine nicht mehr zur Verfügung. Privilegierte Prozessoranweisungen eines Gastsystems müssen nun vom Virtual Machine Monitor erkannt, abgefangen und – mit entsprechendem Performance-Verlust – weiterverarbeitet werden.

[11] Kann eine Exception nicht abgefangen werden, entsteht ein *General Protection Fault* (GPF) und der auslösende Prozess stürzt ab; bei einem Kernel – das virtualisierte Gastsystem läuft einschließlich Kernel im weniger privilegierten Ring 3 – wäre das ganze System betroffen …

Aktuelle CPU-Generationen von Intel und AMD[12] bieten auf Basis der x86-Architektur eine direkte Hardwareunterstützung für die Virtualisierung, die diese privilegierten Befehlsaufrufe verarbeiten kann.

Die wesentliche Neuerung dabei ist die sogenannte *Hypervisor-Schicht*, die durch ein Aufsplitten von Ring 0 gebildet wurde. Die Ausführung des Virtual Machine Monitor (Hypervisors) wird in eine neue Schicht unterhalb von Ring 0 – auch *Ring -1* genannt – verschoben und läuft dann in einem speziellen, privilegierten Modus, dem *Root Mode Privilege Level*. Dadurch ist es möglich, einen nicht modifizierten Gastkernel im Kernelspace von Ring 0, jedoch in einem weniger privilegierten *Non Root Mode*, laufen zu lassen und ihm die Ausführung nichtkritischer Systemaufrufe zu erlauben.

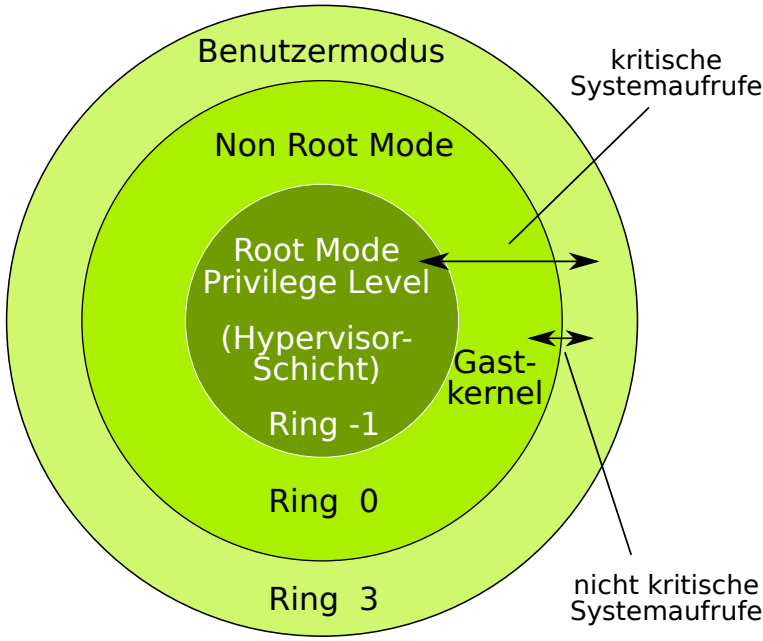

Abb. 1-5 *Ringarchitektur mit Hardwareunterstützung und Hypervisor im Ring -1*

Systemaufrufe bedürfen so nicht mehr in jedem Fall des Einschreitens des Hypervisors; solange keine kritischen Zugriffe[13] erfolgen, kann ein Gastsystem die Systemaufrufe für Anwendungen des Userspace selbst ausführen. Sobald ein Gastsystem versucht, einen kritischen Befehl auszuführen, greift die Hardwareunterstützung des Prozessors ein. Sie fängt diesen Befehl ab und übergibt ihn an den Hypervisor, der den Be-

[12] Prozessoren vom Typ VT-x (Intel) und AMD-V (AMD) sind seit 2006 erhältlich.

