INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Empirische Identifikation von Parametern mit Einfluss auf die Effizienz von Virtualisierung – am Beispiel Xen

Bastian Gebhardt

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Empirische Identifikation von Parametern mit Einfluss auf die Effizienz von Virtualisierung – am Beispiel Xen

Bastian Gebhardt

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Dr. Vitalian Danciu Dr. Nils gentschen Felde Dr. Tobias Lindinger
Abgabetermin:	29. Juli 2010

Hiermit versichere ich, dass ich die vorliegende Projektarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 7. Juli 2077

. . . . . .
(Unterschrift des Kandidaten)

Abstract

In der heutigen Zeit rückt die Virtualisierung von Servern immer mehr in den Vordergrund. Nicht nur durch die Medien, in denen Virtualisierungskonzepte und Produkte, der auf diesem Sektor stark voranschreitenden Softwareentwicklung nahezu täglich angepriesen werden, auch in Rechenzentren und in kleineren Serverräumen ersetzen virtuelle Maschinen immer öfter einzelne, "richtige" Serverrechner. Allerdings ist eine Servervirtualisierung nicht verlustfrei zu realisieren. Virtualisierung von Hardware hat Einfluss auf Antwortzeiten und Datenraten aller verwendeter Hardware.
Wie groß dieser Einfluss ist, soll in dieser Arbeit anhand unterschiedlicher Testszenarien bei der quelloffenen Version des Virtualisierers Xen näher untersucht werden. Xen wurde ursprünglich an der Universität von Cambridge entwickelt, und wird ständig weiterentwickelt. Durch die freie Lizenz und die dadurch resultierende kostenlose Verfügbarkeit ist dieses Produkt auch für kleinere Unternehmen oder auch private Projekte realisierbar. Aus diesem Grund, der Tatsache, dass es unzählige Virtualisierungslösungen gibt, und eine vollständige Untersuchung den Rahmen dieser Arbeit bei weitem sprengen würde, wurde der Augenmerk alleine auf Xen gelegt.
An dieser Stelle sei erwähnt, dass es drei weitere Arbeiten unter identischen Hardware- und Softwarevoraussetzungen von Eva Tsoiprou, Günter Lemberger und Anton Romanjuk gibt, die die Virtualisierer OpenVZ/Virtuozzo, VMware ESXi und Microsoft Hyper-V näher betrachten.
In dieser, wie in jeder der erwähnten drei, wird im ersten Schritt der reine Verlust der Virtualisierung bestimmt, um dann im zweiten Schritt die Leistung unter Mehr-VM-Betrieb zu evaluieren.

Inhaltsverzeichnis

1. Einleitung

In der heutigen Zeit werden beispielsweise in Firmen immer häufiger eine große Anzahl an Rechnern benötigt. Während vor einiger Zeit jeder Serverrechner noch durch einen physischen Rechner dargestellt wurde, werden heute vermehrt alle Server durch virtuelle Maschinen realisiert. Dadurch werden nur wenige, aber leistungsfähige physische Rechner benötigt. Der Vorteil dieser Vorgehensweise liegt auf der Hand. Benötigen gleichzeitig nur wenige Rechner eine hohe Rechnerlast, kann die nicht benötigte Leistung nicht an andere Rechner weitergegeben werden. Bei vielen virtuellen Maschinen auf einem physischen Rechner ist hingegen eine Lastverteilung möglich. Desweiteren können die physischen Ressourcen effektiver genutzt werden. Außerdem hat man bei diesem Prinzip eine viel größere Dynamik. Zusätzliche Rechner können auch nur kurzzeitig bei Bedarf hinzugeschaltet werden. Allerdings müssen Teile der Ressourcen für das Ressourcenscheduling der VMs verwendet werden, wodurch auch Verzögerungen (Latenzen) auftreten können. An diesem Punkt möchte diese Projektarbeit ansetzen. Es soll evaluiert werden wie hoch die Effizienz der virtuellen Maschinen und auch der Hardwareunterstützung ist.

1.1. Motivation und Ziel

Durch Verwendung von Virtualisierung kommt es von Natur aus zu zusätzlichem Rechenaufwand. Hardwareelemente müssen in VMs nachgebildet werden und an die physische Hardware des Virtualisierers weitergeleitet werden. Laufen mehrere VMs gleichzeitig auf einem Virtualisierer, muss der Fokus durch einen Ressourcenscheduler im Wechsel an die verschiedenen VMs gegeben werden. Dieser zusätzliche Aufwand an Rechenleistung schlägt sich negativ auf die eigentliche Leistung in den virtuellen Maschinen nieder. Der Verlust verglichen mit einem nativen System, also der Wirkungsgrad des Virtualisierers, soll in dieser Projektarbeit evaluiert werden. Dazu werden verschiedenste Parameter bestimmt, die Einfluss auf den Wirkungsgrad haben könnten, zum Beispiel das Betriebssystem, die Architektur in den VMs oder ob eine Befehlssatzerweiterung für die Optimierung virtueller Systeme im Host aktiviert ist. Alle Iterationen dieser Parameter werden dann durch geeignete Methoden mit vergleichbaren nativen Systemen auf ihren Wirkungsgrad hin untersucht. Daraufhin soll anhand dieser Untersuchungen eine optimale Konfiguration dazu verwendet werden, um den Hypervisor auf seine Belastbarkeit hin zu überprüfen.

1.2. Vorgehensweise

Die Ausarbeitung gliedert sich wie folgt.
In Kapitel 2 wird kurz auf die nötige Theorie eingegangen. Dabei wird Servervirtualisierung im Allgemein erklärt, verschiedene Arten der Virtualisierung aufgezeigt und Begriffe wie “Hypervisor” erläutert. Dazu wird aufgezeigt, welche Eigenheiten und damit auch Probleme für diese Arbeit Virtualisierung mit sich zieht. Danach wird die Problematik erwähnt, die

bei Messungen in virtuellen Systemen einhergeht, und welche Ansätze verfolgt wurden, um diese zu lösen. Danach wird der in dieser Arbeit verwendete Virtualisierer vorgestellt und Eigenschaften und Einzelheiten erklärt, die spätere Ergebnisse begründen können.
In Kapitel 3 wird näher auf den Aufbau der physischen Geräte eingegangen und näher erläutert welche Softwarelösungen konkret für die Virtualisierung verwendet wurden.
Dazu wird zunächst die verwendete Hardware, die zur Verfügung stand, selbst aufgelistet und veranschaulicht, wie sie in einer Infrastruktur integriert wird. Desweiteren wird die aufgebaute Intrastruktur selbst etwas genauer erläutert, beispielweise wie sichergestellt wurde, dass unkompliziert und effizient auch remote auf alle nötigen Dienste zugegriffen werden kann. Im Folgenden werden die verwendeten Betriebssysteme der Gastsysyteme und der nativen Vergleichsrechner vorgestellt und erläutert, wie eine vergleichbare Situation sichergestellt wurde.
Danach werden die einzelnen verwendeten Tools für die einzelnen Testbereiche vorgestellt und deren Funktionsweise und Anpassungen an das Szenario erklärt.
Im Kapitel 4 geht es um die tatsächliche Durchführung der Tests und deren Auswertung. Dazu werden die Testspezifikationen aufgelistet und erklärt. Dabei wird in sogenannte “synthetische Tests” und parallel durchgeführte Tests getrennt. Bei ersterem wird eine komplett künstliche Situation geschaffen, indem nur eine einzige VM auf einem Hypervisor läuft und auch nur eine Hardwarekomponente (CPU/RAM/Disk/Netz) der VM unter Last gesetzt und bewertet wird. Die Testsituation in zweiten Teil ist etwas realistischer. Zwar werden nach wie vor kontrolliert einzelne Komponenten der Hardware getestet, nur laufen auf dem Hypervisor mehrere (durch die Einschränkungen der Hardware allerdings nur zwei bzw. drei) virtuelle Maschinen gleichzeitig. Auf diesen VMs laufen dann je nach Szenario jeweils einer oder mehrere Tests.
Im letzten Kapitel 5 wird ein kurzes Fazit über die Beobachtungen geschlossen und ein Ausblick über weitere auf diese Arbeit aufbauende Arbeiten gegeben.

2. Theorie und Grundlagen

Seit der Begriff “Servervirtualisierung” das erste Mal in der Informatik aufgekommen ist, hat sich auf diesem Bereich viel getan. Es wurden verschiedenste Konzepte der Virtualisierung entwickelt. Teilweise bauen diese aufeinander auf, teilweise handelt es sich um Kombinationen mehrerer Konzepte. Aus diesem Grund soll nun kurz auf einzelne Virtualsierungstechniken und -software eingegangen werden.

2.0.1. Virtualisierung

Eine virtuelle Maschine, kurz VM, ist ein virtueller Computer. Eine solche Maschine besteht nicht aus Hardware, sondern aus Software. Das Betriebssystem auf diesem Rechner wird also nicht wie üblich direkt auf der physischen Hardware installiert, sondern wird auf eine Zwischenschicht aufgesetzt [vmw].

