FernUniversität in Hagen
Fakultät für Mathematik und Informatik
Lehrgebiet Rechnerarchitektur
Univ.-Prof. Dr. Wolfram Schiffmann

Ausarbeitung

A comparison of software and hardware techniques for x86 virtualization

im Rahmen des Seminars
Virtuelle Maschinen

Bernhard Biendl
Andreas Lemke

Betreuer: Professor Wolfram Schiffmann

Inhaltsverzeichnis

Kurzfassung

Bis vor kurzem kannte die x86 Architektur keine klassische “trap-and-emulate“-Virtualisierung. Statt dessen setzte man auf Virtual Machine Monitors (VMM), die mittels “binary translation“ (BT) als Interpreter die semantisch problematischen Anfragen des Gastsystems modifizieren und so eine Virtualisierung erst möglich machten. Intel und AMD bieten nun erweiterte x86-Prozessoren an, um die klassische Virtualisierung hierbei zu unterstützen.

In dieser Arbeit vergleichen wir die existierenden VMMs auf BT Basis mit den hardwareunterstützten VMMs, die die Erweiterungen der neuen Prozessoren nutzen, erläutern ihre Technologie und Arbeitsweise und versuchen uns an einem objektiven Vergleich. Dieser findet zu einem Zeitpunkt statt, an dem virtuelle Maschinen das Entwicklungsstadium verlassen haben und im produktiven Einsatz angekommen sind. Dementsprechend hart umkämpft ist auch der Markt für solche Lösungen und im Spannungsfeld der Marketingstrategien fällt es zunehmend schwer die Vor- und Nachteile einzelner Produkte herauszuarbeiten.

Schlussendlich ist es schwierig eines der Produkte als Sieger zu deklarieren, da die Entscheidung für den Einsatz eines Produkts im Enterprise-Umfeld auch wesentlich von einer bereits im Unternehmen vorhandenen Infrastruktur mit bestimmt wird. So bleibt es dem Anwender nicht erspart, unter Beachtung seiner Rahmenbedingungen selbst die für ihn passende Virtualisierungslösung zu ermitteln.

1 Anforderungen an virtuelle Maschinen

"Wer drei Wasserhähne braucht, käme kaum auf den Gedanken, dafür drei Leitungen auf sein Grundstück zu verlegen. Genau das aber tut jeder, der sich drei Server in seine Firma stellt. Die Lösung - man ahnt es - lautet: eine Leitung, Virtualisierung, drei Hähne."[Klei08]

Virtualisierung bietet sich überall dort an, wo physische Maschinen mit den ihnen zugeteilten Prozessen nicht voll ausgelastet sind und einen nicht unerheblichen Teil ihrer Betriebszeit im `idle`-Modus zubringen. Existieren dann noch mehrere Maschinen, die mit ähnlichen oder sogar gleichen Aufgaben betraut sind, ist der Einsatz von virtuellen Maschinen beinahe zwingend. Der Vorteil besteht nicht nur in der Einsparung physischer Maschinen und damit der Reduktion von diversen Betriebs- und Anschaffungskosten, sondern dient auch der Verkürzung von Ausfallzeiten. Im Fehlerfall oder während eines Wartungsintervalls lassen sich virtuelle Maschinen relativ einfach auf eine Ersatzmaschine schieben. Die Offlinezeiten sind dabei minimal.

Die Idee einer virtuellen Maschine ist nicht neu. Bereits 1974 formulierten Gerald J. Popek und Robert P. Goldberg drei wesentliche Anforderungen, die nach ihrer Auffassung von jeder virtuellen Maschine erfüllt werden müssten, damit man von einer echten Virtualisierung sprechen kann.

- **Fidelity/Genauigkeit:** Software, die auf einer virtuellen Maschine ausgeführt wird, muss exakt die selben Resultate bringen, als würde sie nativ ausgeführt. Ausgenommen davon ist lediglich ein zeitlicher Mehrbedarf.

- **Performance:** Der größtmögliche Teil aller Instruktionen des Gastsystems wird ohne Intervention des VMM direkt auf der Hardware ausgeführt. Damit soll ein Mindestmaß an Leistungsfähigkeit garantiert werden.

- **Safety/Sicherheit:** Alle physischen Ressourcen werden durch die VMM kontrolliert und im Bedarfsfall zugeteilt.

Die Palette der momentan auf dem Markt befindlichen Virtualisierungslösungen ist relativ breit, ebenso wie die dabei angewandten Technologien. Die individuellen Kosten variieren von gratis bei Open-Source bis zu mehreren Tausend Euro für proprietäre Lösungen. Die folgende Tabelle gibt einen ersten Überblick - die vollständige Übersicht findet sich bei [Spr08].

Produkt	Anbieter	Technologie	OS Host	OS Gast
Xenserver	Xensource	Hypervisor, Paravirtualisierung, Vollvirtualisierung mit Hardwareunterstützung	RedHat (modifiziert)	Linux, Windows
Xen GPL	Linux Distributoren	Hypervisor, Paravirtualisierung, Vollvirtualisierung mit Hardwareunterstützung	Linux, Solaris, FreeBSD, NetBSD	Linux, Windows, BSD, Solaris
VMware VI	VMware	Hypervisor mit Vollvirtualisierung	Linux	Linux, Windows, BSD
VMware Workstation	VMware	Emulation	Windows, Linux	Linux, Windows, BSD
Qemu	Fabrice Bellard	Emulation, optional beschleunigt durch Kernelmodul zum direkten Zugriff auf Ressourcen	Linux, Windows	Linux, Windows, BSD, andere Prozessorarchitekturen möglich

Tabelle 1: Übersicht Virtualisierungslösungen

Die Tabelle gibt nur einen Ausschnitt dessen wieder, was momentan verfügbar ist. Doch die Spalte "Technologie"macht bereits die Vielzahl der unterschiedlichen Ansätze verschiedener Virtualisierungslösungen deutlich. Sie alle lassen sich aber einer von drei Grundformen zuzuordnen bzw. sind Mischformen davon.

2 Überblick Virtualisierungsstrategien

Virtualisierung funktioniert genau dann perfekt, wenn es gelingt etwas ganz Wesentliches zu verbergen. Die Gäste, die in einer virtuellen Maschine laufen, dürfen nicht merken, dass sie nicht nativ auf der Hardware ausgeführt werden, sondern in einer gleichsam *künstlichen Umgebung* existieren. Dazu müssen alle relevanten Hardwarekomponenten wie CPU, Speicher und I/O virtuell nachgebildet werden.

Herzstück einer jeden Virtualisierungslösung ist deshalb der VMM *Virtual Machine Monitor* oder ein Hypervisor. Beide sind für die Trennung von Hardware und Gastsystemen sowie für die Verwaltung und Zuteilung von Ressourcen verantwortlich. VMM und Hypervisor unterscheiden sich von einander durch die Ebene in der sie ausgeführt werden, ihre grundsätzlichen Aufgaben sind aber vergleichbar.

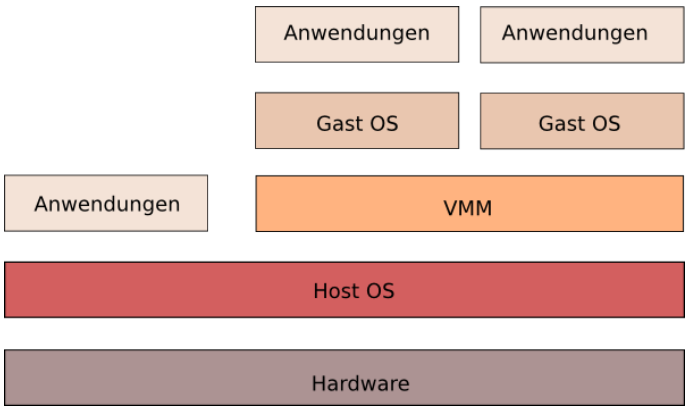

Abbildung 1: Lokalisation eines VMM

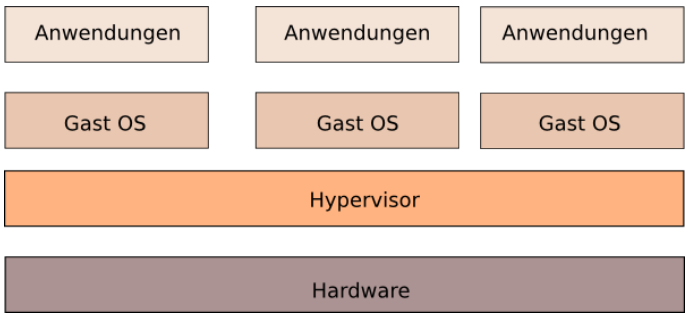

Abbildung 2: Lokalisation eines Hypervisor

Für die Virtualisierung der CPU existieren gleich drei verschiedene Technologien, die im Einzelfall auch miteinander kombiniert werden können. Jede bringt in bestimmten Bereichen Vorteile mit sich, besitzt dafür in anderen aber auch Nachteile, die im Folgenden genauer betrachtet werden.