[13] Kritisch sind alle Aufrufe, die exklusiven Hardwarezugriff erfordern, wie Schreibvorgänge auf die Festplatte, die einen Interrupt auslösen.

fehl im Kernel-Modus weiterverarbeitet. Der Hypervisor sorgt dabei für einen kontrollierten Zugriff der Gäste auf die Ressourcen des Hostrechners. Durch diese Aufteilung können privilegierte Befehle von Gastsystemen sauber von anderen Gastsystemen und vom Hostsystem getrennt und gegebenenfalls störende Auswirkungen eines privilegierten Befehls in unschädlichen Grenzen gehalten werden. Den Gastsystemen wird dabei stets das ordnungsgemäße Verhalten eines unabhängigen Rechners vorgespielt. Es entfällt die Notwendigkeit, den Kernel der Gäste für eine Hypercall-Unterstützung zu patchen oder zu paravirtualisieren.

Dadurch, dass virtuelle Maschinen ohne den Umweg über Software auf Ring 0 zugreifen können, wird ein erheblicher Performance-Gewinn gegenüber einer reinen Softwarelösung erzielt. Darüber hinaus bringt diese Technik auch eine erhöhte Sicherheit mit sich, da die Ausführung der Gäste weit weniger Emulation durch Software erfordert. Eine geringere Fehleranfälligkeit ist die Folge.

1.3.5 Emulation

Während bei der Virtualisierung die Befehle einer virtuellen Maschine größtenteils direkt auf der CPU ausgeführt werden, werden bei der *Emulation* die Befehlssätze einer kompletten Architektur durch Software auf einer anderen Architektur nachgebaut (*emuliert*). Jedes Stück Hardware – auch die CPU – wird also durch Software nachgebildet. Prinzipiell können beliebige Architekturen emuliert werden. Eine komplette Hardwareemulation ermöglicht es sogar, ein nicht modifiziertes System, das für eine andere Prozessorarchitektur als die des Hostsystems geschrieben wurde, zu betreiben.

Im Unterschied zur Virtualisierung können bei der Emulation die Zugriffe aber nicht vom Gast zum Prozessor durchgereicht werden, sondern müssen umständlich und rechenintensiv umgeschrieben werden. Das führt zwar einerseits zu einer sauberen Trennung von Host- und Gastsystemen und bietet die Möglichkeit der Verwendung von grundsätzlich beliebigen Hardwaretreibern, erfordert andererseits aber viel Rechenleistung und stellt hohe Anforderungen an den Emulator. Vorteil der Emulation ist also ihre Flexibilität, Nachteil die geringe Performance. Performance. Eine komplette Emulation der Hardware ist daher für den Produktivbetrieb nicht geeignet.

In der Virtualisierung wird die Emulation dennoch in Teilbereichen eingesetzt: Zum einen, um einem Gastsystem spezielle Hardware, beispielsweise eine bestimmte Netzwerk- oder Soundkarte, bereitstellen zu können, die sich, aus welchen Gründen auch immer, nicht virtualisieren lässt. Zum anderen können Komponenten zur Verfügung gestellt wer-

den, die für jedes System einmalig und exklusiv vorhanden sein müssen, wie beispielsweise der *Watchdog*[14].

Benötigt ein virtuelles System solch eine Systemkomponente, dann wird diese Komponente derart nachgebildet, dass sie aus Sicht der virtuellen Maschine wie ein tatsächlich vorhandenes physikalisches Gerät arbeitet.

Die bekanntesten Vertreter zur Emulation von x86-Prozessor-Architekturen sind Bochs und QEMU. QEMU ist insbesondere deshalb erwähnenswert, weil Teile aus dem QEMU-Projekt bei der Virtualisierung mit KVM genutzt werden, um bestimmte Hardwarebauteile zu emulieren. Darum ist QEMU im folgenden Kapitel auch ein eigener Abschnitt (siehe Abschnitt 2.2, S. 27) gewidmet.

1.4 Virtualisierungsprodukte

Fast alle heute relevanten Virtualisierungsanbieter nutzen primär eine Vollvirtualisierung mit Hardwareunterstützung. Im praktischen Einsatz wird unterschieden zwischen Desktop-Virtualisierung (VMware Workstation, VirtualBox, Virtual PC u.a.) und Servervirtualisierung (XEN, VMware ESXi Server, OpenVZ, Virtuozzo, Windows Server Hyper-V u.a.). KVM ist für beide Einsatzbereiche geeignet.

Hervorzuheben sind VMware und Xen, denen aufgrund ihrer Bedeutung und weiten Verbreitung jeweils ein eigener Abschnitt gewidmet wird.