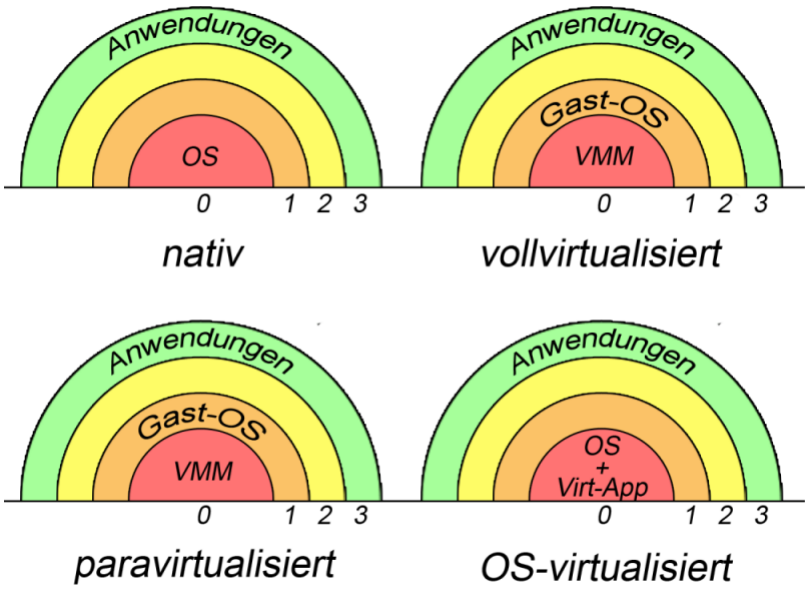


Abbildung 2.1.: Ringmodell, nativ und virtualisiert

2.0.2. Hypervisor

Zwischen die virtualisierten Systeme und der Hardware wird eine Zwischenschicht, der so genannte “Hypervisor” oder auch “Virtual Machine Monitor” (VMM), geschoben. Dieser stellt eine Schnittstelle zwischen der physischen und der virtualisierten Hardware der VMs dar.
Ein Hypervisor kann dabei *hardwarebasiert* oder *hostbasiert* sein. Hardwarebasiert (teilweise

auch als “Typ1 Hypervisor” bezeichnet) bedeutet, der Hypervisor setzt direkt auf der Hardware auf, Gastsysteme sitzen ebenso wie eine privilegierte Einheit auf dem Hypervisor auf. Ein hostbasierter Hypervisor, bzw. ein Hypervisor vom Typ 2, setzt im Gegensatz zu Typ 1 nicht direkt auf der Hardware auf, sondern läuft auf einem vollwertigem Betriebssystem als Programm. Dabei kommt es zu keiner strikten Abkapselung zwischen Hypervisor, Betriebssystem und virtuellen Maschinen, da der Hypervisor eine Anwendung des unmodifizierten Betriebssystems ist [ibm05] [Spe05].
Das Sicherheits- und Privilegiensystem in einem Betriebssystem auf x64 Architektur wird standardmäßig durch ein Ringmodell mit Ringen von 0 bis 3 dargestellt. Während der innerste Ring, Ring 0 den privilegiertesten Bereich darstellt, wird es nach außen immer unprivilegierter. Je privilegierter ein Ring ist, desto mehr CPU-Befehle stehen zur Verfügung. Der Betriebssystemkern und Hardwaretreiber befinden sich in Ring 0, während Benutzeranwendungen sich in Ring 3 befinden. Die mittleren Ringe 1 und 2 werden standardmäßig nicht verwendet. Der Grund für dieses Prinzip ist, dass verhindert werden soll, dass Anwendungen auf den kompletten Speicher oder ähnlich kritische Bereiche zugreifen kann.
Bei einem Typ 2 Hypervisor ändert sich an diesem Modell nichts. Der Hypervisor läuft wie erwähnt als Anwendung, befindet sich also in Ring 3 und nutzt die Treiber des Betriebssystems zum Zugriff auf die Hardware.
Ein Typ 1 Hypervisor setzt wie erwähnt direkt auf der Hardware auf. Das Gastbetriebssystem wird also einen Ring nach außen auf den bisher unbenutzten Ring 1 geschoben und der Hypervisor sitzt auf Ring 0. Die Anwendungen des Gastbetriebssystem bleiben wie gehabt auf Ring 3. Damit hat der Virtualisierer vollen Hardwarezugriff und kann die Anfragen der virtuellen Maschinen verarbeiten und weiterleiten. Eine VM ist dabei von der physischen Hardware und anderen VMs komplett isoliert. Gegenseitiger Zugriff auf Speicher oder ähnliche Ressourcen sind damit ausgeschlossen.

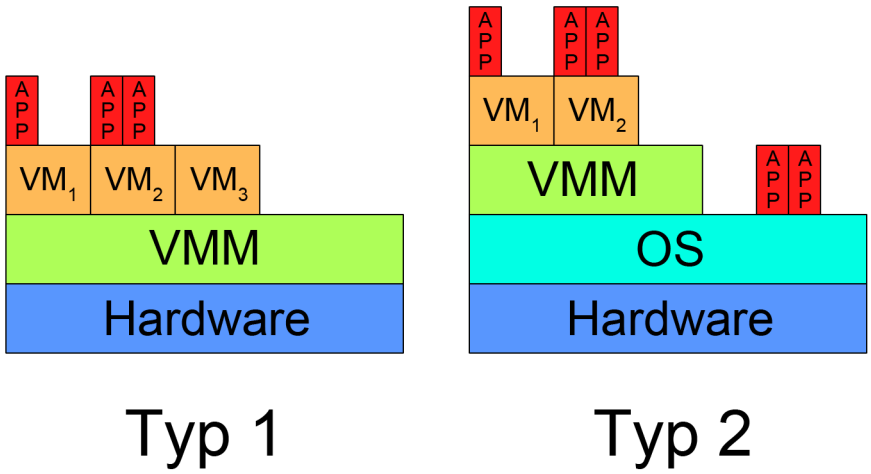


Abbildung 2.2.: Vergleich der Hypervisortypen

2.0.3. Typen

Die Gastsysteme selbst, die auf einem Hypervisor laufen, können dabei auf verschiedene Arten virtualisiert werden. Es wird grundsätzlich zwischen vier Virtualisierungsarten unterschieden.
Bei der Emulation wird die komplette Hardware vollständig durch Software nachgebildet. Dabei lassen sich beispielsweise CPUs mit einer anderen Architektur als die der Hardware CPU darstellen. Ebenso kann eine andere Hardware wie Grafikkarten und Netzwerkkarten emuliert werden.
Bei der Vollvirtualisierung wird versucht die Hardware nicht zu emulieren, sondern die tatsächlich eingebaute Hardware virtuell zur Verfügung zu stellen. Der Grund hierfür ist, dass eine Emulation immer mit Rechenleistung verbunden ist, während eine reine Virtualisierung kaum Rechenaufwand verursacht. Zugriffe auf die virtuelle Hardware werden dabei lediglich an die physische Hardware weitergeleitet. In der Praxis ist dies nur bei CPU und Hauptspeicher problemlos möglich. Allerdings müssen auch hier teilweise CPU Befehle übersetzt werden, während weitere Hardwarekomponenten in der Regel emuliert werden. Es werden dabei Standardgeräte emuliert, für die in allen gängigen Betriebssystem Treiber vorhanden sind.
Bei Paravirtualisierung wird versucht den Virtualisierungssaufwand zu reduzieren. Dies wird durch eine Anpassung des Gastsystems erreicht. Dadurch "weiß" das Betriebssystem, dass es virtualisiert ist. Dazu stehen dem Gastsystem so genannte "Hypercalls" zur Verfügung. Dabei handelt es sich um Erweiterung der Binärschnittstelle, also der Kommunikation des Betriebssytems mit der Hardware. Statt wie bei normalen Systemaufrufen mit der in diesem Fall virtuellen Hardware zu kommunizieren, ist es mit mit Hypercalls möglich von einer VM aus direkt den Hypervisor anzusprechen.
Beim letzten Konzept, der Betriebssystemvirtualisierung, setzt die Virtualisierungsroutine auf einem Standard-Betriebssystem auf. Es wird kein komplettes System virtualisiert, sondern nur innerhalb voneinander getrennter Container eine Laufzeitumgebung im Host-Betriebssystem virtuell zur Verfügung gestellt. Bei der Virtualisierungsroutine handelt sich hierbei also nicht um eine eine Abstraktionsebene zwischen Hardware und Gastsystemen. Siehe auch Abbildung 2.2.

2.0.4. AMD-V

AMD-V (was für "AMD Virtualization" steht) ist eine Virtualisierungsunterstützung der CPU. Wie der Name vermuten lässt, handelt es sich hierbei um eine Entwicklung der Firma AMD für ihre Prozessoren. Ein vergleichbares, aber unabhängig davon entwickeltes Konzept für Intel Prozessoren bietet Intel unter dem Namen *Intel VT* an.
Diese Konzepte sind Befehlssatzerweiterungen der CPU, realisiert durch eine "Secure Virtual Machine" (SWM) [AMD05]. Durch Aktivierung der SVM ist eine höhere Geschwindigkeit bei der Virtualisierung zu erwarten, da zusätzliche CPU-Befehle vorhanden sind, die CPU-Befehle ersetzen, die nicht oder nur mit hohem Aufwand zu übersetzen sind. Manche Virtualisierungstechniken sind auch auf diese SVM angewiesen. So ist AMD-V bei Paravirtualisierung unter Xen optional, bei Vollvirtualisierung, auch unter Verwendung paravirtualisierter Treiber, allerdings Voraussetzung.

2.0.5. Zeitgebung in virtuellen Maschinen

Eine korrekte und auch bei kurzen Intervallen präzise Zeitgebung ist allgemein eine wichtige Anforderung an Rechnersysteme. Bei virtuellen Maschinen wird man vor zusätzliche Herausforderungen gestellt, da durch den vermehrt auftretenden Verwaltungsaufwand Ungenauigkeiten auftreten. Diese sind bedingt durch vermehrte Kontextwechsel zwischen dem Hostsystem und den Gastsystemen. Benötigt man eine sehr präzise Zeitgebung und Zeitmessung, wie in diesem Fall bei Messungen von Benchmarks, ist die vorliegende Zeit in virtuellen Maschinen oft nicht ausreichend genau, da gerade unter Last keine genauen Messungen erzielt werden können.