2.1 CPU-Virtualisierung

Prozessoren stellen für die Ausführung von Programmcode verschiedenen Sicherheitslevel bereit, die als Ringe oder auch Domain bezeichnet werden. Erstmals wurde diese Technologie mit 8 Ringen 1967 zusammen mit dem Betriebssystem *Multics* eingeführt. Die Zahl der Ringe hängt von der jeweiligen Prozessorarchitektur ab und so besitzt ein x86 kompatibler Prozessor die Ringe 0 bis 3. Programmcode der in einem bestimmten Ring läuft, besitzt einen mehr oder minder privilegierten Status und hat die entsprechenden Zugriffsrechte. So läuft der Betriebssystemkernel bei Linux und Windows standardmäßig im Ring 0 und die Anwendungen im Ring 3. Die Ringe 1 und 2 bleiben ungenutzt.

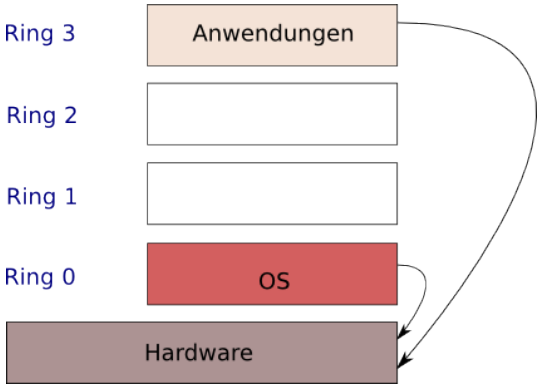

Abbildung 3: Schematische Darstellung der CPU-Ringe ohne Virtualisierung

Zwischen den Ringen existieren Gates, die einen temporären Statuswechsel etwa bei einem Systemaufruf ermöglichen. Anwendungen in Ring 3 sind damit vom Kernel als auch von anderen Anwendungen gekapselt und haben damit auch nur Zugriff auf den in ihrem Prozesskontext festgelegten Speicherbereich. Benötigen sie etwa einen I/O-Zugriff z.B. um Daten auf die Festplatte zu schreiben, müssen die einen `system call` ausführen, damit das Betriebssystem im Ring 0 die gewünschte Operation ausführt und die Daten an die Anwendung liefert.

Basierend auf diesem Ringmodell arbeiten die nachfolgend vorgestellten Virtualisierungsstrategien.

2.1.1 Full Virtualization mittels Binary Translation

Gegenüber der nativen Ausführung ändern sich bei der Full Virtualization zwei Dinge.

1. In den Ring 0 wird ein VMM installiert. Dieser trennt die Gäste teilweise von der Hardware.

2. Der Kernel des Gastbetriebssystems läuft im Ring 1, die Anwendungen arbeiten weiter im Ring 3.

Daraus resultiert folgende Situation: Die Anwendungsprogramme werden weiter direkt auf der CPU ausgeführt. Da die Prozesse des Betriebssystems jetzt im weniger privilegierten Ring 1 arbeiten, ihr Code aber für den Ring 0 geschrieben wurde werden alle Befehle, die die Rechte des Ring 0 benötigen von dem VMM „abgefangen“ und zur Laufzeit umgeschrieben. Dieser Vorgang wird als *binary translation* (siehe Kapitel 3) bezeichnet, da hier kein Quell- sondern nur Binärcode vorliegt.

Außerdem stellt der VMM jedem Gastsystem ein virtuelles BIOS sowie virtuelle Geräte (Festplattencontroller, Platten, Netzwerk, ...) und ein virtualisiertes Speichermanagement zur Verfügung.

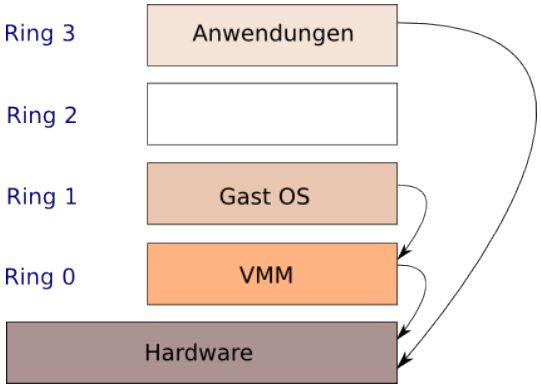

Abbildung 4: Schematische Darstellung der CPU-Ringe mit Full Virtualization

Die Tatsache, dass die Gastsysteme unverändert in der virtuellen Maschine ausgeführt werden können, ist zugleich auch der größte Vorteil dieser Technologie. Damit lassen sich auch closed source Betriebssysteme wie Microsoft Windows einsetzen. Nachteilig wirkt sich allerdings die binary translation aus. Trotz Cache-Strategien kommt es in Abhängigkeit der Anwendungen zu messbaren Performanceeinbußen.

2.1.2 Paravirtualization

Die Paravirtualization verlangt worauf die Full Virtualization verzichten kann - einen angepassten Betriebssystemkernel. Bei diesem Verfahren wird unter dem Ring 0 softwaremäßig eine zusätzliche Schicht - ein Hypervisor - eingezogen. Diese nimmt die Befehle des Ring 0 entgegen. Durch die Anpassung des Kernels wurden die Ring 0 Befehle so geändert, dass sie nicht auf der Hardware arbeiten, sondern an den Hypervisor so genannte *Hypercalls* schicken. Befehle im Ring 3 werden weiterhin direkt auf der Hardware ausgeführt. Der Vorteil liegt in dem geringen Overhead

zur Laufzeit, da alle Anpassungen bereits vorgenommen wurden. Closed Source Betriebssysteme wie Windows bleiben hier zunächst außen vor, da mangels Quellen keine Codeanpassungen möglich sind. Die ersten, von Microsoft selbst modifizierten, Entwicklerversionen wurden aber bereits gesichtet.

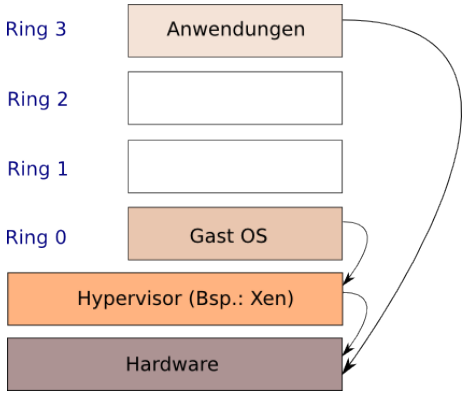

Abbildung 5: Schematische Darstellung der CPU-Ringe bei Paravirtualization

2.1.3 Hardware assistent Virtualization

Die Hardware unterstützte Virtualisierung als dritte Technologie soll die Vorteile von Full-Virtualization und Paravirtualization vereinen. Dazu haben AMD mit Pacifica und Intel mit VT-x die Architektur ihrer aktuellen x86 Prozessoren um einen zusätzlichen Ring erweitert. Dieser auch *Root Mode* genannte Ring liegt unterhalb von Ring 0 und ist damit noch höher privilegiert als die Ringe 0 bis 3, die alle im *Non-Root Mode* laufen. Die eingesetzten Gastbetriebssysteme bleiben unmodifiziert und laufen daher normal im Ring 0 und ihre Anwendungen im Ring 3. Alle Ring 0 Befehle werden aber nun vom Hypervisor im Root Mode abgefangen und von diesem auf der Hardware ausgeführt.

