

TECHNISCHE UNIVERSITÄT DRESDEN

FAKULTÄT INFORMATIK
INSTITUT FÜR TECHNISCHE INFORMATIK
PROFESSUR FÜR RECHNERARCHITEKTUR
PROF. DR. WOLFGANG E. NAGEL

Hauptseminar
“Rechnerarchitektur und Programmierung”

Vergleich der virtuellen Maschinensoftware Xen und
KVM

Sven Jansky
(Mat.-Nr.: 3385705)

Betreuer: Thomas William

Dresden, 28. September 2011

Inhaltsverzeichnis

1	Gründe für eine Virtualisierung	4
1.1	Tests	4
1.2	Migration	4
1.3	Sicherheit und Kosten	4
2	Hardware-Virtualisierung	5
2.1	Ringmodell	5
2.1.1	32 Bit	5
2.1.2	64 Bit	5
2.2	Intel VT-x und AMD-V	6
2.2.1	Sensitive Befehle	6
2.2.2	Privilegierte Befehle	6
2.2.3	Popek-Goldberg-Theorem	6
3	Virtualisierungstechniken	7
3.1	Hardware Virtual Machine (HVM)	7
3.2	Paravirtualisierung	7
3.2.1	32 Bit	8
3.2.2	64 Bit	8
4	Hypervisor	9
4.1	Hypercall	9
4.2	BIOS	9
5	Beispiele für Virtualisierungen	10
5.1	Xen	10
5.2	KVM	11
6	Test-Infrastruktur Tapper	12
6.1	Motivation	12
6.2	TAP	12
6.2.1	Standard TAP	12
6.2.2	Tapper Erweiterungen	14
6.3	Tapper Schichten	14
6.3.1	Automatisierungssystem	14
6.3.2	Test Suite	14
6.3.3	Report Framework	14
6.4	Vorraussetzungen	15
7	Messungen	16
7.1	Messumgebung	16
7.2	Matrixmultiplikation	17

7.3	OpenMP(4)-Matrixmultiplikation	18
7.4	OpenMP(6)-Matrixmultiplikation	19
7.5	OpenMP(8)-Matrixmultiplikation	20
7.6	Memory Bandbreite	21
7.7	MPI-Communication	22
7.8	Hdparm -t	23
7.9	Lesen ohne RAM	24
8	Zusammenfassung	25
	Literatur	26

Zusammenfassung

In diesem Dokument werden die beiden Virtualisierungstechniken KVM und Xen miteinander verglichen.

Dazu werden zuerst die Gründe, warum man virtualisieren sollte beleuchtet.

Anschließend die Forderungen an die Hard- und Software, insbesondere den Prozessor.

Die eigentliche Virtualisierung wird durch den sogenannten Hypervisor realisiert. Aus diesem Grund wird dieser in einem eigenen Kapitel besprochen.

Nachdem die wichtigsten Grundlagen für Virtualisierungen besprochen worden, werden die beiden Virtualisierungen Xen und KVM in Bezug und ihre Besonderheiten angesprochen.

Eine Leistungsuntersuchung durch Messungen erfolgt, nach der Einführung des Automatisierungsframework Tapper, welches zum vereinfachten Installieren und Messen virtueller Maschinen genutzt werden kann.

1 Gründe für eine Virtualisierung

Zuerst einmal scheint eine Virtualisierung, der damit verbundenen Overhead und der zusätzlichen Verwaltungsaufwand wenig sinnvoll. Doch gibt es mehrere Gründe derartige Techniken zu verwenden.

1.1 Tests

Zum Beispiel für die Entwicklung neuer Kernel und deren Tests können sehr gut virtuelle Maschinen eingesetzt werden, sodass jeder Test auf einem bereits laufenden Rechner stattfinden kann. Ein häufiges Neustarten des realen Systems wird damit unnötig.

Weiterhin kann für jeden Test eines Programms eine neue virtuelle Umgebung erstellt und gestartet werden. Damit werden eventuelle Nebeneffekte von vorherigen Versionen oder anderen Programmen minimiert.

1.2 Migration

Die virtuellen Maschinen können ohne große Probleme auf neuere, leistungsfähigere Systeme migriert werden. Dies verhindert Ausfälle und Neuinstallationen durch den Umzug auf neue Hardware.

1.3 Sicherheit und Kosten

Es besteht immer ein Sicherheitsrisiko, wenn mehrere Serverdienste auf einem System laufen. Wenn ein Serverdienst eine Sicherheitslücke hat sind auch alle anderen Serverdienste in Gefahr.

Dies kann man verhindern, indem jeder Serverdienst ein eigenes System erhält, auf diesem ausschließlich dieser Dienst läuft.

Ein nicht unerheblicher Kostenfaktor, will man für jedes System eigene Hardware kaufen und betreiben. Mit virtuellen Rechnern hingegen kann man die vorhandene Hardware für mehrere Systeme verwenden und diese trotzdem logisch voneinander trennen.

Doch nicht nur Sicherheitskonzepte lassen sich durch eine Konsolidierung (Zusammenfassung) von logisch getrennten Rechnern auf die selbe Hardware verbessern.

Man kann damit die Auslastung der Hardware verbessern und durch die besser ausgelasteten Ressourcen die Anzahl der nötigen Ressourcen reduzieren. Dies führt zu Einsparungen der laufenden Kosten, wie zum Beispiel Strom, Kühlsysteme und Reperaturen.

2 Hardware-Virtualisierung

Jeder virtuelle Maschine braucht eine CPU und Zugriff auf die Hardware. Dabei muss jedoch beachtet werden, dass sich die verschiedenen virtuellen Maschinen nicht gegenseitig beeinflussen können, sondern jeweils eine komplett eigenständige Umgebung verfügbar ist.

2.1 Ringmodell

Prozessoren besitzen verschiedene Sicherheitsstufen, die man sich als Ringe vorstellen kann. Je kleiner der Ring, desto privilegiierter ist er.

2.1.1 32 Bit

Die 32 Bit Architektur setzt auf vier Ringe, wobei davon nur zwei Ringe benutzt werden. Ring 0 ist der Kernel-Space, in diesem können alle Befehle uneingeschränkt ausgeführt werden. Hier läuft im Normalfall der Kern des Betriebssystems. Die Ringe 1 und 2 sind unbenutzt. Im vierten Ring 3 laufen alle Programme der Nutzer, dem sogenannten User Space.

2.1.2 64 Bit

AMD, die treibende Kraft bei der Einführung des 64 Bit Standards, hat auf die zwei ungenutzten Ringe 1 und 2 aus der 32 Bit Architektur verzichtet. Somit gibt es nur noch die beiden Ringe 0 (Kernel-Space) und 1 (User-Space).