Um dieses Problem zu beheben und eine angemessene Umgebung zu finden, gibt es verschiedene Ansätze. Eine Methode sind Client-Server Ansätze. Bei diesem Prinzip wird die Zeitgebung komplett von der virtuellen Maschine getrennt und extern gemessen. Die Zeit wird dann über das Netz an den Zielrechner übermittelt. Eine Implementierung dieses Prinzips ist das Protokoll NTP ("Network Time Protokoll", "Netz Zeit Protokoll"), das auf eine verbindungslose Kommunikation mittels UDP Paketen aufbaut. Eine ähnliche Methode, die allerdings etwas mehr zusätzliche Programmierung erfordert, wurde in diesem Fall angewandt.
Gegeben den Fall das verwendete Benchmarktool besitzt eine eigene, von dem eigentlichen Algorithmus zur Lasterzeugung abgekapselte Methode zur Zeitgebung, kann man diese Methode auf eine Client/Serverstruktur hin anpassen. Alle Zugriffe auf lokale Betriebssystemstrukturen werden durch Remoteanfragen an einen externen Zeitgeber ersetzt. Basiert diese Netzkommunikation auf ein verbindungsbasiertes Protokoll wie TCP, muss vor dem Aufruf der eigentlichen Benchmarkroutine noch die Verbindung zum Zeitserver aufgebaut werden. Basiert sie auf verbindungsloser Kommunikation ist keine weitere Anpassung nötig.
Eine weiter Methode zur genaueren Zeitgebung basiert auf der Annahme, dass bei Erhöhen der Intervallgröße zwischen dem einzelnen Abfragen der Zeit der Zeitunterschied zur tatsächlich verstrichenen Zeit sinkt. Wählt man diese Intervalle nun groß genug, so dass der Fehler der Zeitmessung unerheblich klein wird, und besteht die Möglichkeit das in den Benchmarkalgorithmus zu integrieren, kann man vernünftige Ergebnisse erzielen ohne die ungenaue Zeitgebung auszulagern oder erreichen, dass kleine Intervalle präzise sind.

2.1. Über Xen

Bei dem bei dieser Arbeit verwendeten Hypervisor handelt es sich um Xen, ein Hypervisor vom Typ 1. Für Xen benötigt man einen speziell angepassten Linux Kernel. Nach Booten dieses Kernels wird automatisch eine erste virtuelle Maschine, eine privilegierte Domäne, die sogenannte *dom0*, gestartet. Sie dient als Schnittstelle zur Konfiguration des Hypervisors. Außerdem können durch diese Domäne die unprivilegierten Domänen, die sogenannten `domUs`, konfiguriert und kontrolliert werden. Beispielsweise können mittels des Kommandos `xm` unter anderem virtuelle Maschinen gestartet bzw. neu gestartet, gestoppt, zur Laufzeit der vergebene Arbeitsspeicher angepasst oder vCPUs (die virtuellen CPUs der Gastsysteme), in den virtellen Maschinen hinzugefügt oder entfernt werden.
Außerdem kann man über die Domäne 0 auf das komplette Dateisystem des Hosts zugreifen, auf dem auch die Imagedateien der virtuellen Maschinen liegen können. Auf diesen Images sind die virtuellen Festplatten der VMs gespeichert. Zudem kann man das darauf liegende

Dateisystem in das vorliegende System einbinden und so auf das Dateisystem zugreifen.
Des Weiteren läuft in der privilegierten Domäne `xend`, der Daemon von Xen. Ein Daemon (oder auch Service oder Dienst) ist ein Prozess, der im Hintergrund läuft und auf ein Signal wartet um aktiv zu werden. Dieser Daemon kontrolliert die virtuellen Ressourcen und ist nötig, um mit dem Hypervisor zu kommunizieren. Des Weiteren werden die Parameter der privilegierten Domäne und das allgemeine Verhalten von Xen über den xend gesteuert.

2.1.1. CPU

Jede `domU` bekommt eine Anzahl an virtuellen CPU Kernen ("Cores") zugewiesen. Diese können allerdings, wie oben bereits erwähnt, zur Laufzeit mit Hilfe des Tools `xm` erhöht oder verringert werden. Die Anzahl der Kerne in der VM ist allerdings unabhängig von der Anzahl an tatsächlichen Kernen des Hosts.
Der Standardscheduler von Xen ist der Credit Scheduler. Er vergibt Punkte, so genannte Credits an die vCPUs der domUs. Anhand der Anzahl der Credits wird die Priorität einer vCPU bestimmt, je höher die Anzahl ist, desto priorisierter wird die vCPU behandelt, das bedeutet, umso früher bekommt sie Rechenzeit zugewiesen.
Ist eine vCPU im Leerlauf, bekommt sie in regelmäßigen Abständen Credits. Ist eine vCPU einem reellen Kern zugeordnet und mind. ein Prozess in der VM befindet sich auf dieser vCPU in Ausführung, verbraucht diese vCPU Credits. Durch dieses Prinzip ist auch ein Wechsel der vCPU zu einem anderen Kern möglich. So kann es beispielsweise vorkommen, dass eine vCPU beim aktuell zugewiesenen Kern keine Credits mehr zur Verfügung hat, bei einem anderen, wartenden Kern aber noch unverbrauchte besitzt.
Dabei lässt sich auch für eine `domU` eine Maximalnutzung der CPU bestimmen, ein Wert von 50 entspricht dabei 50% der CPU, selbst wenn noch weitere Ressourcen momentan vorhanden wären. Der Standardwert ist hier bei 0, das entspricht keiner Beschränkung und ist ebenfalls der Wert der für diese Arbeit verwendet wurde.
Des Weiteren lässt sich ein Gewicht einer `domU` angeben. Dabei handelt es sich um Priorisierungen von domUs. Besitzt domU 1 das doppelte Gewicht wie domU 2, wird domU 1 auch doppelt soviel CPU Zeit wie domU 2 zugewiesen bekommen. Standardmäßig besitzen alle domUs das gleiche Gewicht, diese Einstellung wurde in dieser Arbeit nicht verändert.
Möchte man diese Werte dennoch anpassen, kann man dies in der dom0 mit Hilfe des Befehls `xm sched-credit` erreichen. Beispielsweise kann mit `xm sched-credit <domU> -w <Wert>` das Gewicht der Domäne und mittels `xm sched-credit <domU> -c <Wert>` die CPU-Beschränkung eingestellt werden [PUGa].

2.1.2. RAM

Der Arbeitsspeicher, der den virtuellen Maschinen zugewiesen wird, muss selbstverständlich strikt voneinander getrennt werden. Es dürfen zu keiner Zeit zwei Domänen - die privilegierte Domäne eingeschlossen - gleichzeitig Zugriff auf einen gemeinsamen Speicherbereich haben. Eine strikte Trennung der Speicheradressen ist hier unerlässlich.
Allerdings kann der vorhandene physische Speicher nicht statisch verteilt werden, da es vorkommen kann, dass VMs mit einer unterschiedlichen Menge an zugewiesenem Arbeitsspeicher gestartet und beendet werden. Des Weiteren ist es möglich, dass der zugewiesene Speicher während dem VM-Betrieb dynamisch angepasst wird.
Dadurch wird ersichtlich, dass es öfters vorkommen wird, dass sich der VM Arbeitsspeicher

aus Fragmenten des physischen Speichers zusammensetzt. Nicht zusammenhängender Speicher wird von Betriebssystemen aber nicht zufriedenstellend unterstützt [Kra07].
Aus diesem Grund wird bei Xen eine zusätzliche Abstraktionsschicht mit Namen "Pseudo-Physical Memory" eingeführt. Mit Hilfe dieser Schicht wird der zugewiesene Speicher auf den physischen Speicher gemappt und umgekehrt. Konkret gibt es zwei Arten von Tabellen. Die dom0 besitzt zwei Arten von Tabellen. Eine sogenannte *machine-to-physical table* in der Größe proportional zum physischen Speicher, und für jede domU eine *physical-to-machine table* in der Größe proportional zum vergebenen RAM. Mit Hilfe dieser Tabellen können die Speicheradressen des "pseudo-physichen" Speichers in den reallen Speicher und zurück übersetzt werden.
Bei statischer Hauptspeichergröße sind nun keine Probleme mehr zu erwarten. Allerdings gibt es noch Probleme bei der dynamischen Änderung der Arbeitsspeichergröße. Eine Änderung zur Laufzeit würde vom Betriebssystem der Gäste nicht registriert werden. Aus diesem Grund wurde der so genannte Ballontreiber eingeführt, der den RAM von dom0 und den domUs verwaltet [PUGb]. So kann zusätzlich zu dem Eintrag `memory = <ram>`, der die Größe des RAM in Megabyte beim Starten der domU angibt, mittels eines Eintrags `maxmem = <max_ram>` der maximal mögliche RAM in Megabyte angegeben werden, der im laufenden Betrieb der domU zugewiesen werden kann. In dieser Arbeit ist eine dynamische Anpassung allerdings nicht nötig, deswegen wurde keine `maxmem` Zeile eingetragen.

2.1.3. Netz

Die Konfiguration der physischen NICs ("Network Interface Card", englisch für Netzwerkkarte) des Hostrechners erfolgt über die `dom0`. Hier konfiguriert man das Netz, damit die privilegierte Domäne und die unprivilegierten Domänen auf das physische Netz des Hostrechners zugreifen können.
Die physischen Interfaces, sonst unter Linux `eth0`, `eth1`, usw. bezeichnet, werden bei Xen in `peth0`, `peth1`, usw. umbenannt. Zusätzlich werden von Xen standardmäßig zweimal je sieben Interfaces mit Namen `veth<vifid>` und `vif0.<vifid>` erstellt, wobei `<vifid>` dabei von 0 zu zählen beginnt. IP und MAC Adresse werden dabei dem `veth<vifid>` Interface zugewiesen, zur Konfiguration wird das `vif0.<vifid>` Interface verwendet [Pal]. Für jede NIC der virtuellen Maschine mit der laufenden Nummer `<domid>` wird ein virtuelles Interface `vif<domid>.<vifid>` in der `dom0` erstellt, wobei `<vifid>` die laufende Nummer der NICs dieser Domäne und `<domid>` dabei die Nummer der domU bezeichnet. `<domid>` beginnt dabei bei 1. Beispielsweise würde man die dritte NIC der zweiten domU über das Interface `vif2.2` in der dom0 ansprechen können.
Die virtuellen Netzwerkkarten der VMs kann man auf verschiedene Arten an das externe Netz anbinden. Die Interfaces können in eine "Netzwerkbrücke"(engl. *"bridge"*) eingebunden werden, sie können geroutet werden oder sie können in ein virtuelles Netzwerk ("VLAN") hinzugefügt werden.