Zusätzlich besitzen die AMD Prozessoren einen *VMCB (Virtual Machines Control Block)*, Intel nennt es *VMCS (Virtual Machines Control Structure)*. Bei beiden handelt es sich um eine Datenstruktur in der die aktuellen Zustände der virtualisierten Gäste gespeichert werden. (siehe Kapitel 4)

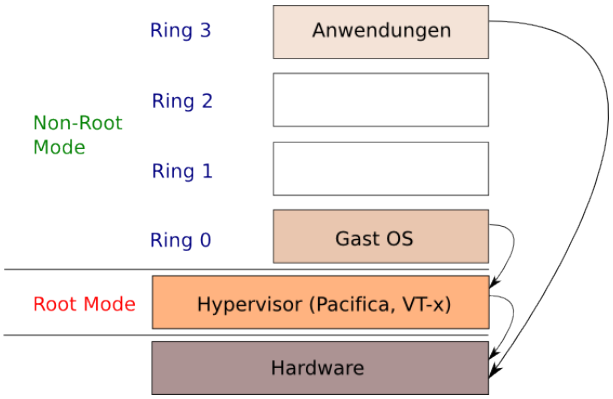

Abbildung 6: Schematische Darstellung der CPU-Ringe bei Hardwarevirtualization

2.2 Speicher Virtualisierung

Speicher-Virtualisierung bedeutet die gemeinsame Nutzung des physikalischen Speichers und das dynamische Zuweisen dessen an die virtuellen Maschinen. Die Virtualisierung des physikalischen Speichers ist dem virtuellen Speicher von modernen Betriebssystemen sehr ähnlich. Die Anwendungen sehen einen Adressbereich, der nicht zwangsweise dem physikalischen Adressbereich entspricht. Das Betriebssystem übernimmt das Zuweisen des virtuellen Speichers zum darunterliegenden physikalischen Speicher. Alle modernen x86-CPUs haben dafür eine MMU (memory management unit) und einen TLB (translation lookaside buffer), um die Performance dieses virtuellen Speicher zu optimieren. Um mehrere virtuelle Maschinen auf einem System zu beitreiben, muss ein zusätzliches Level von Speichervirtualisierung eingeführt werden. Es muss für die virtuelle Maschine die MMU virtualisiert werden. Das Gast-Betriebssystem kontrolliert die Zuordnung der eigenen virtuellen Speicheradressen zu den Adressen, die dem Gast-Betriebssystem als physikalische Adressen zugewiesen worden sind, aber das Gast-Betriebssystem hat keinen direkten Zugriff auf die physikalischen Adressen des Hostes. Der VMM ist für die Zuordnung zwischen dem physikalischen Speicher des Gastbetriebssystem und dem physikalischen Speicher des Hostes zuständig. Die Virtualisierung der MMU erzeugt daher einen gewissen Overhead für virtuelle Maschinen.

2.3 I/O- und Geräte-Virtualisierung

Die Virtualisierung von Geräten und I/O-Anforderungen hat die Aufgabe, den Zugriff zwischen der physikalischen Hardware und den virtuellen Geräten zu managen. Software-basierte I/O-Virtualisierung und I/O-Management bietet im Gegensatz zu

direktem Zugriff viele Möglichkeiten und erleichtert das Management. Die physikalische Hardware wird virtualisiert und den Gast-Maschinen in standardisiert zur Verfügung gestellt. Diese virtuellen Geräte emulieren die Hardware und setzen dabei virtuelle Requests zu Hardware-Requests um. Damit geht eine Standardisierung der virtuellen Geräte einher, welche die Portabilität der virtuellen Maschinen erhöht. Um eine virtuelle Maschine auf einen anderen physikalische Computer zu portieren, ist man von der Hardware unabhängiger, man braucht nur die gleiche Virtualisierungs-Unterstützung.

3 Binary Translation (BT)

3.1 Simple BT

Bei der x86-Virtualisierung entstehen semantische Probleme, die mit Hilfe eines Interpreters anstelle der direkten Ausführung durch die CPU abgefangen werden können. Damit können die Forderungen von Popek und Goldberg nach Fidelity und Safety gewährleistet werden, die Performance leidet jedoch darunter. Mit binary translation können sowohl die semantischen Probleme abgefangen als auch die Programme mit hoher Performance ausgeführt werden. Der *binary translator* hat folgende Eigenschaften:

- **Binary**: Der Input ist binär. Der Übersetzer hat damit keinen Zugriff auf den Quellcode.
- **Dynamic**: Die Übersetzung passiert zur Laufzeit, parallel zu bereits in der Ausführung befindlichem Code.
- **On demand**: Der Code wird erst unmittelbar vor seiner Ausführung übersetzt.
- **System level**: Der Übersetzer macht keine Annahmen über den Gast-Code. Grundlage für jegliche Übersetzung ist der x86-Befehlssatz. Damit findet bei der Übersetzung auch keine Optimierung des Codes statt.
- **Subsetting**: Der Übersetzer akzeptiert eingangsseitig den gesamten x86-Befehlssatz einschließlich aller privilegierten Befehle. Sein Output ist eine Teilmenge davon, die im wesentlichen aus *user level* Befehlen besteht.
- **Adaptive**: Der übersetzte Code ist so angepasst, dass die Antwort für das Gastsystem passt.

Während der Übersetzung unterscheidet der Translator zwischen Code, der übersetzt werden muss und Code, der identisch ausgeführt werden kann. Ein Großteil der Anweisungen muss nicht übersetzt werden. Ausnahmen bilden die Gruppen der folgenden Befehle:

- PC-relative adressing
- Direct control flow
- Indirect control flow
- Privileged instructions

Der Übersetzer versucht hierbei nicht, den Code zu optimieren sondern geht davon aus, dass dies bereits die Entwickler des Betriebssystem gemacht haben.

Der Vorteil der BT liegt darin, dass nur der Code übersetzt wird, der tatsächlich angepasst werden muss. Damit wird sowohl Fidelity, Safety als auch eine gute Performance gewährleistet.

Das folgende Beispiel, entnommen aus [VM], veranschaulicht die Arbeitsweise des Übersetzers. Ausgangspunkt ist der folgende C-Code, der die Überprüfung der Primzahleigenschaft realisiert.

```
int isPrime(int a) {
  for (int i = 2; i < a; i++) {
    if (a % i == 0) return 0;
  }
  return 1;
}
```

Daraus resultiert der folgende Assembler-Code:[1]

```
isPrime:   mov    %ecx, %edi ; %ecx = %edi (a)
           mov    %esi, $2   ; i = 2
           cmp    %esi, %ecx ; is i >= a?
           jge    prime      ; jump if yes
nexti:     mov    %eax, %ecx ; set %eax = a
           cdq               ; sign-extend
           idiv   %esi       ; a % i
           test   %edx, %edx ; is remainder zero?
           jz     notPrime   ; jump if yes
           inc    %esi       ; i++
           cmp    %esi, %ecx ; if i >= a?
           jl     nexti      ; jump if no
prime:     mov    %eax, $1   ; return value in %eax
           ret
notPrime:  xor    %eax, %eax ; %eax = 0
           ret
```

Der VMware Übersetzer sammelt eine Folge von maximal 12 Anweisungen oder einen kompletten Anweisungsblock, gekennzeichnet durch eine abschließende Kon-

[1] Der Übersetzter erwartet zwar binären Code, die Darstellung in Assembler ist aber wesentlich übersichtlicher und lässt die Arbeitsweise erkennen.

trollflussanweisung, falls dieser kürzer ist und bildet daraus eine *translation unit* (TU). So lautet dann auch die erste TU:

```
isPrime:   mov    %ecx, %edi
           mov    %esi, $2
           cmp    %esi, %ecx
           jge    prime
```

Die ersten drei Anweisungen lässt der Übersetzer unverändert passieren. Bei der Anweisung `jge` (***j**ump if **g**reater or **e**qual*) handelt es sich aber um einen bedingten Sprung, also eine Kontrollflussanweisung. Diese wird vom Übersetzer behandelt, indem er eine Fallunterscheidung einfügt. Da das das Code-Layout durch die Übersetzung verändert werden kann, wird der Sprung zur Marke `prime` durch zwei dynamisch berechnete Speicheradressen `[takenAddr]` und `[fallthrAddr]` ersetzt. Damit ändert sich die erste TU wie folgt:

```
isPrime':  mov    %ecx, %edi    ; IDENT
           mov    %esi, $2      ; IDENT
           cmp    %esi, %ecx    ; IDENT
           jge    [takenAddr]   ; JCC
           jmp    [fallthrAddr]
```