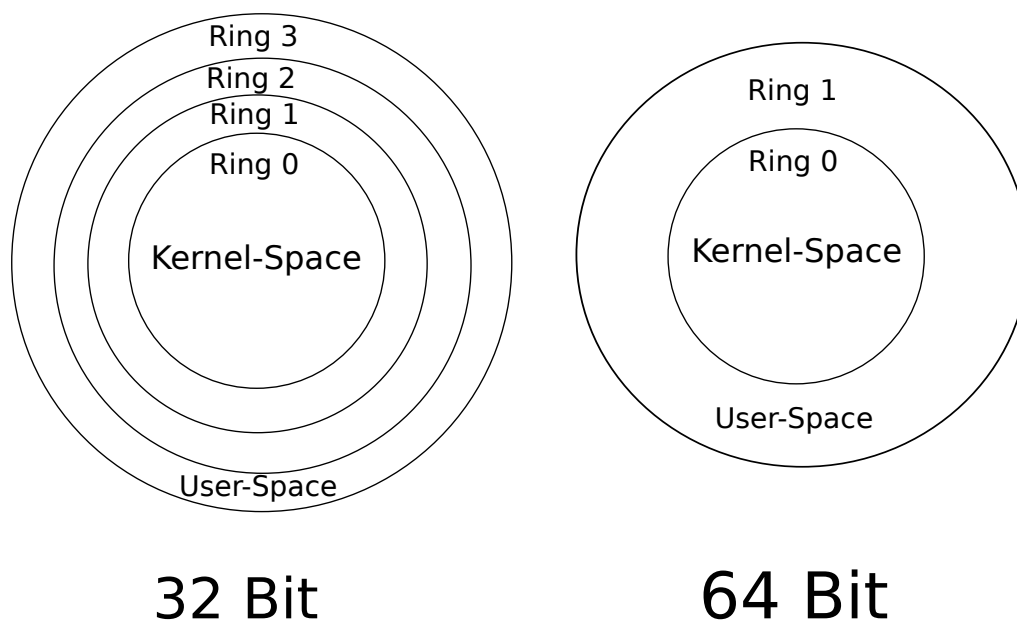


Abbildung 1: Ringmodell der x86 und AMD64 Architektur

2.2 Intel VT-x und AMD-V

Um die Unzulänglichkeiten in der x86 Architektur, in Bezug auf die Virtualisierbarkeit, auszugleichen haben die beiden Hersteller AMD und Intel neue Virtualisierungslösungen in ihre Prozessoren eingebaut.

Damit kann der notwendige Software-Overhead im Hypervisors minimiert werden. Dies wird durch neue Befehle der Prozessoren realisiert.

Diese Veränderung in der Hardware ist für eine effiziente Virtualisierung notwendig, da die x86-Architektur laut dem Popek-Goldberg-Theorem nicht effektiv virtualisierbar ist. [Fis09][VMW]

2.2.1 Sensitive Befehle

Sensitive Befehle sind Befehle, die zu einer Zustandsänderung einer Resource führen, oder solche die einen bestimmten Zustand einer Resource voraussetzen.

2.2.2 Privilegierte Befehle

Privilegierte Befehle sind alle Befehle, die im Kernel-Mode ohne weiteres ausgeführt werden können und im User-Mode zu einer Hardware-Trap führen.

2.2.3 Popek-Goldberg-Theorem

Das von Popek und Goldberg im Jahr 1974 veröffentlichte Theorem beschreibt die Anforderungen an die Hard- und Software für eine effektive Virtualisierung.

Laut diesem Theorem müssen die sensitiven Befehle, welche den Zustand der Maschine ändern können eine Untermenge der privilegierten Befehle sein.[Pop]

3 Virtualisierungstechniken

Man unterscheidet grundsätzlich zwei Virtualisierungen.

3.1 Hardware Virtual Machine (HVM)

Erstens die Hardware Virtual Maschine, oder hardwarebasierte Virtualisierung.

Dabei wird auf die Virtualisierungsbefehle des Prozessors zurückgegriffen. Damit ist es nicht notwendig das Host- und Gastsystem anzupassen. Dies erlaubt die Installation von Virtuellen Maschinen auch auf geschlossenen Betriebssystemen, wie zum Beispiel Windows. Für diese Art der Virtualisierung muss die verwendete CPU allerdings Virtualisierungsbefehle unterstützen. Erkennbar hat den CPU-Flags “vmx” für Intel und “svm” für AMD-Prozessoren.

3.2 Paravirtualisierung

Bei einer Paravirtualisierung greifen alle virtuellen Maschinen durch den Hypervisor gesteuert auf die Hardware zu. Für diese Virtualisierungart muss allerdings der Kernel des Host-Betriebssystems angepasst werden und damit werden die Quellen des Kernels benötigt.

Aus diesem Grund wird die Paravirtualisierung hauptsächlich auf Open Source-Systemen, wie zum Beispiel Linux, verwendet. Der Hypervisor wird “zwischen” die Hardware und den Kernel des Hostsystems geschaltet. (Siehe Abbildung 2) Bei diesem direktem “durchschalten” der Hardware an die jeweilige vir-

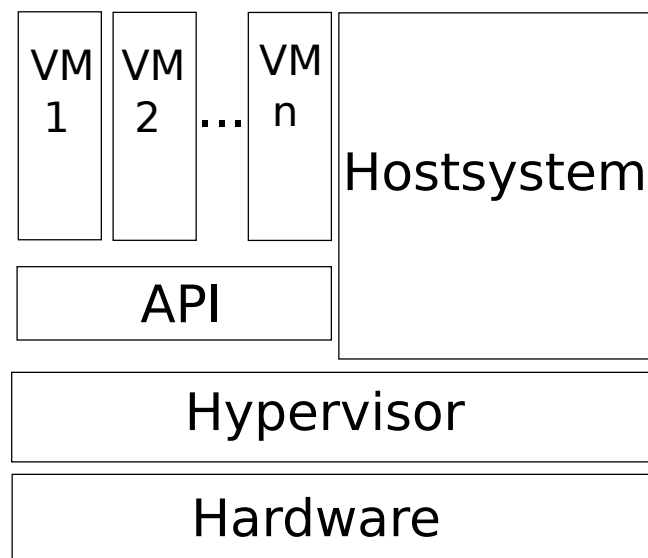


Abbildung 2: Paravirtualisierung

tuelle Maschine rechnet man nur mit Leistungseinbußen von 2-3 Prozent. [Fis09] Jedoch muss man bei der Paravirtualisierung zwischen 32Bit und 64Bit unterscheiden.

3.2.1 32 Bit

Da es bei einem 32 Bit Prozessor 4 Ringe gibt und damit Ring 1 und 2 ungenutzt ist kann der Kern des Systems in Ring 1 “gehoben” werden, damit der Hypervisor in Ring 0 privilegierter und vom eigentlichen Kern des Gastsystems getrent arbeiten kann.

Bei dieser Lösung gibt es allerdings Probleme mit Befehlen des Kerns, die Ring 0 erfordern.

Um dies zu beheben wird der Kern des Systems angepasst, sodass dieser auch im Ring 1 laufen kann.

Die eigentlich für Ring 0 privilegierten Befehle des Kerns müssen an den Hypervisor übergeben werden, da nur dieser im Ring 0 arbeitet und somit die Ausführung des Befehls ermöglicht.

3.2.2 64 Bit

Wie bereits erwähnt gibt es in einer 64 Bit Architektur nur die Ringe 0 und 1.

Eine Tatsache, die neue Anforderungen an die Virtualisierung stellt.

Es können nicht mehr, wie in einer 32 Bit Architektur, die verschiedenen Ringe genutzt werden.

Das bedeutet, dass der Hypervisor zusammen mit dem Kernel des Host-Betriebssystems im Ring 0 laufen muss. Um diesem Problem zu begnennen wurde die Segementierung innerhalb eines Ringes eingeführt.

Dies trennt den Hypervisor und den Host-Kern logisch voneinander, obwohl sie im selben Ring laufen.