Bridging

Eine Netzwerkbrücke verbindet NICs auf OSI Schicht 2. Bridges lassen sich auch durch Software realisieren. Dazu wird ein Bridge-Interface erstellt, an das die jeweiligen Interfaces gebunden werden.
Verbindet man beispielsweise das erste virtuelle Interface der ersten VM in der `dom0`, also

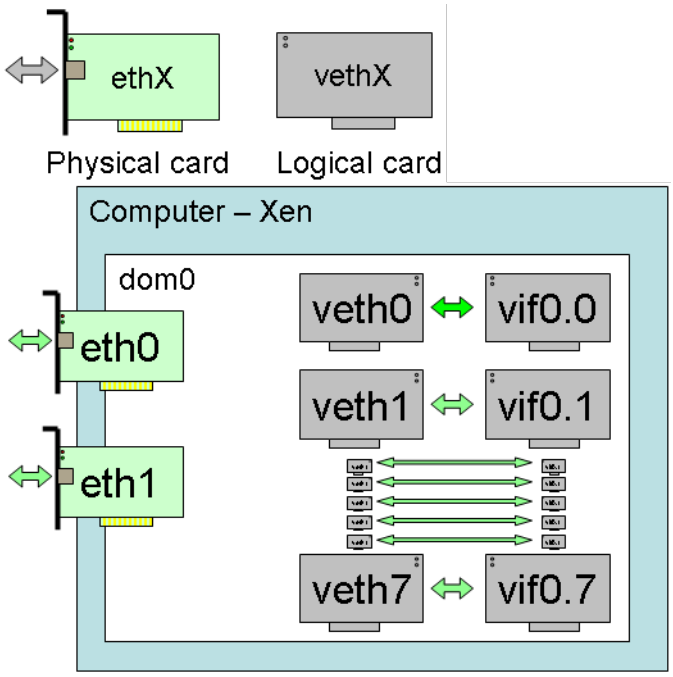


Abbildung 2.3.: virtuelle Interfaces in Xen [Pal]

`vif1.0` in einer Bridge mit dem Interface `peth0`, so hat man die VM mit dem Netz der ersten physischen NIC verbunden. Möchte man die `dom0` ebenfalls mit diesem Netz verbinden, gibt man einfach der Bridge selbst noch eine IP aus diesem Netz.
Für das automatische Erstellen einer solchen Bridge gibt es unter Xen das Skript `network-bridge`, das mittels einer Konfigurationsdatei angepasst werden kann. Xen verwendet hierbei das Linuxtool `brctl` aus dem Paket `network-bridge`. Möchte man nicht das Skript von Xen verwenden, kann man die Konfiguration auch direkt mit `brctl` durchführen.

Routing

Es ist ebenfalls möglich, die VMs mittels Routing zum physischen Netz zu verbinden. Das Verbinden erfolgt hier auf OSI-Schicht 3.
In der `dom0` muss dazu nur IP-Forwarding, das bedeutet die Routingfunktion von Linux aktiviert werden. Dies erfolgt durch das Eintragen einer 1 in `/proc/sys/net/ipv4/ip_forward` wenn das Netz auf IP Version 4 basiert. Diese Einstellung ist nach einem Neustart verloren. Alternativ kann auch fest in der Datei `/etc/sysctl.conf` die Zeile `net.ipv4.ip_forward=1` eingetragen werden. Bei Xen wird das allerdings automatisch vom Skript `network-route` übernommen.
Wird nun eine `domU` gestartet, muss die IP der NIC in der VM in das entsprechende `vif<domuid>.<vifid>` Interface in der `dom0` eingetragen werden und eine statische Route für die IP zu dem Interface `vif<domuid>.<vifid>` hinzugefügt werden. Diese Schritte lassen sich ebenfalls durch das Skript `network-route` automatisieren.

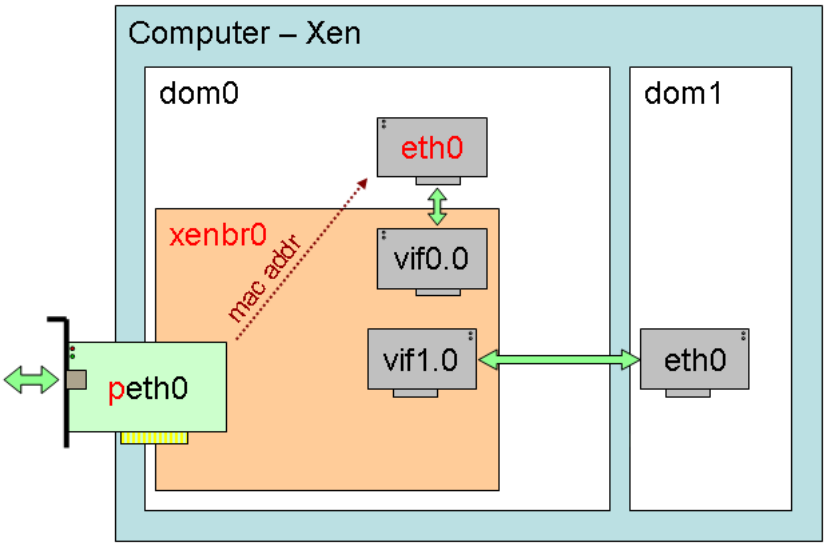


Abbildung 2.4.: Bridging in Xen [Pal]

VLAN

Zusätzlich gibt es noch die Möglichkeit alle `vif<domuid>.<vifid>` Interfaces, und damit alle virtuellen NICs der `domU`s in virtuelle Netze zusammenzufassen. Der Vorteil dieser Methode ist, dass man in der `domU` keine statische IP vergeben muss, sondern einen DHCP Client zur automatischen Vergabe der IP (und auch des Eintragens des DNS-Servers) starten kann. Trotzdem hat man noch abgesichert, dass keine `domU` den DHCP Server des physischen Netzes - den beispielsweise der Host verwendet - verwenden kann.

2.1.4. iSCSI

`iSCSI` ("Internet SCSI") ist ein Protokoll um das SCSI Protokoll - und damit Festplatten - über das Internet, oder in diesem Fall über LAN, also ein lokales Netz zur Verfügung zu stellen.
Dies wird ermöglicht, indem das SCSI Protokoll zum Festplattenzugriff über TCP, das heißt über ein verbindungsbasiertes Übertragungsprotokoll getunnelt wird.
Der Grund warum `iSCSI` in dieser Arbeit verwendet wird ist, dass dies die Standardmethode ist, um virtuellen Maschinen - beispielsweise in Rechenzentren - Festplattenspeicher zur Verfügung zu stellen. Das Szenario ist aus diesem Grund also viel praxisnaher.
Man baut dabei mit dem iSCSI Client, einem sogenannten "Initiator", der als Daemon in der `dom0` laufen muss, eine Verbindung zum Server, dem sogenannten "Target", auf. Normalerweise ist dabei eine Authentifizierung, zum Beispiel über das MSCHAP Verfahren nötig. Ist eine Verbindung zum Target hergestellt, werden die sog. "LUNs" ("logical unit number"), also der einzelne, dem Account mit dem man sich auf dem Target eingeloggt hat, zugewiesene Speicherbereich in das System eingebunden. Dieser Speicher wird dabei ganz normal, wie richtige Festplatten auf dem Rechner in das `/dev/` Dateisystem des Hosts eingebunden. Diese LUNs werden allerdings nicht direkt an die VMs übergeben, sondern es werden auf diese LUNs Festplattenimages - also Dateien auf die ein Dateisystem formatiert wurde - gelegt, welche an die VMs übergeben werden.

Der Grund hierfür ist eher ein organisatorischer als ein technischer. Im Praxisbetrieb wäre eine andere Methode sogar in der Regel nicht anwendbar. In der Praxis liegt die Verwaltung des iSCSI Targets nämlich meist nicht in der Hand derer, die die Virtualisierer verwalten. Außerdem ist eine Änderung der Anzahl und Größe der LUNs aufwendig und nicht praktikabel. Zu guter Letzt und dieser Grund alleine würde schon verhindern, dass man LUNs direkt an VMs übergibt muss nach einer Änderung der LUNs der Hostrechner sich vom iSCSI Target abmelden und erneut anmelden. Dieser *"Stop-the-world"-Vorgang* würde erzwingen alle virtuellen Festplatten aller virtuellen Maschinen zu entfernen, nur um die Größe einer zu ändern.
Da, wie bereits mehrere Male erwähnt, Xen auf Linux aufsetzt, ist es möglich bei der Konfiguration auf Standardprogramme unter Linux zurückzugreifen. Als iSCSI-Initiator kann daher das Paket `open-iscsi` verwendet werden.

3. Aufbau, Konfiguration und Anpassung

In diesem Kapitel wird die Hardware und die Software beschrieben. Es werden also die Testgeräte und die Infrastruktur erläutert und die verwendete Software, nativ und virtualisiert, aufgezeigt.