Die Übersetzung der übrigen TUs erfolgt analog. Bereits übersetzte Codeblöcke werden im *translation cache* (TC) abgelegt, und entsprechend der Cachestrategie verwaltet. Würde man nun etwa (in Anlehnung an [VM]) die Funktion `isPrime(49)` aufrufen, so stünde nach 6 Iterationen im TC folgender modifizierter Code:

```
isPrime':  mov    %ecx, %edi    ; IDENT
           mov    %esi, $2      ; IDENT
           cmp    %esi, %ecx    ; IDENT
           jge    [takenAddr]   ; JCC
                                ; fall-thru into next CCF
nexti':    mov    %eax, %ecx    ; IDENT
           cdq                  ; IDENT
           idiv   %esi          ; IDENT
           test   %edx, %edx    ; IDENT
           jz     notPrime'     ; JCC
                                ; fall-thru into next CCF
           inc    %esi          ; IDENT
```

```
            cmp    %esi, %ecx     ; IDENT
            jl     nexti'         ; JCC
            jmp    [fallthrAddr3];

notPrime': xor     %eax, %eax     ; IDENT
            pop    %r11           ; RET
            mov    %gs, 0xff39eb8(%rip), %rcx    ; spill %rcx
            movzx  %ecx, %r11b
            jmp    %gs:0xfc7dde0(8*%rcx)
```

3.2 Advanced BT

Moderne CPUs haben teure Traps, so dass ein BT-VMM einen klassische VMM in Performance überholen kann, indem privileged instruction traps vermieden werden. Wenn simple BT auch traps von privileged instructions umgehen kann, bleibt das Problem bestehen, dass non-privileged instructions (z. B. laden und speichern) auf sensitive Daten wie page tables zugreifen. Um das zu verbessern wird adaptive BT verwendet. Die Anweisungen des Gast-Systems werden solange 1:1 ausgeführt, bis die Hindernisse auffallen und dann wird die Übersetzung angepasst:

- Die Anweisung wird für neu übersetzt unter Vermeidung der traps.
- Die Originalanweisung wird von IDENT umgeleitet auf non-IDENT.

Anweisungen, die so adaptiert wurden, haben nun einen Mehraufwand um auf die Ersatzanweisung umgeleitet zu werden. Dieser Nachteil wird jedoch durch die bessere Code-Ausführung wieder aufgehoben.

4 Hardware-unterstützte Virtualisierung

Sowohl die Virtualisierung mittels binary translation als auch die Paravirtualisierung erzeugen einen Overhead, der zu einer messbaren Performanceeinbuße gegenüber der nativen Ausführung oder der idealen Virtualisierung führt. Sowohl AMD als auch Intel sind mit ihren neuen x86 Prozessoren angetreten, dieses Problem zu lösen. In Anschnitt 2.1.3 wurde das Konzept der der CPU-Ringe im Zusammenhang mit dem neu eingeführten Root-Mode vorgestellt. Im folgenden sollen die Details näher betrachtet werden.

Da sich die Architekturen von AMD und Intel weitgehend gleichen, wird nachfolgend soweit möglich nur das gemeinsames Konzept beschrieben, wichtige Unterschiede werden extra hervorgehoben.

4.1 Erweiterungen der x86 Architektur

Folgende Erweiterungen finden sich in der x86 Architektur:

- Zwei neue Ausführungslevel: Non-Root Mode und Root Mode
- Der VMCB Virtual Machine Control Block (AMD) oder VMCS Virtual Mchine Control Structure (Intel) - eine 4 kByte große Datenstruktur, die den Status einer virtuellem Maschine speichert.
- `vmrun` und `vmexit` zum Betreten und Verlassen einer virtuellen Maschine, sowie `vmxon` und `vmxoff` zum Starten und Stoppen des Hypervisors bzw. der Virtualisierungsumgebung

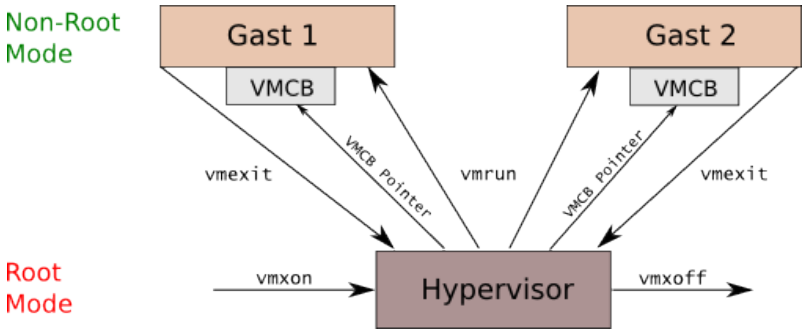

Abbildung 7: Ein Hypervisor und seine Gäste

4.2 Non-Root und Root Mode

Die Einführung eines neuen Ausführungslevel und die damit verbundenen Unterteilung in Non-Root Mode und Root Mode ist die eigentliche Neuerung der hardwareunterstützten Virtualisierung. Ähnlich wie bei der Paravirtualisierung läuft das Gast OS im Ring 0 und damit im Non-Root Mode. Alle Zugriffe des Gastes auf die Hardware werden von dem im Root Mode laufenden Hypervisor automatisch abgefangen.

Wechsel zwischen Non-Root Mode und Root Mode werden als VMX Transitions bezeichnet. Je nach Richtung handelt es sich um VM Entries oder VM Exits. (siehe Abschnitt 4.4)

Um die Hardwareunterstützung und damit den Root Mode nutzen zu können, muss diese zunächst aktiviert werden. Intel stellt dazu den Befehl `vmxon` bereit, bei AMD muss das **EFER.SVME** Bit gesetzt werden. Analog dazu lässt sich diese mit `vmxoff` bzw. das Löschen des entsprechenden Bit abschalten. Ohne die Hardwareunterstützung arbeitet der Prozessor im klassischen x86 Mode und Programmcode kann nativ ausgeführt werden.

4.3 Der VMCB/VMCS

Beim VMCB handelt es sich um eine 4 kByte große Datenstruktur, in der alle wichtigen Zustände einer virtuellen Maschine abgelegt sind. Für jede, auf einem Host laufende virtuelle Maschine, kann ein eigener VMCB angelegt werden. Der Zugriff auf einen VMCB erfolgt jeweils über einen 64-Bit Vektor den VMCB-Pointer.

Der VMCB ist in sechs Abschnitte untergliedert:

- **Guest-state area** Beim Verlassen einer virtuellem Maschine mit `vmexit` oder dem erneuten Betreten mit `vmrun` werden die Prozessorregister in diesen Abschnitt geschrieben bzw. ausgelesen. Im einzelnen sind dies:
  - Kontrollregister CR0, CR3 und CR4
  - Debug Register DR7
  - RSP (Stack Pointer), RIP (64-Bit Instruction Pointer) und RFLAGS (64-Bit Statustregister)
  - Die wesentlichen Felder[2] der folgenden Register: CS (Code-Sement), SS(Stack-Segment), DS(Daten-Segment), ES FS GS(Extra-Segment), LDTR(Local-Description-Table-Register) und TR(Task-Register)
  - Einige Felder[3] der Register GDTR (Global-Description-Table-Register) und IDTR (Interrupt-Description-Table-Register)
- **Host-state area** Wird eine virtuelle Maschine verlassen, werden die Prozessorregister von hier geladen.
- **VM-execution control fields** Diese Felder steuern das Verhalten des Prozessors, wenn sich dieser im non-root Mode befindet. Zum Teil werden hier Trigger für das Verlassen einer VM definiert.

[2] Selektor (16 Bit), Basis-Adresse (64 Bit), Segment-Grenze (32 Bit) und die Zugriffsrechte (32 Bit)

[3] Basisadresse (64 Bit) und Grenzadresse (32 Bit)

- **VM-exit control fields** Dieses Feld kontrolliert das Verlassen einer VM.
- **VM-entry control fields** Dieses Feld kontrolliert das Betreten einer VM.
- **VM-exit information fields** Aus diesen read-only Feldern können detailierte Informationen über den zuletzt ausgeführten VM-exit ausgelesen werden.