[Fis09]

4 Hypervisor

Der Hypervisor ist das eigentliche Herzstück und seiner Wortherkunft nach der “Aufseher” einer Virtualisierung. Er verwaltet und steuert den Zugriff auf die Hardware und hat damit als Einziger volle Rechte auf die Hardware, dafür ist eine eigene Speicher- und Treiberverwaltung implementiert.

Damit der Hypervisor die vollen und höchsten Rechte auf dem System hat muss dieser in Ring 0 laufen und alle Informationen über das System haben. Diese vollständigen Informationen hat ausschließlich der Hypervisor.

Die Gastssysteme brauchen nicht zu wissen, wieviele Systeme noch laufen, wer gerade Zugriff auf die Hardware hat oder wer Zugriff haben will.

4.1 Hypercall

Als Hypercall versteht man Befehle, die vom Gast- oder Hostsystemen in weniger privilegierten Ringen ausgeführt werden, allerdings nach dem Sicherheitskonzept der Ringe in diesem Ring nicht erlaubt wären. Wenn ein Gastsystem einen privilegierten oder sensitiven Befehl ausführen will muss dieser durch einen Hypercall ersetzt werden. Man kann die Hypercalls mit den System-Calls unter Unix vergleichen. Damit werden die in der Architektur fehlenden Hardware-Traps durch Software-Traps ersetzt um die Trennung der virtuellen Maschinen voneinander zu gewährleisten. Das Hostsystem muss die Hypercalls unterstützen. Für die Gastssysteme gibt es zwei Möglichkeiten.

Entweder das Gastsystem bekommt auch einen angepassten Kernel, wodurch direkt im Kernel die entsprechenden Befehle durch die Hypercalls ersetzt werden.

Eine weitere Möglichkeit ist, dass entsprechende Treiber verwendet werden, wodurch das Gastsystem nicht angepasste Befehle ausführt. Die eigentliche Umsetzung in einen Hypercall macht in diesem Fall erst der Treiber.

Diesen Umweg muss man bei Systemen gehen, die nicht Quelloffen sind, wie zum Beispiel Windows. In jedem Fall bedeutet ein Hypercall immer die Nutzung eines angepassten Codes, damit der Hypervisor die Kontrolle hat und nicht die Gastssysteme den direkten Hardwarezugriff haben.

4.2 BIOS

Damit ein Rechner booten kann benötigt er Zugriff auf das BIOS. In Bezug auf Virtualisierung ist dies ein Problem, denn die virtuellen Maschinen sollen keinen direkten Zugriff auf die Hardware haben. Indem der Hypervisor ein BIOS simuliert und die virtuellen Maschinen ihre Informationen aus diesem simulierten BIOS beziehen, kann die Trennung von der Hardware gewährleistet werden.

Es müssen drei Teile simuliert werden.

Als erstes wird eine Start-Info-Page benötigt. Dabei werden essentielle Informationen für den Gast bereitgestellt. Ohne die keine Initialisierung des Kernels möglich wäre.

Weiterhin wird die Sharing Info Page bereitgestellt. Über diese werden Live-Informationen für das System verfügbar gemacht. Diese werden auch noch aktualisiert, wenn der Gast bereits läuft.

Als dritten und letzten Teil gibt es XenStore. Mit Hilfe dieser Komponente kann der Gast ermitteln, welche virtuelle Hardware nutzbar ist. [Fis09]

5 Beispiele für Virtualisierungen

Eine Virtualisierungssoftware besteht aus zwei Teilen.

Zum einen aus dem eben besprochenen Hypervisor, mit all seinen Aufgaben und einer Reihe von Verwaltungssoftware. Diese Softwaretool dienen zum Beispiel dem Erstellen, Löschen, Starten und Beenden der virtuellen Maschinen.

Eine weitere wichtige Funktion ist die Migration der virtuellen Maschinen.

Dabei können bereits installierte und laufende virtuelle Maschinen von einem Hostsystem auf einem neuen Host übertragen werden. Dies kann mitunter sogar im laufenden Betrieb passieren, das heißt auch während der Migration ist die virtuelle Maschine die ganze Zeit erreichbar.

Es gibt eine ganze Reihe von Virtualisierungslösungen.

Ein paar dieser sind zum Beispiel:

- Xen
- KVM
- VMware
- VMware Server
- VirtualBox
- QEMU

5.1 Xen

Die Virtualisierungslösung Xen wurde von der Universität Cambridge ins Leben gerufen. Die Entwicklung begann 2001, eine erste Veröffentlichung zu Xen wurde Ende 2003 herausgegeben. Anfangs war Xen ein Teil des XenServer-Projektes. Jedoch wurde aufgrund des großen Erfolges ein eigenes Projekt der Firma XenSource daraus. Später wurde XenSource von Citrix übernommen.

Citrix bietet kommerzielle Varianten von Xen inklusive dazugehörigen Support an. Xen ist komplett OpenSource und, sofern man auf den kommerziellen Support verzichtet, kostenlos. Es handelt sich um eine Paravirtualisierer, das heißt Xen benötigt immer einen eigenen Kernel auf dem Hostsystem. Allerdings werden in aktuellen Varianten auch die Möglichkeiten der speziellen CPU-Befehlssätze im Zuge der Hardware Virtual Maschine (HVM) genutzt.

Nicht einmal das Hostsystem “sieht” die virtuellen Maschinen. Diese werden nicht als eigener Prozess oder Ähnliches angezeigt. Der Arbeitsspeicher wird fest einer Maschine zugeordnet, dabei zeigt nach dem Starten einer virtuellen Maschine das Hostsystem weniger maximal verfügbaren Arbeitsspeicher an, da dieser an die gestartete virtuelle Maschine vergeben wurde und damit fest für diese Maschine reserviert ist.[Fis09]

5.2 KVM

Die Kernel based Virtual Maschine wird von Qumranet, welches mittlerweile von Red Hat aufgekauft wurde, entwickelt. Die ersten Versionen von KVM waren später als Xen verfügbar. KVM wurde 2006 veröffentlicht.

KVM selbst besteht nur aus zwei Modulen. Dem `kvm.ko` und je nachdem welcher Prozessor verwendet wird `kvm_intel.ko` oder `kvm_amd.ko`.

Diese Unterscheidung muss gemacht werden, da sich INTEL-VT und AMD-V zu stark voneinander unterscheiden und ein Modul für beide Prozessoren unnötig viel Funktionalität beinhaltet.

KVM wurde als Hardwarevirtualisier konzipiert und entwickelt. Erst später wurden Teile einer Paravirtualisierung hinzugefügt. Bei dieser Virtualisierungslösung gibt es keine direkte Verwaltungssoftware!

Um die mit KVM virtualisierten Maschinen verwalten zu können nutzt man eine modifizierte QEMU (Quick Emulation) Verwaltungs-Software.

Im Gegensatz zu Xen ist bei KVM jede virtuelle Maschine auf dem Host sichtbar.

Diese werden als eigener Prozess dargestellt. Dieser Prozess verbraucht dann den Arbeitsspeicher auf dem Hostsystem.

Demnach ist auch mit gestarteten virtuellen Maschinen der volle Arbeitsspeicher auf dem Hostsystem sichtbar und jede Maschine verbraucht nur soviel Arbeitsspeicher, wie sie aktuell wirklich benötigt. Der Arbeitsspeicher wird nicht auf Reserve einer virtuellen Maschine zugeordnet.[KVM]

6 Test-Infrastruktur Tapper

Tapper ist eine Sammlung von in Perl geschriebenen Tools, mit denen sich ganze Betriebssysteme testen lassen. Sie wurde von AMD als OpenSource freigegeben.