3.1. Hardware

Als Hostrechner wurde ein handelsüblicher PC mit einem Dualcore AMD Prozessor mit der Bezeichnung "Athlon X2 4800+" verwendet. Der Prozessor besitzt folglich zwei Kerne, wodurch wenn auch im geringen Maße Multicoreunterstützung von Xen überprüft werden kann. Die Taktung der CPU beträgt 2,4 Gigahertz.
Diese Dual-Core CPU besitzt pro Kern einen Level 1 Cache der Größe 64 Kilobyte und pro Kern einen Level 2 Cache der Größe 512 Kilobyte. Die Besonderheit ist hierbei, dass es nicht einen gemeinsamen Level 2 Cache für beide Kerne gibt, sondern zwei einzelne Caches gibt, die über separate Busse angesprochen werden können. Für den Arbeitsspeicher wurden vier DDR2 Speichermodule des Typs "PC2-6400" der Firma Samsung mit je einem Gigabyte Speicher und einer Taktung von 800MHz verwendet. Dabei sind je zwei Riegel per Dual-Channel verbunden. Dieses Prinzip bedeutet, dass der Speichercontroller des Mainboards zwei Hauptspeicher Steckplätze parallel betreiben kann. Dadurch sind weit höhere Datenraten als ohne dieses Prinzip möglich.
Bei diesem Verfahren sind allerdings separate Busse vom Speichercontroller zu allen Steckplätzen nötig, was sich stark auf die parallelen Tests auswirken kann, da bei parallelen Hauptspeicher Zugriff die einzelnen Busse unabhängig angesprochen werden können.
Die lokalen Festplatten des Hosts, auf dem das Betriebssystem und die Virtualisierungssoftware sowie die Imagedateien mit dem Betriebssystem der Gäste liegen (aber, wie in Kapitel 2.1.4 erwähnt, nicht die Images, die für die Festplattentests in Kapitel 4.1.1 verwendet werden), sind über S-ATA angeschlossen. Alle Testnetze (und damit das Netz für die Anbindung an das iSCSI Target sowie das dedizierte Netz für Zeitabfragen und Netztests, nicht aber beispielsweise das für Tests unerhebliche Managementnetz) sind über Gigabit Ethernet mittels CAT6 Kabel mit RJ45 Anschlüssen verbunden.

3.2. Infrastruktur

Zur Konfiguration der Testsoftware, als zuverlässiger Rechner zur Zeitgebung (gewährleistet durch ein natives System), aber auch für die zentrale Speicherung und Verwaltung der Testergebnisse und als Gateway in das öffentliche Netz und zentraler Rechner für den Remotezugriff auf Hosts und Gäste wurde ein Managementrechner eingerichtet. (Abbildung 3.1)

Auf diesem Managementrechner wurden als dualboot Windows XP und Ubuntu 8.10 ("Intrepid Ibex") installiert, da manche Konfigurationstools der Benchmarks ein Windows bzw.

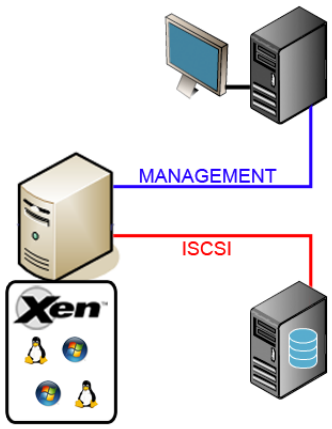


Abbildung 3.1.: Hardware Infrastruktur

ein Linuxsystem benötigen.
Zusätzlich wurde diesem Rechner eine dedizierte Leitung gelegt, über die die Tests des Netzes oder die Zeitabfragen durchgeführt werden. Über ein extra Netz, dem SAN ("Storage Area Network") ist der Hostrechner zusätzlich noch mit dem iSCSI verbunden.

3.3. Konfiguration des Hostsystems

Wie in Kap. 2.1 erläutert, wird als Virtualisierer das freie Xen von `http://www.xen.org`, basierend auf Linux verwendet. Als Linuxdistribution wurde das verbreitete und gut supportete Ubuntu (`http://www.ubuntu.com`) in der Version 9.04 ("Jaunty Jackalope") verwendet. Xen liegt hier in der Version 3.3 vor [ubuc] und kann mit Hilfe des Meta-Pakets "ubuntu-xen-server" gleich direkt mit allen benötigten Zusatzprogrammen installiert werden [ubub]. Einzig der angepasste Kernel, der die `dom0` zur Verfügung stellt (siehe Kap. 2.1), muss extra installiert werden. Allerdings liegt in den Paketquellen von Ubuntu Jaunty ein solcher vorkompilierter Kernel nicht vor, weswegen ein Kernel der Version 2.6.24 aus den Quellen der Vorvorgängerversion 8.04 "Hardy Heron" installiert wurde [ubua].

3.4. Native Systeme

Wie in den vorherigen Kapiteln unter der Hand schon erwähnt, werden als Gäste einerseits Linux- aber auch Windowssysteme getestet. Um die virtuellen Systeme mit den nativen vergleichen zu können, wurden alle Systeme mit der gleichen Version und Konfiguration auch nativ installiert. Konkret waren es in diesem Fall Ubuntu Server 9.04 und Windows Server 2003 SP1 RC2 jeweils in 32 und 64 Bit.

3.5. Virtualisierte Systeme

Die Imagedatei, auf denen das Betriebssystem der domUs liegen wird, wurde mit den Linuxtool `dd` ("Disk Dump") erstellt. Mit diesem Tool ist es möglich Dateien, aber auch ganze

Partitionen oder Festplatten zu kopieren. Dabei wurde sich die Datei `/dev/zero`, die auf jedem Linuxsystem im virtuellen Dateisystem unter `/dev` liegt, zu Nutze gemacht. Virtuelles Dateisystem deshalb, weil darin keine tatsächlichen Dateien enthalten sind, die auf der Festplatte liegen, und auf die nach dem Herunterfahren des Rechners noch tatsächlich auf der Festplatte existieren. Stattdessen wirken sie für den Benutzer und für Programme, die auf sie zugreifen wie Dateien, werden aber vom Linuxkernel angeboten. Bei der Datei `/dev/zero` im Speziellen, handelt es sich um eine Datei die theoretisch unendlich groß ist, und bei der jedes Bit 0 ist. Diese Datei wird verwendet, wenn man Informationen benötigt, die Informationen selbst aber irrelevant sind. Mit dem Befehl `dd if=/dev/zero of=virt1.img count=6144 bs=1M` erstellt man daher eine sechs Gigabyte große, leere Datei die als virtuelle Festplatte verwendet werden kann. Zusätzlich sei noch erwähnt, dass in der Praxis eher eine dynamisch wachsende Datei mit Hilfe des Befehls `dd if=/dev/zero of=virt1.img count=0 bs=1 seek=6G` erstellt werden würde. Dieses Image sieht für die VM ebenfalls wie eine sechs Gigabyte große Datei aus, belegt auf der Festplatte des Hosts aber zu Beginn nur den tatsächlich benötigten Platz. Der Vorteil hierbei liegt im Geschwindigkeitszuwachs beim Erstellen - nahezu keine Wartezeit im Vergleich zu mehreren Minuten je nach Festplatte - und dem möglicherweise anfangs eingespartem Speicherplatz.
Auf dieses Verfahren wurde in dieser Arbeit bewusst verzichtet, damit sichergestellt war, dass durch das nachträglich durchgeführte Anwachsen der Datei die Ergebnisse nicht verfälscht wurden. Da auf den Dateisystemen der VMs außer dem Betriebssystem selbst und den unerheblich kleinen Benchmarktools keine Dateien abgelegt werden müssen, kann das Image klein gehalten werden, 5-10GB reichen aber je nach Betriebssystem in der Regel sicher aus.
Als Arbeitsspeicher wurde jeweils 1GB vergeben. Einerseits ist ein Gigabyte - wie später unter 3.6.2 erwähnt - ausreichend für alle Testgrößen, andererseits kann so eine Situation geschaffen werden, mit der in den späteren Tests auch noch mehrere VMs gleichzeitig getestet werden können, ohne dass der Hypervisor oder die dom0 nicht mehr genügend Arbeitsspeicher zu Verfügung haben.
In den VMs wurde darauf geachtet keine Auslagerungsdatei (oder englisch "swap file“) anzulegen, da sonst bei voller Benutzung des Arbeitsspeichers Daten auf die Festplatte gespeichert (“ausgelagert”) worden wären, was die Testergebnisse verfälschen würde.
Eingetragen wird die Größe des RAMs durch die Zeile `memory = '1024'` in die Konfigurationsdatei der VM.
Damit die Parallelisierungsfähigkeit der CPU durch den Hypervisors getestet werden konnte, wurden jeder VM nur eine vCPU zugewiesen. Da der Hostrechner zwei Kerne besitzt, müsste die Leistung bei CPU Tests daher bei zwei VMs jeweils nahezu identisch zu nur einer laufenden VM sein. Ob dies zutraf, konnte durch diese Konfiguration getestet werden.
Für den Zugriff auf das physische Netz des Hosts wurden zwei Interfaces in den domUs eingerichtet, die mittels Bridging wie in Kap. 2.1.3 beschrieben eingerichtet wurden. Die erste NIC wurde dabei dem Managementnetz, die zweite dem dedizierten Netz hinzugefügt.
Die Anbindung zu dem iSCSI Netz muss nicht direkt in den domUs verfügbar sein, da die LUN in der privilegierten Domäne eingebunden wurde, und ein Image der Größe 5 Gigabyte auf dieser LUN mittels `dd` erstellt und der domU übergeben wurde.

vollvirtualisiertes Linux

Zur Installation der vollvirtualisierten Linuxgäste wurde der `vnc` (“virtual network computing”) Server der dom0 aktiviert. Dies geschieht durch ein einfaches Einfügen der Zeile `vnc`

= 1 in die Config der VM. Möchte man parallel mehrere VNC Server zu unterschiedlichen domUs aufmachen, ist das durch hinzufügen einer ID und eines Ports möglich. Mittels dieses Dienstes ist es möglich sich remote zu der Grafikoberfläche eines Gastes zu verbinden, um dort wie bei einem nativen System die Installationsroutine auszuführen. Die Installation an sich verläuft also identisch zu der Installation eines nativen Systems. Als Kernel wird dabei der Standardkernel der verwendeten Ubuntuversion 9.04 Kernelversion 2.6.28-11-server installiert. Bei der Installation wird dabei ein Master Boot Record (mbr) und eine Partitionstabelle mit einer Partition - im Falle von Linux ohne Swap-Partition - angelegt. Anpassungen der Installation können mittels `chroot` durchgeführt werden, wenn man die Partition der virtuellen Festplatte mittels des Parameters `offset=<offset>*512` beim Befehl `mount` angibt. Den Wert des Offsets erhält man durch den Aufruf `fdisk -l -u <image>`.
Diese Festplatte der VM wird - genau wie die NICs - emuliert. In der Konfigurationsdatei geschieht das über den Zusatz `ioemu`. Die entsprechenden Zeilen in der Config könnten also wie folgt aussehen:

```
 kernel =  /usr/lib/xen/boot/hvmloader
builder =  hvm
   disk =  [ 'file:/pfad/zum/domU.img,ioemu:hda,w' ,
           'file:/mnt/iscsi.img,ioemu:hdb,w' ]
vif =  ['type=ioemu, mac=[...], bridge=xenbridge',
       'type=ioemu,mac=[...],bridge=xenbridge2']
```