4.4 `vmrun` und `vmexit`

Die beiden Befehle `vmrun()` und `vmexit` realisieren schließlich die in 4.2 beschriebenen VMX-Transitions. `vmrun` realisiert den Übergang vom Root Mode in den Non-Root Mode und damit das Betreten einer bestimmten virtuellen Maschine. Als einziges Argument wird der entsprechende VMCB-Pointer übergeben.

Durch `vmrun()` wird eine Reihe von Anweisungen - die *basic operations* - ausgeführt, bevor am Ende die virtuelle Maschine läuft. Im einzelnen sind das folgende Operationen:

- **Sichern des Host-Status** in den VMCB
- **Laden des Gast-Status** aus dem VMCB
- **Lesen der Control Bits** aus dem VMCB
- **Lesen des Segment-Status** aus dem VMCB
- **Konsistenzprüfung zwischen Host- und Gast-Status** Zwischen den im VMCB abgelegten Daten von Host und Gast kann es unter Umständen zu Inkonsistenzen kommen. Wird eine solche gefunden, zieht das automatisch eine `vmexit`-Anweisung für die betreffende Maschine nach sich.

Ein `vmexit` kann durch verschiedenen Ereignisse ausgelöst werden und initiiert analog zu `vmrun` eine Reihe von Teiloperationen:

- **Sperren aller Interrupts** durch löschen des GIF (Global Interrupt Flag). Nur so ist sichergestellt, dass der Hypervisor den Statuswechsel vollständig ausführen kann.
- **Sichern des Gast-Status** in den VMCB
- **Sichern des EXITCODE** Damit wird das den Exit auslösende Ereignis im VMCB abgelegt. Abhängig vom Ereignis erlaubt AMD die Speicherung weiterer Informationen in den Feldern EXITINFO1 und EXITINFO2.

- **Laden des Host-Status** aus dem VMCB und prüfen auf Konsistenz.

Treten beim Wiederherstellen des Host-Status Fehler oder inkonsistente Daten auf, so hat dies automatisch einen *shutdown* des Prozessors zur Folge.

5 Vergleich der Technologien

In den vorangegangenen Kapiteln wurden die existierenden Virtualisierungstechnologien für die x86-Architektur ausführlich vorgestellt. In diesem Kapitel soll es nun um deren Vergleich gehen. Da alle drei Strategien im Wesentlichen den Kriterien von Popek und Goldberg genügen, stellt sich die Frage, welches dann die maßgeblichen Kriterien zur Bewertung und Entscheidungsfindung sind. Für die Autoren erscheinen die nachfolgend genannten Punkte wichtig, welche Techologie in einem bestimmten Umfeld eingesetzt werden soll.

- Leistungsfähigkeit im Vergleich mit nativer Ausführung
- Flexibilität bei Systemausfällen und Wartungsfenstern
- Kosten für Anschaffung und Unterhaltung

In Ermangelung anderer Testergebnisse werden wir den Schwerpunkt auf eine Gegenüberstellung der Produkte von VMware als Vertreter einer auf BT basierenden proprietären Lösung und XEN als Beispiel für Paravirtualization und hardwareunterstützte Virtualisierung legen.

5.1 Leistungsvergleich

Natürlich sollten virtuelle Maschinen ähnlich leistungsfähig sein, wie nativ arbeitende Systeme. Werden Virtualisierungslösungen im Bereich der Serverkonsolidierung eingesetzt, geht es aber auch darum, vorhandene Hardware besser - idealerweise zu 100% - auszulasten und so freigewordene Maschinen einzusparen, bzw. gar nicht erst anschaffen zu müssen. Gemessen an der Leistungsfähigkeit aktueller Maschinen bedeutet das aber auch, dass man einen, durch die Virtualisierung bedingten, geringen Performanceverlust im einstelligen Prozentbereich verschmerzen kann, wenn sich dadurch Leerlaufzeiten reduzieren lassen.

Die Anbieter von Virtualisierungslösungen werben gern mit den Leistungsdaten ihrer Produkte. Insbesondere VMware ist uns durch ein äußerst aggressives Marketing eher negativ aufgefallen. Die zahlreichen von den Herstellern durchgeführten Messungen basieren in der Regel auf dem direkten Vergleich eines nativen Systems

mit einem virtualisierten, das exklusiv auf einer Maschine läuft. Messungen, die im Kontext von beispielsweise 50 virtuellen Webservern auf einer physischen Maschinen durchgeführt wurden und damit ein realistisches Szenario nachbilden, konnten wir nicht finden.

Erschwerend kommt hinzu, dass sich die Testberichte von konkurrierenden Produkten zum Teil diametral gegenüberstehen, je nachdem, welcher Hersteller diese präsentiert. Außerdem veröffentlicht VMware den Quellcode seiner selbst kreierten Benchmarks nicht und verbietet außerdem in der EULA seiner Produkte die Veröffentlichung von jeglichen Testergebnissen, ohne Zustimmung. Dieser Schritt wird damit begründet, dass schlecht konstruierte Benchmarks das Image des Unternehmens beschädigen und bei den Kunden einen Vertrauensverlust hinterlassen könnten.

So bleibt uns nur, die zur Verfügung stehenden Ergebnisse soweit als möglich zu analysieren und ihre Entstehung kritisch zu hinterfragen.

5.1.1 Nanobenchmarks

Einer der wenigen Testalgorithmen, die uns im Quellcode vorliegen, ist der `forkwait` Nanobenchmark.

```
int main(int argc, char *argv[]) {
  for (int i = 0; i < 40000; i++) {
    int pid = fork();
    if (pid < 0) return -1;
    if (pid == 0) return 0;
    waitpid(pid);
  }
  return 0;
}
```

Dieses C-Programm erzeugt und vernichtet 40000 Prozesse. Damit sind bei einer virtuellen Maschinen die intensive Beanspruchung des Hypervisors sowie zahlreiche Kontextwechsel verbunden, was sich in den Ausführungszeiten niederschlägt.[4]

- nativ: 6.0 sec
- Software VMM (BT): 36.9 sec
- Hardware VMM: 106.4 sec

[4]Gemessen auf einem Intel Pentium 4, 3.6 GHz

Bei dem Hardware VMM handelt es sich um einen von VMware selbst entwickelten und nicht veröffentlichten Hypervisor, der die VT Erweiterung des Pentium 4 nutzt. So wundert es nicht, dass der Software VMM mit binary translation als klarer Favorit hervorgeht.

Die University of Cambridge präsentiert für XEN ebenfalls Nanobenchmarks, nutzt dafür aber die `lmbench` Benchmarksuite, welche im Quellcode vorliegt. Teil dieser Suite ist ein Test, der analog zu `forkwait` mittels `fork` Prozesse erzeugt, und die dafür benötigte Zeit ausgibt. In [Xen] findet man folgende Ergebnisse:

Config	fork proc
L-SMP	143
L-UP	110
Xen	198
VMW	874
UML	21000

Tabelle 2: `lmbench`: fork proc Benchmark - alle Zeiten in μs (L-SMP: Linux SMP Kernel, L-UP: Linux Uniprocessor Kernel, VMW: VMware Workstation, UML: User-Mode Linux

Diese Ergebnisse relativieren die des VMware `forkwait` Benchmarks. Ähnlich verhält es sich bei den anderen Nanobenchmarks `syscall, in, cr3wr, callret, pgfault, divzero` und `ptemod`. Allein bei `syscall` und `callret` gewinnt der Hardware VMM, bei den Anderen dominiert der Software VMM um Klassen (man beachte die logarithmische Achsenteilung).