Diese Infrastruktur besteht auf verschiedenen Tools, Protokollen und Anwendungen für das Testen. Dabei wird das Test Anything Protokoll (TAP) verwendet.

Tapper bietet direkt eine Xen-Unterstützung, wodurch das Testen von virtuellen Maschinen mithilfe von Tapper sehr einfach ist.

6.1 Motivation

Mithilfe dieser Test-Infrastruktur kann automatisch getestet werden. Durch eine derartige Automatisierung von vordefinierten Tests hat man eine Reihe von Vorteilen.

Man kann für jeden Test eine eigene Umgebung schaffen und immer nur genau einen Test in dieser Umgebung machen, damit werden eventuelle Nebeneffekte, von vorherrigen Tests vermieden.

Bei Anwendungsfällen die mehrere identische Testdurchläufe nötig machen werden zu installierendes Paket oder Test nicht vergessen. Was, bei entsprechenden Anwendungsfällen, eine deutliche Zeitersparnis mit sich bringt.

6.2 TAP

Das Test Anything Protokoll ist für viele verschiedene Sprachen verfügbar, wie zum Beispiel:

- C++
- Perl
- Ruby
- Java

Ursprünglich entwickelt wurde TAP allerdings ausschließlich für Perl.

Es handelt sich um ein einfaches textbasiertes Interface.

6.2.1 Standard TAP

```
1 1..3
2 ok 1 – erster Test
3 ok 2 – zweiter Test
4 not ok 3 – dritter Test
```

Eine Ausgabe mithilfe von TAP.

In der ersten Zeile muss angegeben werden, wieviele Test durchgeführt wurden. Sollte diese Angabe nicht mit den den tatsächlich durchgeführten Tests übereinstimmen meldet der Parser einen Fehler.

```
1 1..3
2 ok 1 – erster Test
3 not ok 3 – dritter Test
```

In so einem Fall fehlt ein Test, was durch den Parser erkannt und angemerkt wird. Dies kann passieren, wenn ein Test nicht zum Ende kam und damit kein Ergebnis vorhanden ist.

Jede weitere Ausgabezeile, abgesehen von der ersten, besteht aus einer Angabe, ob der Test Erfolgreich “ok” oder nicht erfolgreich “not ok” war.

Alle weiteren Informationen sind optional.

Die Angabe der Testnummer erleichtert die Suche nach eventuell nicht durchgeführten oder beendeten Tests, obwohl dieser geplant war. Getrennt durch ein Bindestrich ist es möglich eine Beschreibung zu diesem Test anzugeben.

Die bisherrigen Möglichkeiten dienen rein der Ausgabe und ermöglichen keine Angaben, was weiterhin mit diesem Test passieren soll.

```
1 1..3
2 ok 1 – erster Test
3 ok 2 – zweiter Test #TODO unerwartet positiv
4 not ok 3 – dritter Test #TODO
5 # Failed test 'dritter Test'
6 # at t/data_dpath.t line 410.
7 # got: 'foo'
8 # expected: 'bar'
```

Tests können als TODO oder SKIP markiert werden. TODO bedeutet, dass ein anderes Ergebnis erwartet wurde, als getestet wurde. Im Falle von positiven Test bedeutet dies, dass ein Fehlschlagen des Testes erwartet wurde.

SKIP bedeutet, dass dieser Test nicht wirklich gelaufen ist, sondern einfach auf “ok” gesetzt wurde. Für TODO können und für SKIP müssen Gründe angegeben werden.

Durch das Kommentarzeichen # ist es möglich weitere Informationen zum Fehlschlagen eines Test zu geben. Im obigen Beispiel kam ein anderer Wert vom Test zurück als erwartet wurde.

6.2.2 Tapper Erweiterungen

Die Tapper Test-Suite hat das TAP-Protokoll um Features erweitert, diese sind nicht Bestandteil des TAP-Standards.

```
1 # Tapper-cpuinfo          —— CPU
2 # Tapper-uname           —— kernel information
3 # Tapper-osname          —— OS information
4 1..2
5 # Tapper-section: arithmetics
6 ok 1 – add
7 ok 2 – multipliyiel
8 1..1
9 # Tapper-section: string handling
10 ok 1 – concat
11 1..2
12 # Tapper-section: benchmarks
13 ok 1
14 ok 2
```

Durch die Tapper-Erweiterungen ist es möglich mehrere Sektionen in einer Testdatei zusammenzufassen. Die Tests werden durch die eigentlich erste Teile in jeder Datei (1..n) getrennt. Weiterhin können im Header Meta-Daten, wie zum Beispiel Informationen über die CPU, Ram, und weiteres mitgeliefert werden.[AMDb]

6.3 Tapper Schichten

Tapper besteht aus drei Schichten, welche alle voneinander unabhängig arbeiten.

6.3.1 Automatisierungssystem

Tapper bietet ein System zum automatischen Erstellen, Installieren der Testumgebung und Testen von Maschinen.

Dabei können auch direkt virtuelle Maschinen erstellt werden.

6.3.2 Test Suite

Die eigentlichen Tests werden von der Test Suite durchgeführt.

Dabei werden Reports mithilfe von TAP erstellt und können an das Report Framework weitergereicht werden.

6.3.3 Report Framework

Das Report Framework ist zum Empfangen und Auswerten der Daten.

Es kann vollkommen eigenständig verschiedenste Testdaten auswerten, wichtig ist nur, dass die Daten mithilfe von TAP übermittelt werden.[AMDa]

6.4 Voraussetzungen

Für die Tests können eine Reihe von Voraussetzungen angegeben werden. Es gibt zwei mögliche Voraussetzungen: Normal und Makro.

Zum Beispiel:

- zu installierende Pakete
- zu kopierende Pakete
- Bootmanager konfigurieren
- Befehle ausführen
- Rechner neu starten

Damit kann sichergestellt werden, damit die Testumgebung wirklich stimmt.

Wenn eine Voraussetzung für einen Test nicht erfüllt wird, wird der entsprechende Test nicht durchgeführt.

7 Messungen

7.1 Messumgebung

Für die Test der virtuellen Maschinen wurde folgendes Testsystem verwendet:

- Intel Core2 Quad Q6700
 - 4 Kerne
 - 32kB I-L1-Cache
 - 32kB D-L1-Cache
 - Jeweils 2 Kerne teilen sich einen 4 MB L2-Cache
- 8 GB RAM
- Software
 - Gcc version 4.4.5 (Debian 4.4.5-8)
 - Open MPI 1.4.2
 - Xen 4.0
- Dienste
 - Openssh-server

Für die virtuellen Maschinen wurde folgende Konfiguration verwendet:

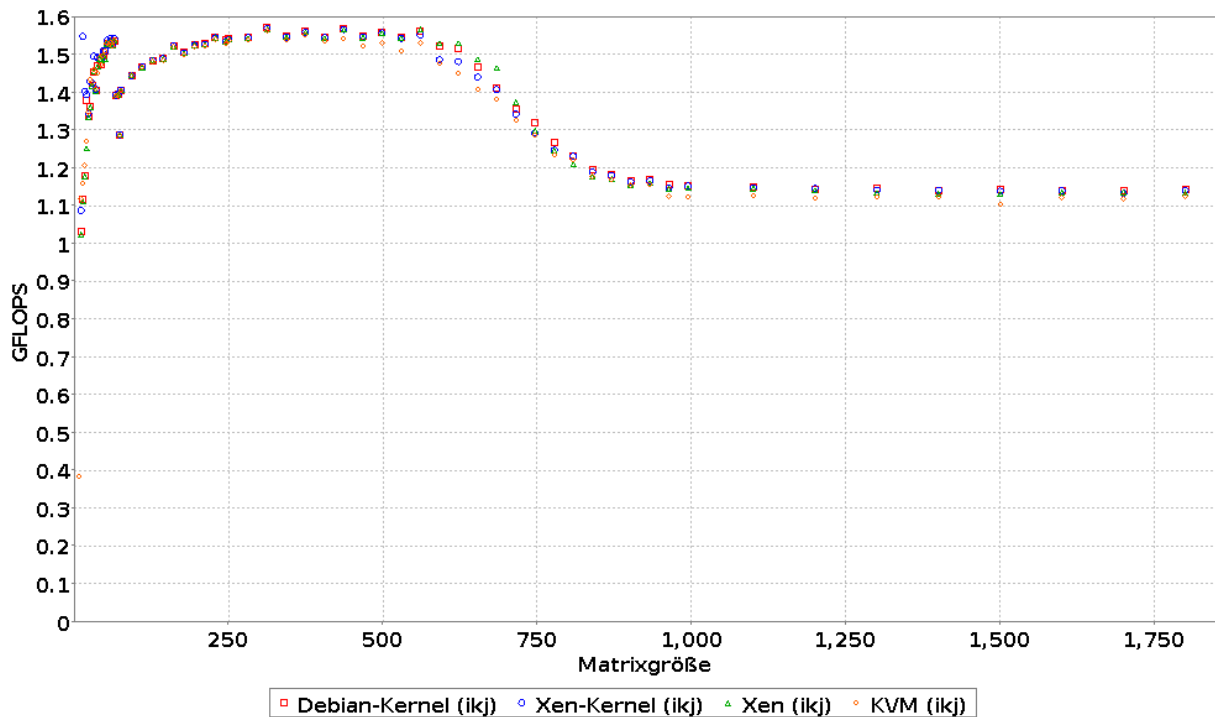
- 128 MB RAM
- Software
 - Gcc version 4.4.5 (Debian 4.4.5-8)
 - Open MPI 1.4.2
- Dienste
 - Openssh-server

Je nach Messung und untersuchten Effekten wurden Konfigurationen mit 1,4,6 oder 8 Prozessoren gewählt.

7.2 Matrixmultiplikation

Um die Rechenleistung der virtuellen Maschinen besser beurteilen zu können wurde eine Matrixmultiplikation verwendet.[ZIH]

Als erstes eine Messung auf nur einem Kern, sowohl virtuell als auch auf dem Hostsystem selbst.

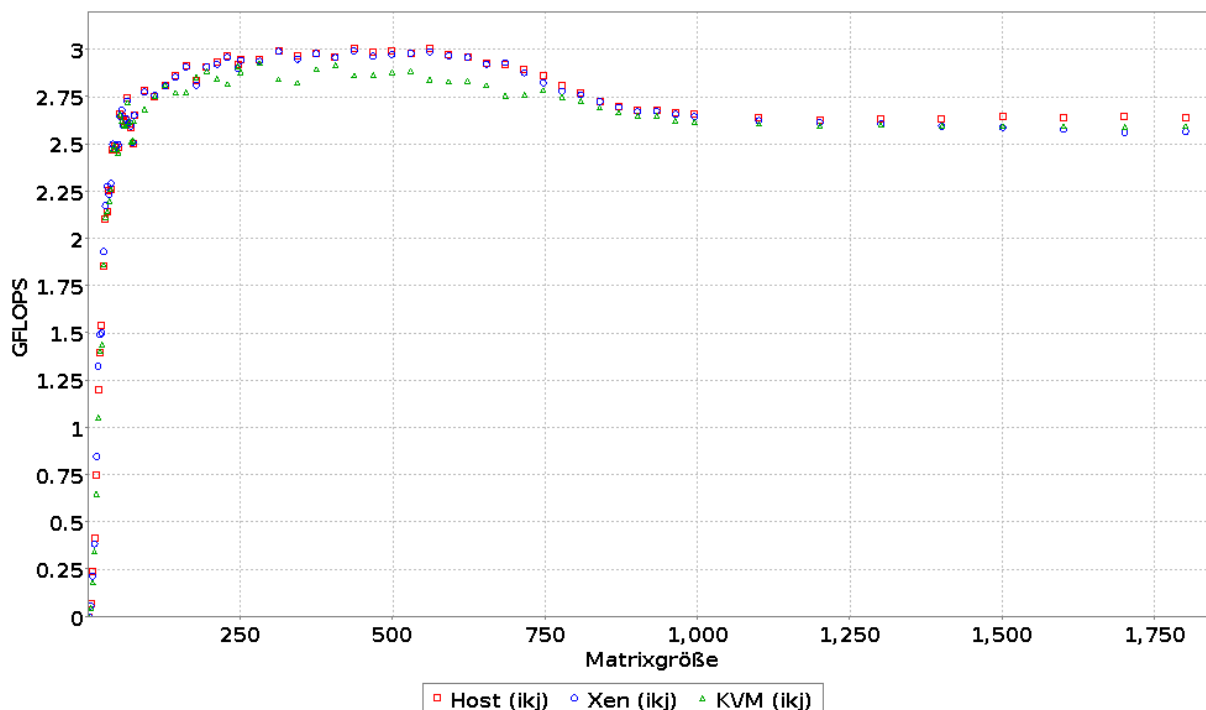


Bei dieser Messung fällt auf, dass alle vier Messungen die annähernd die gleichen Ergebnisse liefern.

Da das restliche System nicht ausgelastet ist kann einer der vier Kerne komplett dem virtuellen Prozessor zugeordnet werden, wodurch es zu keinen Leistungseinbußen kommt.

7.3 OpenMP(4)-Matrixmultiplikation

Zum Testen der vollen Auslastung des Systems mit 4 Kernen wurde eine Matrixmultiplikation mit mithilfe von OpenMP gestartet. In einem ersten Test mit 4 Threads. Diese Messung wurde auf dem Hostsystem und virtuellen Maschinen mit 4 Kernen durchgeführt.