Die ersten drei Byte der MAC, die den Hersteller der Netzwerkkarte angeben, sollten `00:16:3e` sein, da dieser Bereich für Xen reserviert ist. Die restlichen Byte sind beliebig, müssen aber innerhalb des LANs eindeutig sein. Durch die Angabe `bridge=...` werden die Interfaces automatisch der jeweiligen Bridge hinzugefügt, diese wird vorher erstellt. Möchte man von der dom0 aus auch auf das Netz zugreifen, muss der Bridge selbst noch eine IP vergeben werden, dies ist zum Beispiel über einen Eintrag in die `/etc/network/interfaces` möglich.

paravirtualisiertes Linux

Die Installation der paravirtualisierten Linux Gäste verläuft etwas anders.
Wie vorher bereits erwähnt, liegen auf den Images dieser Gäste keine ganzen virtuellen Festplatten mit Partitionstabelle, sondern lediglich die einzelnen verwendeten Partitionen. Die Installation kann also nicht durch die Installationsroutine erfolgen. Stattdessen wird das Basissystem mittels `debootstrap` aus den Ubuntuquellen geladen. Mit Hilfe dieses Programms kann man ein Minimalsystem aus einem APT Repository geladen werden. Als Parameter werden noch die gewünschte Distribution, in diesem Fall Jaunty, und die gewünschte Architektur, also 32 sowie 64 Bit angegeben. Ein Kernel oder ein Bootloader wie grub ist dabei nicht enthalten, wird allerdings auch nicht benötigt. Stattdessen wird in der Config der VM der zu verwendende Kernel eingetragen. Dieser muss in der dom0 liegen. Da das Hostsystem ebenfalls auf Ubuntu Jaunty basiert, ist der Standard Serverkernel bereits installiert und wird aus Gründen der Vergleichbarkeit mit der vollvirtualisierten Variante verwendet. Es wäre theoretisch aber auch möglich direkt den gebooteten Xen Kernel zu verwenden. Es wird angenommen der Kernel und die dazugehörige initramfs Datei liegen wie üblich unter `/boot`. Die Partition nun nicht paravirtualisiert eingebunden wird, fehlt auch der Zusatz `ioemu`, der Eintrag in der Konfigurationsdatei könnte also wie folgt aussehen:

```
 kernel =  '/boot/vmlinuz-2.6.28-11-server'
ramdisk =  '/boot/initrd.img-2.6.28-11-server'
   disk =  [ 'file:/pfad/zum/domU.img,xvda1,w' ,
             'file:/mnt/iscsi.img,xvda2,w' ]
    vif =  ['mac=[...], bridge=xenbridge',
             'mac=[...],bridge=xenbridge2']
```

vollvirtualisiertes Windows

Die Installation eines vollvirtualisierten Windows Gastes erfolgt im Prinzip identisch zu der Installation eines Linux Gastes wie in Kapitel 3.5 beschrieben. Es wird erst mittels `dd` ein Image erstellt, dieses über die Konfigurationsdatei der VM zugewiesen, und VNC aktiviert (3.2). Dann wird über den VNC Server eine normale Installation vorgenommen. Nach der Einrichtung des Netzes für den Gast wird allerdings im Gast der RDP Server aktiviert, da eine Konfiguration und Handhabung über VNC meist sehr unpraktikabel ist. Über RDP hingegen ist eine sehr hohe Performanz und geringe Latenz erreichbar.

Windows mit paravirtualisierten Treibern

Die Einrichtung eines Windows Gastes mit paravirtuellen Treibern basiert auf der eines vollvirtualisierten Gastes. Dabei kann ein vollvirtualisierter Gast als Grundlage verwendet werden, sprich ein solches Images (vor der Aktivierung von Windows) kopiert und angepasst werden.
Danach werden die paravirtuellen Treiber von [Mea] für das entsprechende System und Architektur geladen und installiert. Der Ausdruck "`wnet`" bezeichnet dabei die Version für Windows 2003. Nach einem Reboot stehen die Treiber für Netzwerkkarten, Festplatte und Grafikkarte zur Verfügung. Dabei ist darauf zu achten, dass die virtuellen Interfaces der vollvirtualisierten Version, die während der Installation noch zur Verfügung stehen, unkonfiguriert sind. Die Installation muss also über eine Verbindung zum VNC Server erfolgen.
Unter Umständen muss dabei noch eine Anpassung der Registry des Gastes erfolgen, dem Schlüssel `DisableTaskOffload` unter
`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters`
muss dabei noch der Wert 1 zugewiesen werden. Möglicherweise muss dieser Schlüssel vom Typ `DWORD` erst erstellt werden [MSD].
Dadurch wird der so genannte "Task Offload" (in etwa "Weiterleitung von Aufgaben") von prozessorintensiven Aufgaben an die Netzwerkkarte deaktiviert. Normalerweise werden unter Windows Aufgaben wie das Berechnen der TCP/IP Prüfsumme oder der Größe der Paketfragemente an den Netz-Adapter weitergereicht und nicht von der CPU berechnet. Durch das Setzen des Wertes in der Registry wird diese Funktion deaktiviert.
Im Rahmen dieser Arbeit war diese Anpassung nur bei Windows Server 2003 Gästen in der 32 Bit Variante nötig.

3.6. Benchmarktools

An die verschiedenen Programme, mit denen die virtuellen Maschinen letztendlich getestet wurden, wurden verschiedene Anforderungen gestellt. Eigenschaften, die alle Programme erfüllen mussten, waren, dass sie in einer Version für alle verwendeten Systeme zur

Verfügung standen, freie Software waren, also unter der GPL ("GNU Public License") oder einer ähnlichen freien Lizenz standen. Der Grund hierfür liegt der Annahme zu Grunde, dass dies einen erhöhten Grad an Nachvollziehbarkeit schafft, da ersichtlich wird, welche Operationen genau zum Testen durchgeführt werden, und damit keine zusätzlichen Kosten für die Durchführung anfallen.
Wie bereits in 2.0.5 beschrieben, müssen die Benchmarktools an die virtuellen Maschinen und deren Zeitgebung angepasst werden. Eventuelle Probleme bei manchen Tools bei der Anpassung an die 32 bzw. 64 Bit Versionen der Windows und Linux Systeme werden im Folgenden behandelt.

3.6.1. CPU

Beim CPU Benchmark fiel die Wahl auf den sehr bekannten Benchmark LINPACK.

LINPACK ist eigentlich eine Programmbibliothek zur Berechnung von linearen Gleichungssystemen. Ursprünglich wurde es in Fortran geschrieben, um damit Supercomputer in den 70er und 80er Jahren zu testen. Mittlerweile ist es jedoch nicht nur eine Programmbibliothek, sondern wird unter anderem dafür verwendet, CPU Leistungen miteinander zu vergleichen, hierfür sind auch verschiedenste Versionen in sehr vielen gängigen Programmiersprachen vorhanden, wie z.B. die hier verwendete Version in C. Der LINPACK-Benchmark misst die Geschwindigkeit der CPU in FLOPS. FLOPS sind "floating point operarations per second", also die Gleitkommaoperationen pro Sekunde. Als Gleitkommaoperationen bezeichnet man die Befehle und Berechnungen einer CPU unter Verwendung von Gleitkommazahlen. Dieser Wert gibt wider, wie schnell die CPU ein dichtes $N \times N$ lineares Gleichungssystem mit linearen Gleichungen $Ax = b$ berechnet. Solche Berechnungen werden häufig in komplexen Systemen verwendet und eignen sich daher gut für eine Geschwindigkeitsmessung. Diese Matrix wird mit der "Gaussian Elimination with partial pivoting" gelöst. LINPACK bedient sich hierzu der BLAS ("Basic Linear Algebra Subprograms") Bibliothek. In unserem Fall haben wir die Systeme jedoch nicht anhand der FLOPS, sondern nur mit der Berechnungszeit verglichen, aus welcher sich die FLOPS problemlos berechnen lassen. Der Grund hierfür ist, dass die Angabe an erreichten FLOPS durch die Round Trip Time des dedizierten Netzes beim Abfragen der Systemzeit vom Zeitserver verfälscht wird. Eine Angabe in FLOPS würde suggerieren, dass der Wert mit dem normalen LINPACK vergleichbar wäre, was nicht gewährleistet werden kann.

Anpassung

Wie bereits in Kapitel 2.0.5 beschrieben, ist die Zeitgebung in einer virtuellen Maschine unter Last nicht zwingend korrekt. Da der LINPACK-Benchmark eine sehr hohe Last erzeugt, war hier eine Anpassung des Tools notwendig. Der Standard-LINPACK wurde in eine Server/Client-Architektur umgebaut, um somit eine korrekte Zeitgebung in einer VM über einen Zeit-Server auf einem nativen System zu erreichen. Für Linux wurden die Daten per TCP übertragen, welches jedoch unter Windows manchmal zu Verzögerungen und dadurch Ungenauigkeiten führt. Deshalb wurde für Windows noch eine UDP-Version des Servers erstellt, eine Anpassung der Sockets war für diese Version sowieso notwendig. Auf Client Seite war die Anpassung des LINPACK etwas komplizierter, da der Quelltext selbst, damit eine korrekte Berechnung der FLOPS gewährleistet ist, nur an ein paar Stellen verändert werden darf.