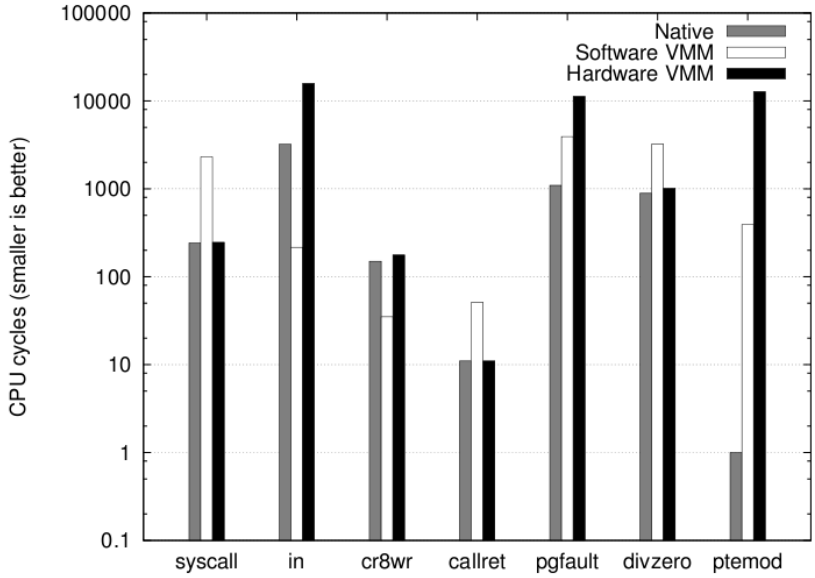

Abbildung 8: VMware Nanobenchmarks aus [VM]

Auch hier zeigt sich beim Betrachten der Xen-Benchmarks erneut ein gegensätzliches Bild:

Config	null call	null I/O	stat	open closeTCP	slct	sig inst	sig hndl	fork proc	exec proc	sh proc
L-SMP	0.53	0.81	2.10	3.51	23.2	0.83	2.94	143	601	4k2
L-UP	0.45	0.50	1.28	1.92	5.70	0.68	2.49	110	530	4k0
Xen	0.46	0.50	1.22	1.88	5.69	0.69	1.75	**198**	**768**	**4k8**
VMW	0.73	0.83	1.88	2.99	11.1	1.02	4.63	874	2k3	10k
UML	24.7	25.1	36.1	62.8	39.9	26.0	46.0	21k	33k	58k

Config	2p 0K	2p 16K	2p 64K	8p 16K	8p 64K	16p 16K	16p 64K
L-SMP	1.69	1.88	2.03	2.36	26.8	4.79	38.4
L-UP	0.77	0.91	1.06	1.03	24.3	3.61	37.6
Xen	**1.97**	**2.22**	**2.67**	**3.07**	**28.7**	**7.08**	39.4
VMW	18.1	17.6	21.3	22.4	51.6	41.7	72.2
UML	15.5	14.6	14.4	16.3	36.8	23.6	52.0

Abbildung 9: Xen Microbenchmarks aus [Xen] - alle Zeiten in μs

Wichtig ist nochmals festzuhalten, dass VMware in seinen Benchmarks seine Software VMM gegen eine hardwareunterstützte VMM antreten lässt, Xen dagegen präsentiert sein paravirtualisiertes OS im Vergleich mit der Software VMM von VMWare. In sofern sind die Ergebnisse nicht vollkommen gleichwertig und können damit nicht 1:1 verglichen werden.

5.1.2 Makrobenchmarks

Während die Nano- und Microbenchmarks einen Einblick in die Arbeit des Systems gewähren, lassen die Makrobenchmarks Rückschlüsse auf das Verhalten einer virtuellen Maschine im produktiven Einsatz zu. Zur Anwendung kommen in der Regel eine Reihe von Tests, die je nach Zusammenstellung verschiedene Anwendungsprofile nachbilden können. Im Gegensatz zur Leistungsmessung im realen Einsatz sind

diese jedoch standardisiert und ihre Ergebnisse lassen sich bei bekannten Rahmenbedingungen reproduzieren.

Sowohl VMware aus auch die University of Cambridge haben für ihre Virtualisierungslösungen Ergebnisse von Makrobenchmarks veröffentlicht, die im wesentlichen auf Testsuiten der SPEC.org[5] basieren. Ähnlich wie bei den Nanobenchmarks erscheinen auch hier die Produkte in einem anderen Licht, je nachdem wer die Messergebnisse veröffentlicht. Anfang 2007 veröffentlichte VMware eine Studie mit dem Titel *A Performance Comparison of Hypervisors* [VM2]. Darin werden VMware ESX Server 3.0.1, Xen 3.0.3 und ein natives System miteinander verglichen. Als Benchmarks kommen dabei SPECcpu2000 Integer, SPECjbb2005, Passmark und Netperf zum Einsatz. Im Ergebniss präsentiert die Studie folgende Ergebnisse.

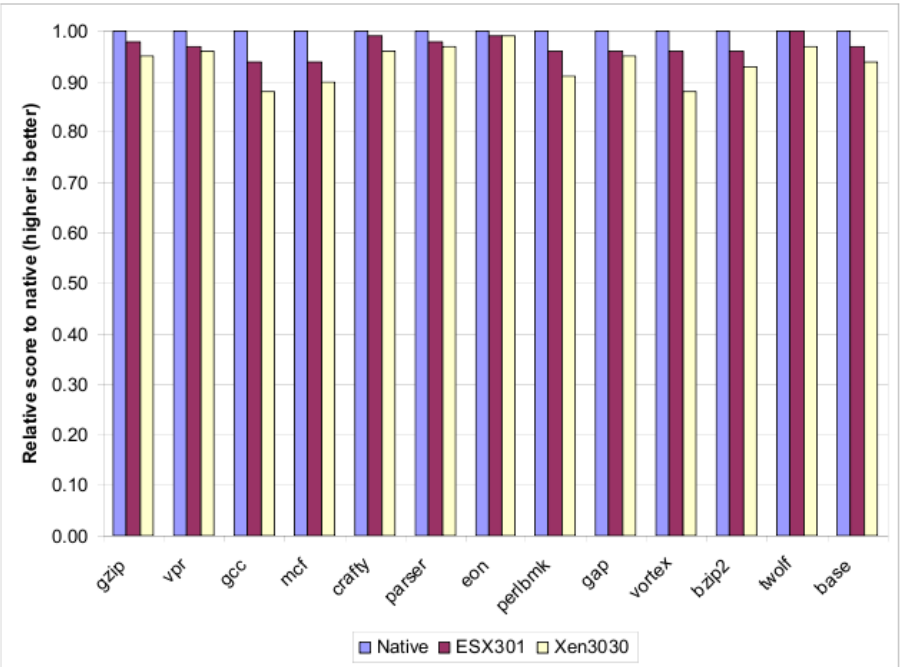

Abbildung 10: VMware: Benchmarkergebnisse der SPECcpu INT 200

Auf den ersten Blick sind zwar Unterschiede zwischen den drei getesteten System erkennbar, jedoch liegen alle Ergebnisse im oberen Leistungsbereich. Bei genauer Betrachtung ist aber auffallend, dass bei keinem der 13 Einzeltests das System mit Xen Hypervisor besser abschneidet als der VMware ESX Server 3.0.1. In der Studie werden neben anderen auch die beiden folgenden Diagramme veröffentlicht:

[5]Standard Performance Evaluation Corporation, `http://www.spec.org`

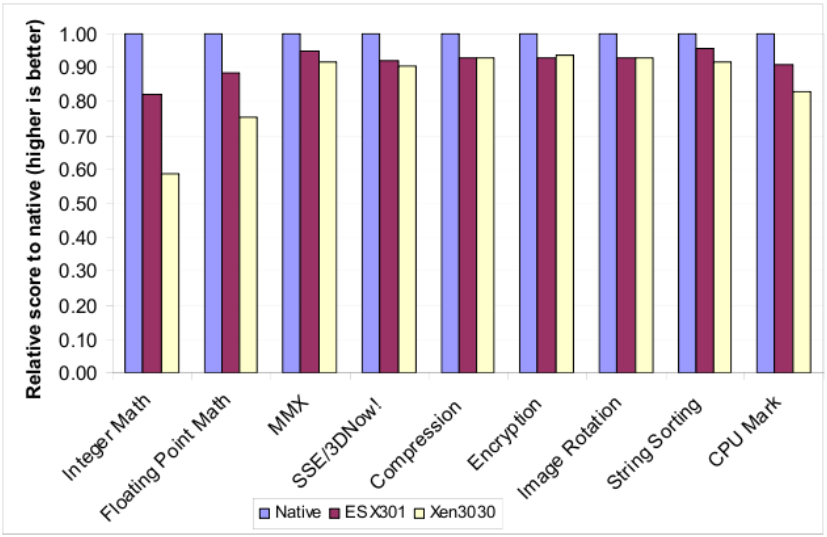

Abbildung 11: VMware: Benchmarkergebnisse mit Passmark

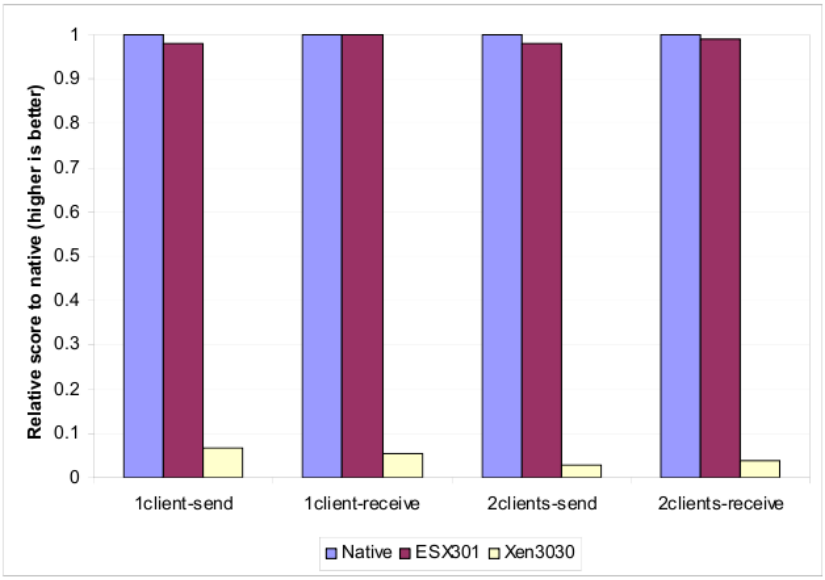

Abbildung 12: VMware: Benchmarkergebnisse mit netperf

VMware kommentiert das letzte Diagramm mit: [...] *VMware ESX Server can successfully virtualize network I/O intensive datacenter applications such as Web servers, fils servers and mail servers. The very poor network performance makes the Xen hypervisor less suitable for any such applications.*[...] [VM2] und identifiziert an andere Stelle auch die Ursache für das durchweg eklatante Versagen von Xen: [...] *The remaining performance gap is due to the 'stateless' nature of the hardware VMM.*[...] [VM] Weiterhin werden die in den Hardwareerweiterungen von Intel und

AMD implementierten `exits` beim Kontextwechsel als extrem teuer identifiziert und daher mit verantwortlich für das extrem schlechte Abschneiden in den Tests.

Als Antwort auf diese Veröffentlichung und das scheinbar völlige Versagen von Xen, veröffentlicht XenSource ebenfalls 2007 eine eigene Studie [Xen2], in der die Test unter nahezu identischen Bedingungen wiederholt werden. Die VMware EULA verbietet zwar das Veröffentlichen der mit ESX Server 3.0.1 gewonnenen Testergebnisse, was aber kein entscheidender Nachteil ist, da VMware die Messlatte für das eigene Produkt ja bereits festgelegt und öffentlich gemacht hatte. So findet man dann die folgenden drei Diagramme:

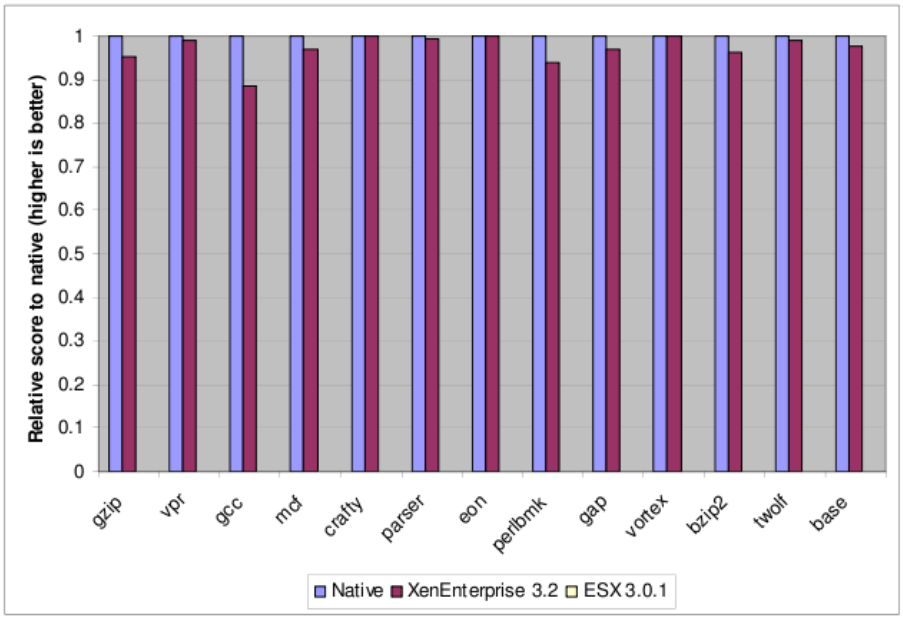

Abbildung 13: XenSource: Benchmarkergebnisse der SPECcpu INT 200

Bereits hier zeigen sich deutliche Unterschiede und deutliche Performancezugewinne verglichen mit der VMware Veröffentlichung. Bei den Einzeltests `crafty`, `eon` und `vortex` zeigt das mit Xen virtualisierte System sogar native Leistung.

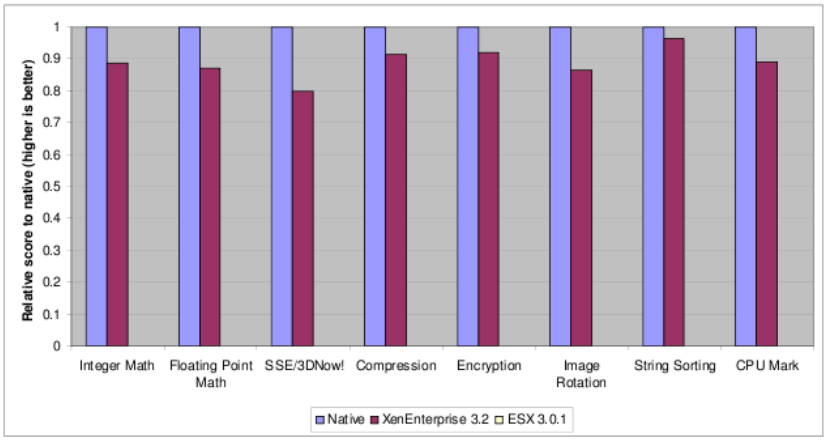

Abbildung 14: XenSource: Benchmarkergebnisse mit Passmark

Auch bei Passmark Benchmark schneidet Xen in den meisten Kategorien sichtbar besser ab. Die deutlichsten Differenzen zeigen sich aber beim `netperf` Benchmark:

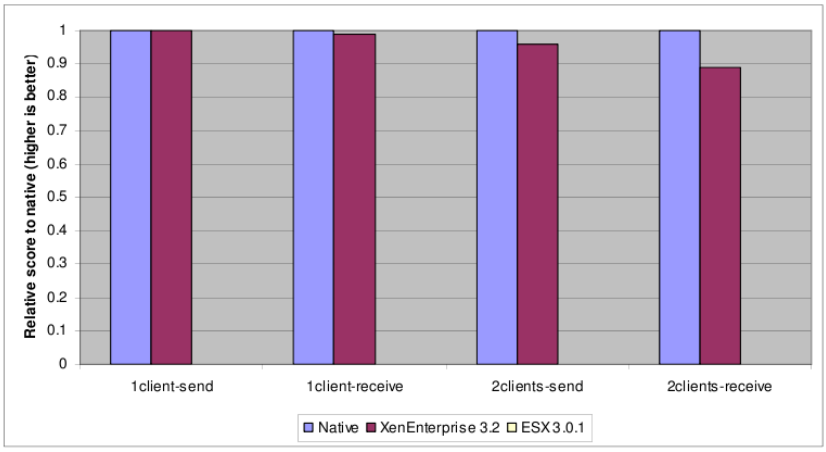

Abbildung 15: XenSource: Benchmarkergebnisse mit netperf

Der SPECjbb2005 testet einen Java Application Server und verursacht nur sehr wenig I/O. Dafür werden hierbei die Systeme auf Multithreading-Tauglichkeit getestet. Auch hierbei schneidet Xen mit fast nativer Performance ab. Bei VMware plakatiert man dagegen, dass es nicht möglich gewesen sei, eine SMP Windows auf dem XenHypervisor zu booten, obwohl Xen seit der Version 3.0 Intel-VTx und seit 3.0.2 AMD Vanderpol grundlegend unterstützt.