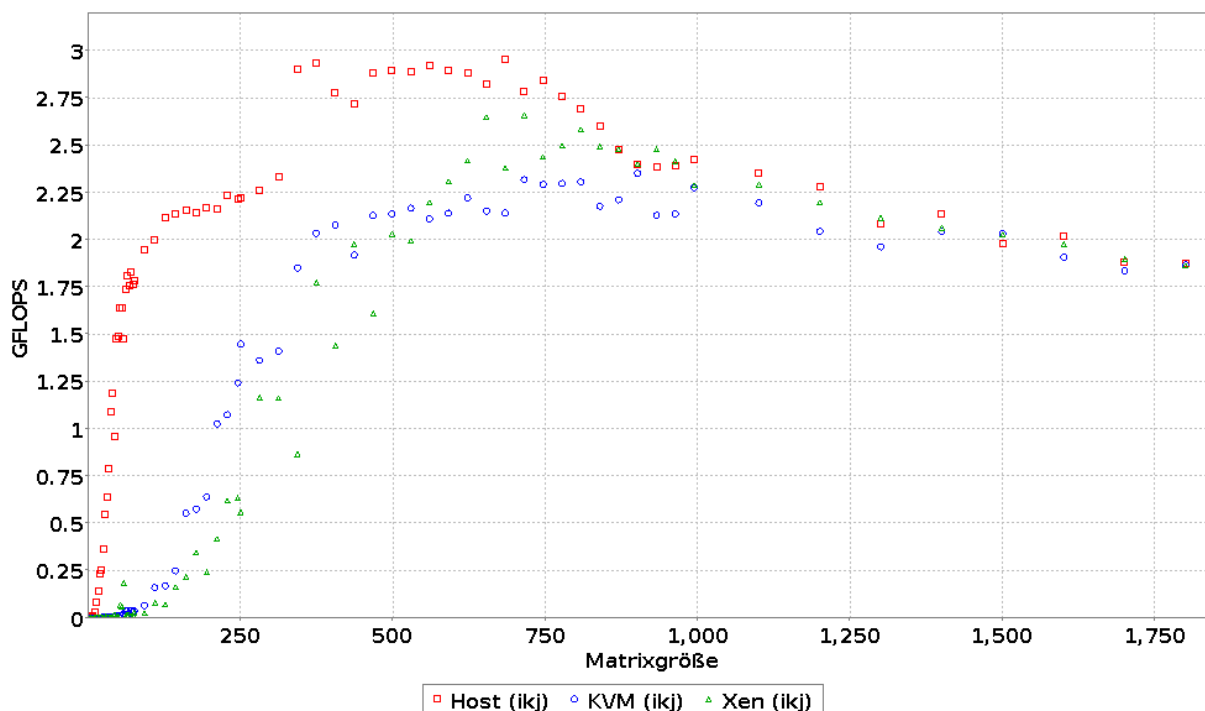
Selbst bei voller Auslastung des Systems können die virtuellen Maschinen zeigt Xen keine Performanceeinbrüche. Nur im Falle von KVM sind im Bereich des L2-Caches Einbrüche von rund 5 % im Vergleich zum Hostsystem zu sehen. Im Speicher verlieren sowohl Xen als auch KVM leicht (1-2 %) an Performance.

7.4 OpenMP(6)-Matrixmultiplikation

Durch eine Messung mit 6 Threads kann ein unbalancierter Überlastfall erzeugt werden.

In diesem Fall wurden die 6 Thread auf dem Hostsystem auf den 4 Kernen gestartet und anschließend vom Betriebssystem den Kernen zugewiesen wird.

Die virtuelle Maschine wurde mit sechs Prozessoren erstellt und damit mit mehr virtuellen Prozessoren als physisch vorhandene Prozessoren.



Wie zu erwarten verliert man durch die Überlast und dem damit verbundenen ständigen Taskswitch auf dem Hostsystem an Performance.

Jedoch sind die Einbrüche bei den virtuellen Maschinen sehr deutlich.

Matrixgrößen, die noch im Cache berechnet werden könnten werden sehr langsam, teilweise gerade einmal 100 MFLOP. Erst bei größeren Matrizen steigt die Performance an.

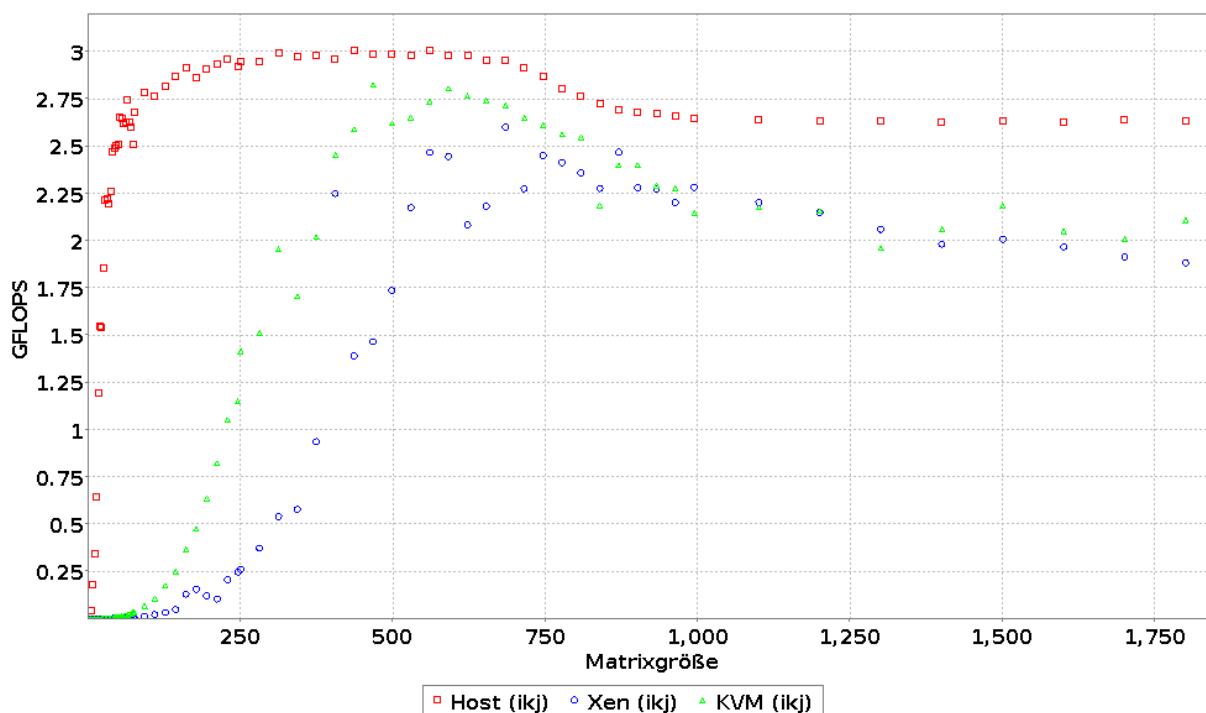
Im Speicher sind die virtuellen Maschinen wieder gleichauf mit dem Hostsystem.

Jedoch fällt auf, dass bei kurzen Aktivitäten der virtuellen Maschinen beträchtliche Leistungseinbußen zu verzeichnen sind.

7.5 OpenMP(8)-Matrixmultiplikation

Bei einer Messung mit 8 Kernen liegt ein balancierter Überlastfall vor, da idealerweise jeweils zwei Threads auf einem Kern laufen. Auch hier wurde auf dem Hostsystem die 8 Threads auf den vier vorhandenen Kernen gestartet.

die virtuellen Maschinen wurden mit acht virtuellen Kernen erstellt.