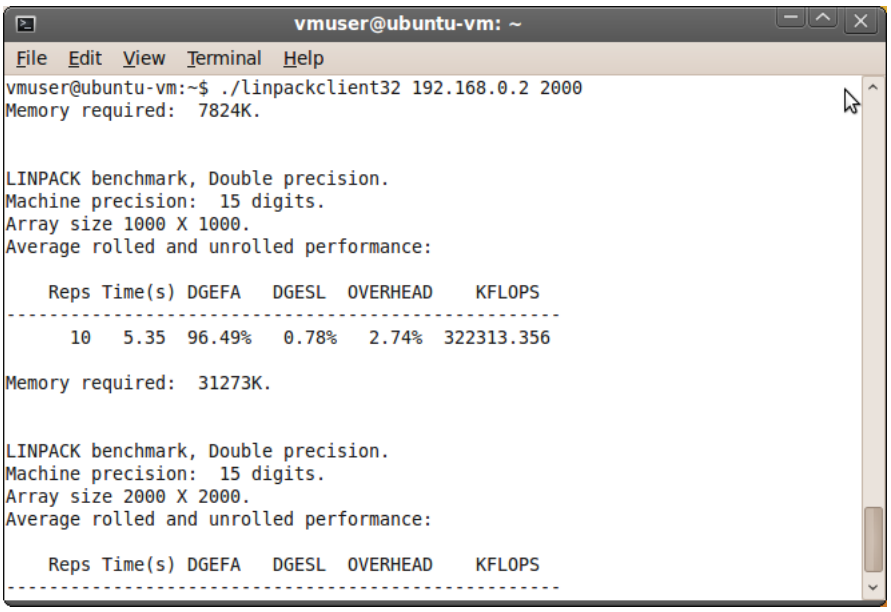


Abbildung 3.2.: Ausgabe von LINPACK

Beim Programmstart wird eine Socket erstellt und eine Verbindung mit dem Server hergestellt. Die für die Zeitgebung implementierte Funktion second() wurde um die Übertragung der Zeit über die Sockets erweitert. Des Weiteren wurde das Grundgerüst des Benchmarks leicht modifiziert. Die Eingabe der Matrizengröße wurde gelöscht und in einer Schleife die Berechnungen der Größen 1000, 2000, 4000, 8000 festgelegt. Diese Matrizen sollten jeweils 10 Mal berechnet werden. Jedoch sollten nach maximal 20 Minuten die Berechnungen nicht mehr weitergeführt werden, um somit eine sehr lange Dauer der Tests zu vermeiden.

3.6.2. RAM

Bei dem Benchmark RAMspeed handelt es sich ebenfalls um ein freies Programm. Es steht unter einer eigenen, der GPL ähnlichen Lizenz [RMH] [Ent].
Obwohl durch den Namen suggeriert, handelt es sich bei RAMspeed nicht um ein reines Benchmarkprogramm, um den Hauptspeicher zu testen. Es gibt unterschiedliche Tests, die den Cache der CPU, den Hauptspeicher, aber auch die ALU ("arithmetic logic unit", deutsch "Arithmetisch-logische Einheit") und die FPU ("Floating Point Processing Unit", deutsch "Gleitkommaeinheit") der CPU testen kann. RAMspeed besteht aus zwei Hauptkomponenten.

- `INTmark` und `FLOATmark`. Mit Hilfe dieser Tests wird die maximal mögliche Cache- und Hauptspeichergeschwindigkeit ermittelt. Dazu werden Daten in der Größe 2^x KB $(x \in 0, \ldots, 12)$, also 1KB bis 4MB, blockweise gelesen bzw. geschrieben. Es wird nicht ständig zwischen Lese- und Schreiboperationen gewechselt, sondern beide unabhängig voneinander, nacheinander durchgeführt. `FLOATmark` spricht dabei die erwähnte FPU an.
- `INTmem` und `FLOATmem`. Diese Tests bestehen jeweils aus vier Untertests mit Namen

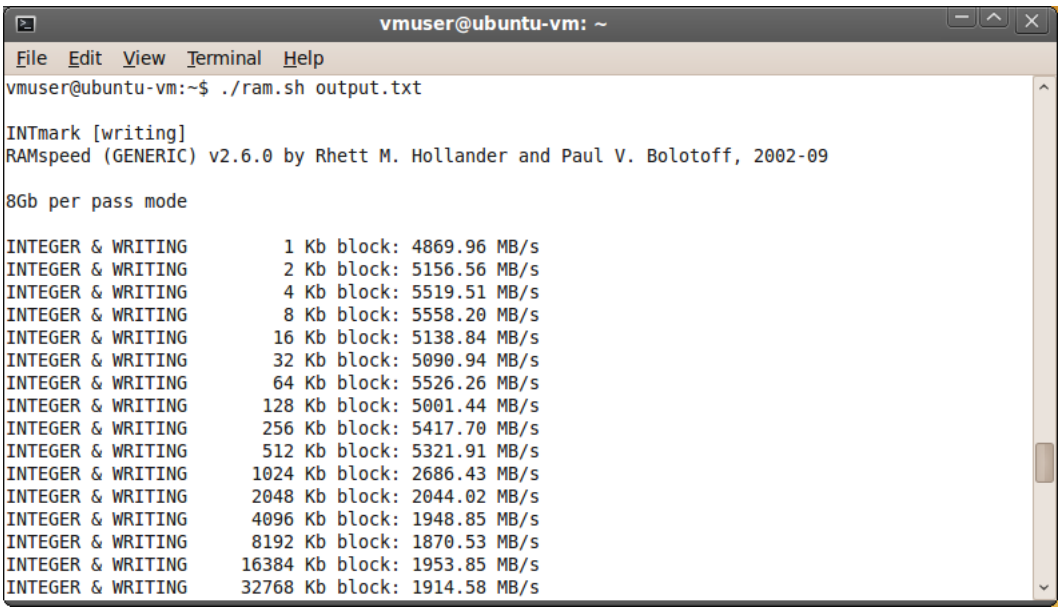


Abbildung 3.3.: Ausgabe von RAMspeed

`Copy`, `Scale`, `Add` und `Triad`. Bei diesen Tests handelt es sich zwar um synthetische Tests, allerdings soll dabei so gut wie möglich ein Realitätsbezug hergestellt werden. Bei `Copy` wird ein Bereich aus dem Hauptspeicher gelesen und an einem anderen Platz geschrieben (mathematisch: $A = B$). Bei `Scale` ebenso, nur wird der Wert vorher mit einer Konstante multipliziert (mathematisch: $A = m \cdot B$). Bei `Add` werden zwei Bereiche gelesen, diese addiert und an einen dritten Bereich gespeichert (mathematisch: $A = B + C$). Der letzte Untertest `Triad` ist eine Kombination aus `Add` und `Scale`, also mathematisch $A = m \cdot B + C$. `INTmem` zielt also allein darauf ab, die ALU zu testen. Bei `FLOATmem` handelt es sich dann um eine Kombination aus einem FPU und einem ALU Test.

Anpassung

Die Zeitmessung in virtuellen Maschinen ist wie unter 2.0.5 erwähnt vor allem bei kurzen Intervallen ungenau. Ist das Intervall ausreichend lang, sind korrekte Ergebnisse zu erwarten. RAMspeed führt im Gegensatz zu LINPACK keine hochfrequenten Zeitabfragen durch, sondern misst die Zeit nur vor und nach dem Test. Dadurch war eine Anpassung des Benchmarktools nicht nötig.
Zur einfacheren Durchführung der Tests wurde daher lediglich ein Bash-Skript unter Linux, bzw. eine Batch-Datei unter Windows erstellt, mittels der die gewünschten Tests der Reihe nach durchgeführt und in eine Ausgabedatei umgelenkt wurden. Damit wurde sichergestellt, dass die Ausgabe auch unmittelbar in die Datei geschrieben wurde und nicht im Hauptspeicher bleibt, wurde danach ein `sync` durchgeführt. Damit wurde sichergestellt dass die Ausgabe den Test nicht beeinflussen kann. Unter Linux ist das Programm `sync` bereits vorhanden, unter Windows kann es von Sysinternals [Sysb] heruntergeladen werden [Sysa].

3.6.3. Netz und Disk

Bei den Netzwerk und Festplatten Benchmarks fiel die Wahl auf Iometer, weil dieses Tool die Anforderungen am Besten erfüllte. Iometer ist ein Input/Output Messsystem. Es misst die Performanz unter einer kontrollierten Last und es besteht aus einem Lastgenerator, welcher die I/O Operationen erzeugt, und einem Messwerkzeug. Dieses untersucht die Performanz und zeichnet diese und die Auswirkung auf das System auf. Es kann soweit konfiguriert werden, dass es mehrere verschiedene Szenarien simuliert. Verschiedene Testszenarien mit Festplatten und Netzwerkzugriffen sind hiermit möglich. Auch reine synthetische Benchmarks können durchgeführt werden. Folgende Tests sind denkbar:

- Geschwindikeitstests der Festplatten- und Netzwerkcontroller
- Bandbreite und Latenzverhalten von Bussen.
- Netzwerkdurchsatz der angeschlossenen Karten
- Shared-Bus Performanz.
- System-Level Festplatten-Performanz
- System-Level Netzwerk Performanz.

Iometer besteht aus zwei Programmen, Iometer und Dynamo.
Iometer ist die Kontrolleinheit des Tools. Es besitzt eine graphische Oberfläche mit der man der Last und die Parameter der Tests steuern kann. Iometer teilt einem oder mehreren Dynamos mit, was diese zu tun haben und wertet die gewonnenen Daten aus. Normalerweise läuft immer nur ein Iometer auf einem Server zu dem sich die einzelnen Dynamos verbinden. Der Dynamo generiert die Last. Es besitzt keine Benutzerschnittstelle. Er erledigt und generiert lediglich die Last und erzeugt die Input und Output Operationen für das Netzwerk oder die Festplatten. Es können auch mehrere Dynamos gleichzeitig auf einem System laufen, welche unterschiedliche Lasten erzeugen. Jede laufende Version von Dynamo wird auch als “Manager” bezeichnet, jeder darin enthaltene Thread als “Worker”.