Nun stellt sich beim Leser die Frage, wie sich die schlechten, von VMware publizierten, Ergebnisse für Xen erklären lassen.

Xen bietet zwei Produktreihen parallel an. Zum einen Xen als Open-Sorce Projekt, momentan verfügbar in der Version 3.1 und zum anderen daraus abgeleitet

die XenServer Enterprise Produktreihe, bestehend aus Enterprise, Standard und Express Edition. VMware war 2007 mit seiner Studie verschiedene Hypervisor vergleichen. Erklärtes Ziel war: [...] *to validate their readiness for Enterprise Datacenters.*[...] [VM2] Man wollte also explizit Virtualisierungslösungen für den Hochlast- und Hochverfügbarkeitsbetrieb untersuchen. Dazu schreibt VMware weiter: [...] *A series of performance experiments was conducted on the latest shipping versions...* Aber genau das tat man bei VMware nicht. Anstatt das bereits damals verfügbare Produkt XenServer Enterprise zu testen, benutzte man die weniger optimierte Open-Source Version 3.0.3, die zudem nur eine teilweise Unterstützung für die Intel-VT Erweiterung besaß, was nach Angaben von Xensource auch den entsprechenden Releasnotes dokumentiert war. Über die Gründe für diesen unfairen Vergleich gibt es seitens VMware verständlicherweise keine Stellungnahme. Der Leser möge hier eigene Schlüsse ziehen.

Es bleibt also zunächst festzuhalten, dass sich auf Grund ihrer Leistungsdaten sowohl XenServer als auch VMware ESX Server trotz ihrer unterschiedlichen Technologien für den Enterprise-Einsatz gleichermaßen eignen.

5.2 Lizenzkosten

Sowohl XenServer Enterprise als auch VMWare Infrastructure Enterprise, worin auch der ESX Server enthalten ist, sind vollwertige Komplettlösungen für die Servervirtualisierung. Neben dem eigentlichen Hypervisor sind weitere Komponenten wie etwa VMMotion oder XenMotion, die den Umzug auf eine andere Maschine im laufenden Betrieb ermöglichen, enthalten. Bei VMware muss man beim Erwerb einer Infrastructure Enterprise Lizenz automatisch einen Supportvertrag mit erwerben, bei XenServer ist dies nicht der Fall. Für ein Rechenzentrum mit 20 Servern und einer angenommenen Laufzeit von 5 Jahren ergeben sich allein durch den Erwerb der Lizenzen folgende Kosten:

	XenServer Enterprise	VMware Infrastructure Enterprise
Lizenzkosten pro Server:	2.600 US$	9.416 US$
Support pro Server (Jahr 1-3)	400 US$	incl.
Support pro Server (Folgejahr)	400 US$	991 US$
Gesamtkosten 20 Server in 5 Jahren	92.000 US$	227.960 US$
Center Administrator Lizenz	Bestandteil von Enterprise Server	8.180 US$
Center Administrator Support ab 4. Jahr	-	862 US$
Gesamtkosten:	**92.000 US$**	**237.864 US$**

Tabelle 3: Anschaffungskosten für 20 Serverlizenzen und einen Nutzungszeitraum von 5 Jahren (Preise vom 11.02.2008, Quelle: `www.vmware.com` und `www.xensource.com`, Die Supportfolgekosten für VMware sind aus der Preisdifferenzen für eine 2 und 3 Jahreslizenz berechnet.)

Der Kostenunterschied ist also erheblich und kann somit die Wahl der Virtualisierungslösung entscheiden. Weiterhin ist aber die Hardware, soweit vorhanden, mit entscheidend. Die hardwareunterstützte Lösung mit XenServer setzt auch entsprechend ausgestattete 64-Bit Maschinen voraus, die die benötigten Prozessorerweiterungen bereit stellen, was aber auf alle Server ab Baujahr 2006 zutreffen dürfte. Ist dies nicht der Fall so bleibt entweder der Weg über die Paravirtualisierung oder der Einsatz von VMware ESX Server.

6 Zusammenfassung

Der Markt für Virtualisierungslösungen bietet ein enormes Potential und ist dementsprechend marketingtechnisch stark umkämpft. Die verfügbaren Produkte für den Enterprise-Einsatz sind gleichermaßen ausgereift, und bieten eine Performance, die der nativen Ausführung kaum nachsteht. Nachdem Virtualisierung nur Sinn macht, wenn Server zusammengefasst werden, die allein die vorhandenen Ressourcen der Hardware nicht ausnutzen, dürften minimale Leistungseinbußen hierbei keine Rolle spielen. Zusammen mit den grundsätzlichen Vorteilen der Virtualisierung - Einsparung von Energie, Kosten für Kühlsysteme und Platz, Verschieben von Gastsystemen zwecks Wartung oder Optimierung der Auslastung, schneller Aufbau von Testumgebungen mit Sicherungs- und Restoremöglichkeiten - erhält der Anwender heute einen echten Mehrwert. Für welches Produkt man sich im Einzelfall entscheiden wird, hängt nicht unwesentlich von den Rahmenbedingungen wie etwa der vorhandenen Hardware und dem Budget ab, ebenso wie von bereits vorhandenen Lizenzen. Ein Rechenzentrum, das bisher mehrheitlich auf Microsoft Windows und seine Serverderivate gesetzt hat und somit bereits im Besitz dieser Lizenzen ist, wird sich möglicherweise für VMware entscheiden, da hier die Integration recht einfach ist. Bei einem UNIX/Linux dominierten Datacenter wird die Entscheidung wahrscheinlich anders ausfallen.

So gibt es zum jetzigen Zeitpunkt keinen eindeutigen Gewinner. Möglicherweise wird zukünftig die hardwareunterstützte Virtualisierung eine zunehmend stärkere Rolle spielen, wenn ältere Rechnergenerationen durch neue ersetzt werden und somit die Einschränkungen der x86 Architektur entfallen und damit die Notwendigkeit für BT und Paravirtualization.

Literatur

[AMD05] *AMD64 Architecture Programmer's Manual Volume 2: System Programming*,
URL:www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/24593.pdf
Download vom 17.02.2008

[Int05] *Intel Virtualization Technology Specification for the IA-32 Intel Architecture*,
URL:www.cs.utah.edu/classes/cs7940-010-rajeev/spr06/papers/vm.pdf
Download vom 17.02.2008

[Klei08] Jan Kleinert: *Virtualisierung 2008*, Linux Magazin 02/2008, S.31

[Spr08] Henning Sprang, Jürgen Quade: *Engagierte Spielmacher*, Linux Magazin 02/2008 S.33

[VM] Keith Adams, Ole Agesen: *A Comparison of Software and Hardware Techniques for x86 Virtualization*,
URL:www.vmware.com/pdf/asplos235_adams.pdf
Download vom 09.02.2008

[VM2] *A Comparison of Hypervisors*,
URL:http://www.vmware.com/pdf/hypervisor_performance.pdf
Download vom 11.02.2008

[VM3] VMware *Understanding Full Virtualization, Paravirtualization and Hardware Assist*,
URL:http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf
Download vom 03.12.2007

[Xen] *Xen and the art of Virtualization*,
URL:www.cl.cam.ac.uk/research/srg/netos/papers/2003-xensosp.pdf
Download vom 09.02.2008

[Xen2] *A Performance Comparison of Commercial Hypervisors**,
URL:http://blogs.xensource.com/simon/wpcontent/uploads/2007/03/hypervisor_performance_comparison_1_0_3_no_esx-data.pdf
Download vom 11.02.2008

Die Unterzeichnenden versichern, dass diese Seminararbeit selbstständig verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen verwandt wurden. Die Stellen, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen wurden, sind in jedem einzelnen Fall unter Angabe der Quelle als Entlehnung (Zitat) kenntlich gemacht. Das gleiche gilt für beigefügte Skizzen und Darstellungen. Außerdem räume ich dem Lehrgebiet das Recht ein, die Arbeit für eigene Lehr- und Forschungstätigkeiten auszuwerten und unter Angabe des Autors geeignet zu publizieren.

Hagen, den (18.02.2008)

Bernhard Biendl

Andreas Lemke