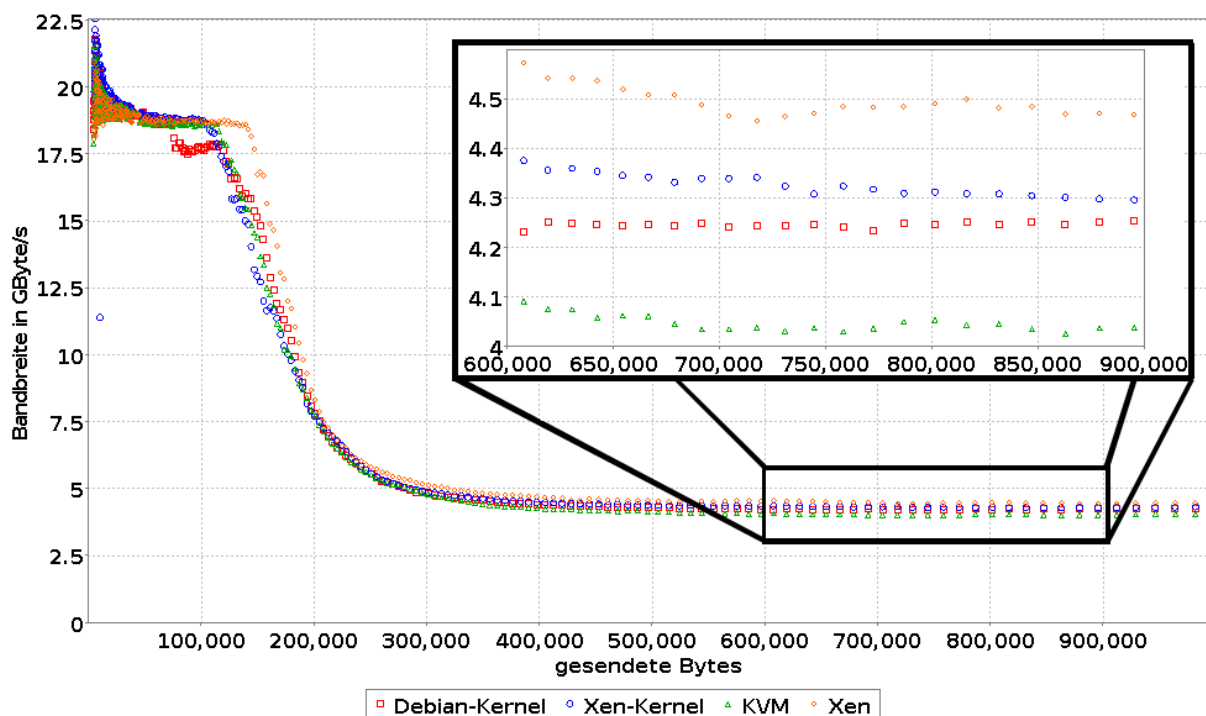
Auf dem Hostsystem erreicht man mit doppelter Threadanzahl die selbe Performance.

Bei den virtuellen Maschinen sind wieder, genau wie bei der Messung mit sechs Kernen, deutliche Performanceeinbuße zu sehen.

Im Gegensatz zur Messung mit 6 Threads sind hier auch Unterschiede bei Messungen mit Matrixgrößen, die nicht mehr in den Cache passen, zu sehen. Im kompletten Verlauf der Messung zeigt KVM eine bessere Performance als Xen.

7.6 Memory Bandbreite

Um die Leistung der Virtualisierungen beurteilen zu können wurde auch die Memory Bandbreite gemessen.



Die Speicherbandbreite zeigt nur leichte Unterschiede zu erkennen, erst bei genauerer Betrachtung fallen Unterschiede auf.

Erstaunlicherweise ist die mit Xen virtualisierte Maschine die mit der größten Bandbreite, gefolgt vom Hostsystem mit Xen-Kernel. Erst dann kommt das Hostsystem mit Standardkernel und als Schlusslicht hat sich KVM erwiesen.

Diese Effekte, dass die virtuellen Maschinen schneller werden, als das eigentliche Hostsystem lässt sich durch die zusätzliche Verwaltungshierarchie, die für die Speicherzugriffe zuständig ist, damit treten Cache-Effekte auf.

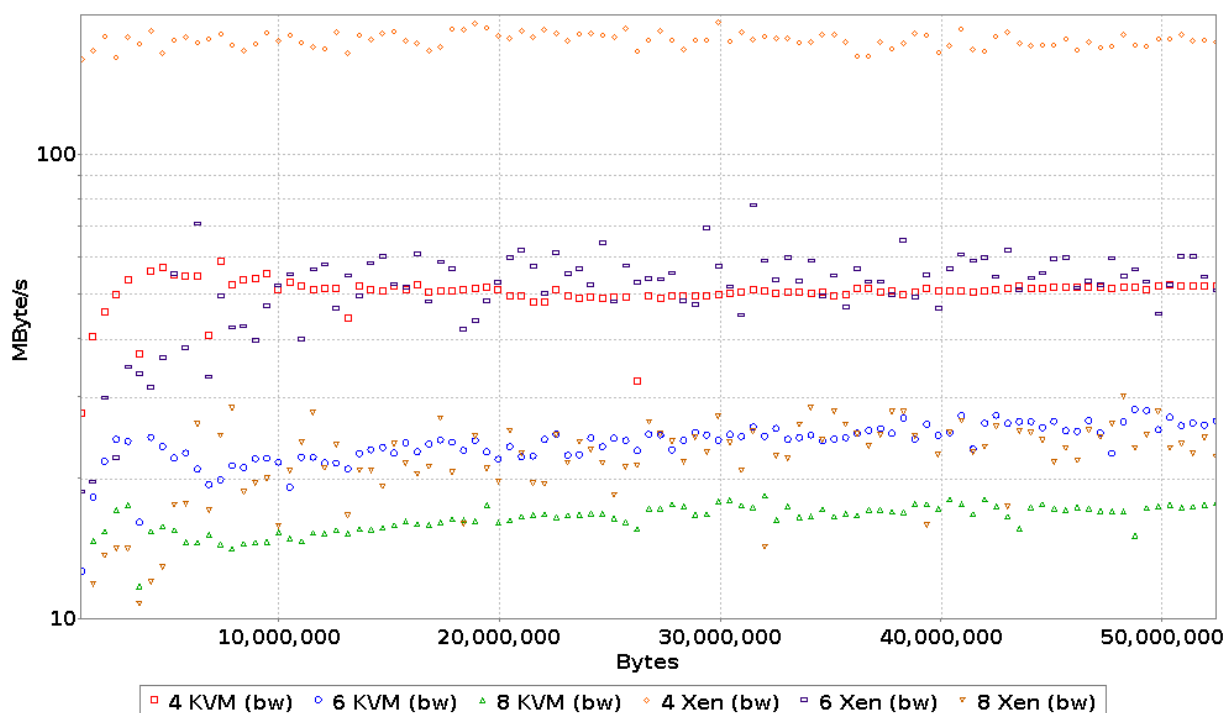
Diese Effekte lassen die Bandbreite etwas höher erscheinen.

7.7 MPI-Kommunikation

Nun soll die Kommunikationsbandbreite zwischen den virtuellen Maschinen getestet werden.

Für diesen Test wurden entsprechend viele virtuelle Maschinen gestartet und mithilfe von MPI die Kommunikationsbandbreite gemessen.

Dabei ordnet der Messkern die Maschinen einem Binärbaum zu und misst jeweils mit einer Punkt zu Punkt Kommunikation. Demnach wird eine Broadcast Kommunikation zu allen Maschinen realisiert.



Zur besseren Darstellung der Messergebnisse wurde eine logarithmische Skala verwendet.

Die Kommunikation zwischen den virtuellen Maschinen verläuft mir gerade einmal 120 MB/s ab und das auch nur im Falle von vier virtuellen Xen-Maschinen. Alle anderen Testfälle fallen schlechter aus.