1.4.1 VMware vSphere Hypervisor (ESXi)

www.vmware.de

Im Vergleich zu den bereits genannten Vollvirtualisierungslösungen wie z.B. Virtual PC geht VMware mit vSphere einen anderen Weg. Der ESXi-Server von VMware setzt eine Methode ein, die auch als *Full Virtualization with Binary Translation* bezeichnet wird.

Dabei wird die Virtualisierung des Userspace durch die Verwendung eines eigenen Kernels in den Kernelspace verschoben. Durch den Wegfall eines Hostbetriebssystems und der damit verbundenen Hardwarenähe kann VMware im Vergleich zu anderen Lösungen weitaus effektiver arbeiten. Der Virtual Machine Monitor muss aber weiterhin eine Emulationsschicht für die Hardware bieten, um den Gästen den exklusiven Zugriff auf die Hardware vorspielen zu können. Der Virtual Machine Monitor läuft im privilegierten Ring 0 und wird durch einen

[14] Ein Watchdog sorgt dafür, dass der Kernel regelmäßig ein lebenserhaltendes Signal erhält, das den voreingestellten System-Reset des Kernels unterbindet.

Mikrokernel realisiert. Infolge seiner Hardwarenähe kann er die Ressourcen sehr performant aufteilen. Mit dieser Implementierung bewegt sich der Overhead etwa im gleichen Rahmen wie bei einer Paravirtualisierung.

Ursprünglich wurde ESX als Virtual Machine Monitor verwendet. Dieser musste ein angepasstes Red Hat Linux mit einem stark angepassten Kernel starten, um ein Konfigurationsmanagement zu bieten. Der seit vSphere 4 eingesetzte ESXi-Kernel stellt selbst eine entsprechende API bereit. Durch den Wegfall des separat notwendigen Konfigurationssystems entfällt auch die Pflege eines eigenen Kernel-Tree, da dieser ja immer wieder an den proprietären Mikrokernel angepasst werden musste.

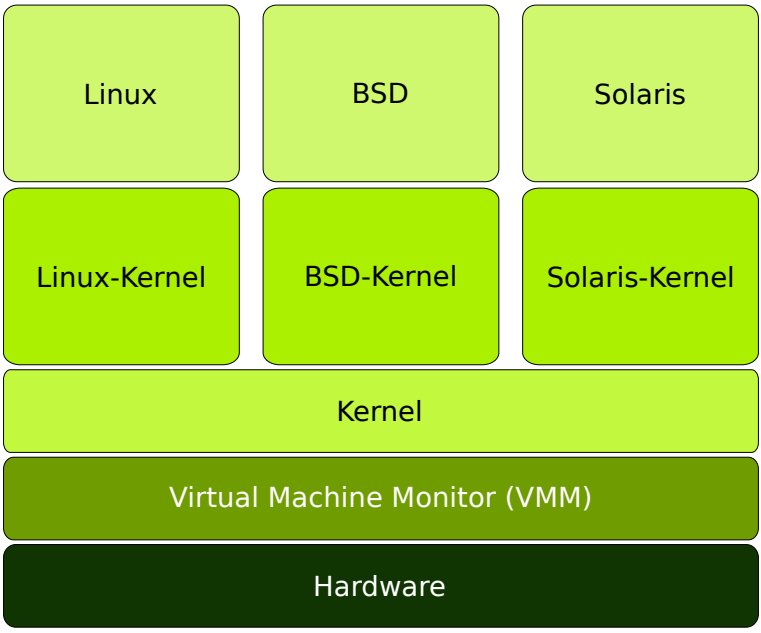

Abb. 1-6
VMware vSphere: Mikrokernel ohne Hostsystem

1.4.2 Xen

`www.xen.org`

Xen entstand ursprünglich an der britischen Universität Cambridge und wird unter der GPL entwickelt. Xen fand prominente Unterstützer wie Microsoft, Oracle, Intel und AMD, IBM, HP, Novell/SUSE und bis 2008 auch Red Hat. Um Xen zum Industriestandard zu machen, gründeten die Entwickler die Firma XenSource Inc., die 2007 von dem US-Unternehmen Citrix Systems übernommen wurde. Seitdem verantwortet Citrix das »Xen Open Source Hypervisor«-Projekt. Aktuell liegt Xen in der Version 4.1 vor.