Wobei KVM jeweils langsamer ist als Xen.

7.8 Hdparm -t

Zur ersten Einschätzung der Plattengeschwindigkeit kann das Linux-Tool “hdparm” verwendet werden. Diese misst die Lesegeschwindigkeit einer Festplatte.

Dieser hdparm-Test wurde mit Debian- und Xen-Kernel auf dem Hostsystem, KVM und Xen-virtuellen Maschinen mit einer Image-Datei als Festplatte und einer Variante mit LVM-Festplatte.

Messung	Debian-Kern	Xen-Kern	Xen (Image)	Xen (LVM)	KVM
1	40.97	41.92	34.59	18.42	4.16
2	41.97	40.15	124.78	17.78	20.92
3	41.62	40.31	406.64	15.26	31.43
4	38.84	43.25	501.85	14.50	37.75
5	43.56	43.20	518.54	17.97	42.24
...					
22	41.75	41.13	491.23	15.12	219.93

Je öfter man misst, desto schneller scheinen die virtuellen Festplatten zu werden.

Nur bei LVM tritt dieser Effekt nicht auf.

Im Falle von Image-Dateien die virtuelle Festplatte bei entsprechend häufigen Zugriffen immer weiter in den Arbeitsspeicher kopiert. Dadurch wird immer weniger von der physischen Festplatte gelesen, sondern die Messungen spiegeln immer stärker die zwischengespeicherte virtuelle Festplatte im Arbeitsspeicher wieder.

Bei Xen wird dieser Wert schon nach 4 Messungen stabil, bei KVM dauerte eine derartige Stabilisierung 22 Messungen.

Trotz des gleichen Effektes ist Xen um mehr als Faktor 2 schneller als KVM und sogar um rund Faktor 12 schneller als ein natives System. KVM erreicht immerhin noch rund Faktor 6 schnellere Lesegeschwindigkeiten als das Hostsystem selbst.

Eine virtuelle Maschine mit LVM wird nicht schneller, da die Daten direkt auf die physische Festplatte geschrieben werden und nicht im RAM zwischengespeichert.

Im Gegenteil durch die Verwendung von LVM werden die Lesegeschwindigkeiten um Faktor 2 langsamer.

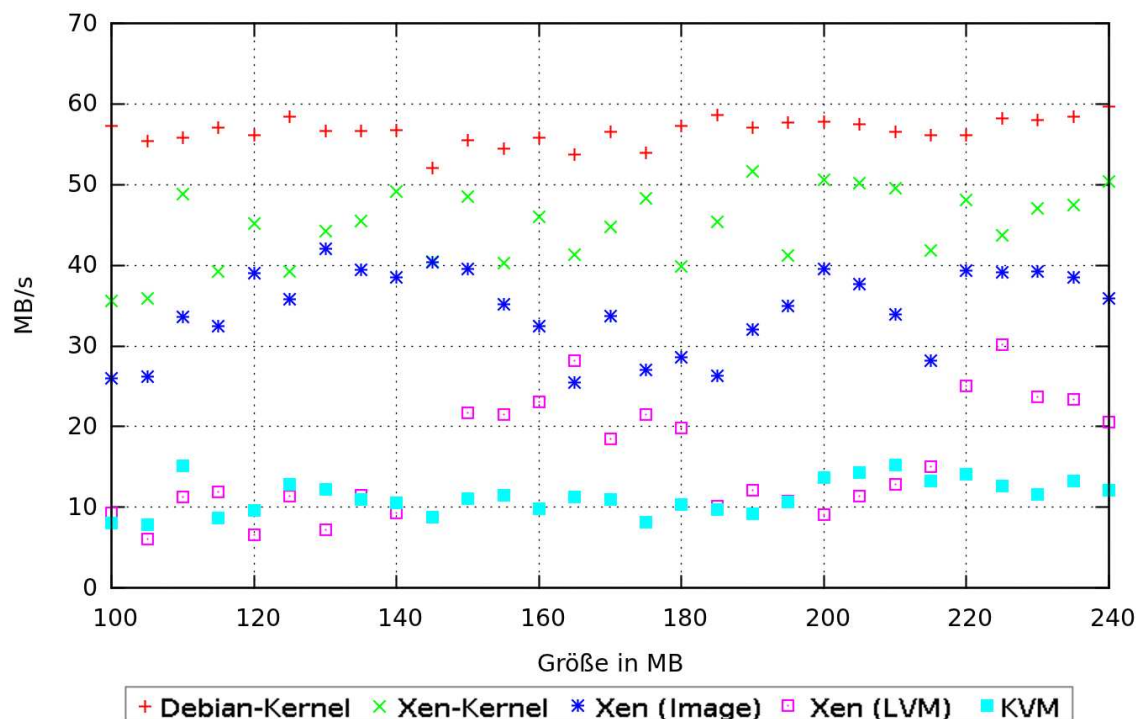
7.9 Lesen ohne RAM

Um dieses Cacheverhalten aus den Messung zu vermeiden wurden zwei kleine Programme erstellt.

Das Erste schreibt Dateien definierter Größe mit Zufallsdaten.

Das Zweite liest diese Daten und misst die dafür benötigte Zeit.

Zwischen dem Schreiben und lesen wurde der komplette Rechner neu gestartet um den Arbeitsspeicher sicher zu leeren.



Mithilfe dieses Umweges zeigen sich die wirklichen Leistungseinverluste bei virtuellen Maschinen. So erreicht eine Xen-virtuelle Maschine mit Image rum 30 MB/s und Xen-virtuelle Maschinen mit LVM und KVM jeweils 10 MB/s.

8 Zusammenfassung

Eine effektive Virtualisierung stellt spezielle Herausforderung an die Hard- und Software, um einen sicheren Betrieb der virtuellen Maschinen zu gewährleisten.

Dabei unterscheidet man zwischen der Paravirtualisierung und der hardwarebasierten Virtualisierung, zwei verschiedene Techniken, die allerdings auch in Kombination eingesetzt werden können.

Der wichtigste Teil einer Virtualisierungslösung ist der Hypervisor, dieser steuert den Hardwarezugriff der virtuellen Maschinen.

Ein Vergleich der beiden freien Virtualisierungslösungen Xen und KVM hat gezeigt, dass es kaum Performanceverluste gibt, sofern man nicht mehr virtuelle Kerne verwendet, als im System wirklich vorhanden sind.

Die größten Unterschiede finden sich im Bereich der Festplattenzugriffe und der Kommunikation zwischen den VMs.

Hier hat Xen die besseren Werte erzielen können.

Literatur

- [AMDa] AMD. *AMD Tapper*. <http://developer.amd.com/zones/opensource/AMDTapper/Pages/default.aspx>
- [AMDb] AMD. *Tapper Manual*. <https://github.com/amd/Tapper-Manual>
- [Fis09] FISCHER, Marcus: *Xen - Das umfassende Handbuch*. Galileo Computing, 2009
- [KVM] KVM. *KVM*. <http://www.linux-kvm.org>
- [Pop] POPEK, Goldberg. *Formal requirements for virtualizable third generation architectures*. <http://dl.acm.org/citation.cfm?doid=361011.361073>
- [VMW] VMWARE. *Software and Hardware Techniques for x86 Virtualization*. http://www.vmware.com/files/pdf/software_hardware_tech_x86_virt.pdf
- [ZIH] ZIH. *BenchIT*. <http://www.benchit.org>