Anpassung

Iometer hat bereits eine Server/Client Architektur, weshalb die Zeitgebung in diesem Fall keine Probleme darstellt. Jedoch hatte die verwendete Version einen Bug. Der dynamo wurde gepatched und musste für die verschiedenen Systeme, Windows und Linux - jeweils 32 und 64 Bit, neu kompiliert werden. Das Problem war, dass nach Ablauf des Tests Iometer, bei einer Verbindung zwischen zwei dynamo Programmen, einen Stopbefehl an beide dynamos schickte. Dieser Befehl kommt aber in der Regel nicht bei beiden gleichzeitig an, worauf sich ein dynamo früher beendet. Das hat zur Folge dass der zweite eine Exception warf und auf Stop-Befehle von der iometer.exe nicht mehr reagierte [pcd].

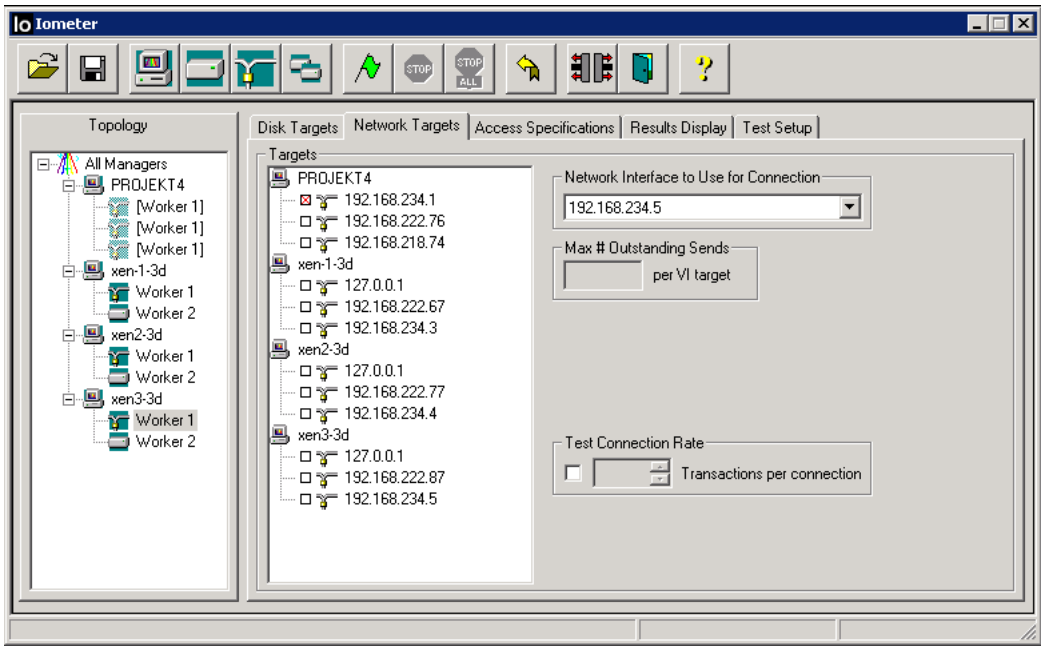


Abbildung 3.4.: GUI des Benchmarks Iometer

4. Durchführung und Auswertung der Tests

Als Erstes wurde mit Hilfe der synthetischen Benchmarks der Wirkungsgrad der Virtualisierung festgestellt. In Folge dessen wurde eine optimale Konfiguration gewählt, und auf dieser die Skalierungsbenchmarks durchgeführt.

4.1. Wirkungsgrad Benchmarks

Die ersten Tests, die in den virtuellen Maschinen durchgeführt wurden, sind die synthetischen Benchmarks. "Synthetisch" bedeutet in diesem Fall, dass eine künstliche Situation geschaffen wird, die in der Praxis nie auftreten würde. Unter diesen Voraussetzungen können alle einzelnen Hardwarekomponenten der VM gesondert untersucht werden, ohne dass sie sich gegenseitig beeinflussen. So wird während der Tests nur eine einzige VM auf einem Virtualisierer gestartet, und auch nur ein Benchmark gestartet, der nur eine Komponente testet. Die ermittelten Werten werden mit den ermittelten Werten bei nativen System unter gleichen Voraussetzungen verglichen. Interessant ist dabei der Wirkungsgrad der Virtualisierung. Das bedeutet das Verhältnis der Leistung der virtuellen Maschine durch die Leistung der nativen (= "Overhead"). Dadurch kann der Verlust berechnet werden, der durch die Virtualisierung der Hardware entsteht.

4.1.1. Durchführung

Bei den Durchführungen wurde über alle möglichen Konfigurationen iteriert.
Als Gastbetriebssystem wurde wie erwähnt Windows und Linux gewählt. Speziell wurden Windows Server 2003 und Ubuntu Linux 9.04 gewählt. Außerdem wurden beide Systeme jeweils in 32 und 64-Bit installiert. Linux dabei jeweils einmal voll- und einmal paravirtualisiert. Bei Windows ist dies durch das proprietäre System nicht möglich, daher wurden die paravirtualisierten Treiber installiert bzw. weggelassen.
Des Weiteren wurde das in Kapitel 2.0.4 angesprochene AMD-V bei paravirtualisiertem Linux an- und ausgeschaltet. Bei vollvirtualisierten Systemen ist AMD-V für den Einsatz notwendig
Auf all diesen Systemen wurden dann alle vier Benchmarks durchgeführt.

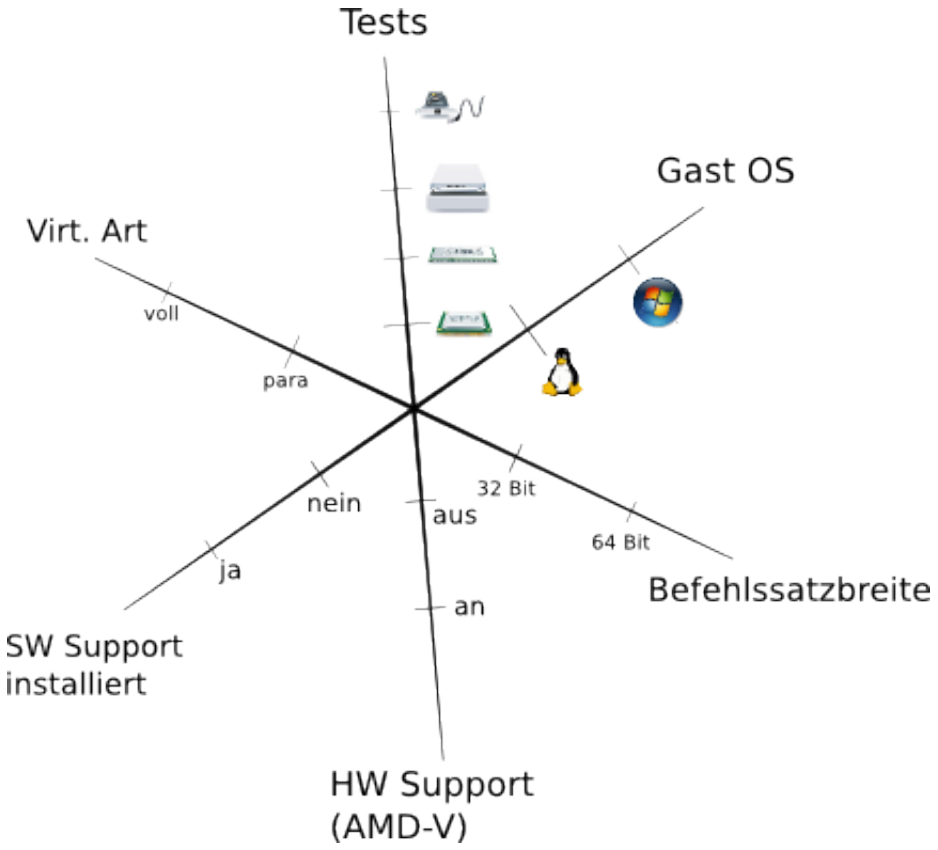


Abbildung 4.1.: Alle Dimensionen, über denen iteriert wurde

Linpack

Für die CPU Tests wurde ein eigens erstellter Linpack Server auf dem Managementrechner gestartet, als Parameter bekam dieser Server einen Port, auf dem er auf zu verbindende Clients lauschen sollte. Die Windows Variante bekam zusätzlich noch eine Host IP, um den Server fest an eine NIC zu binden. Die Befehle zur Ausführung waren also:
`./linpackserver <port>`
bzw.
`linpackserver <hostip> <port>`
Über SSH oder die Xen Konsole bei Linuxgästen bzw. RDP bei Windowsgästen hat man sich zu den VMs verbunden und in der Konsole den Client gestartet. Als Parameter bekam der Client IP und Port des Linpackservers:
`./linpackclient32 <hostip> <port>`
bzw.
`./linpackclient64 <hostip> <port>`
oder
`linpackclient <hostip> <port>`
Linpack benutzt quadratische Matrizen für die Berechnung. Als Größen wurden 1000, 2000, 4000 und 8000 fest in den linpack einprogrammiert. Gemessen wurde jeweils die Zeit die für 10 Durchläufe benötigt wurde.

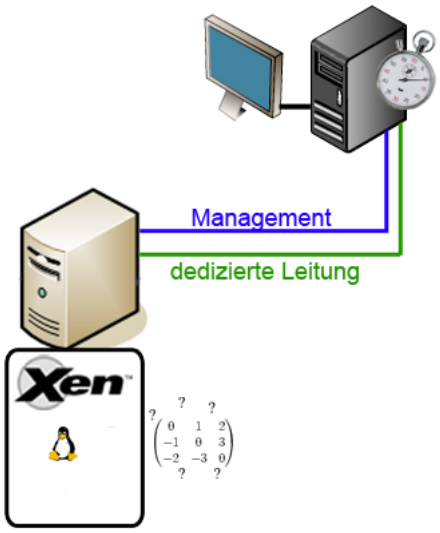


Abbildung 4.2.: Aufbau für den CPU Test

RAMspeed

Die einzelnen Speichertests die `RAMspeed` unterstützt, können über Parameter angegeben werden. Dabei wird `RAMspeed` eine Zahl mitgegeben, die einen bestimmten Test repräsentiert, z.B.: `./ramspeed -1`, `./ramspeed -2` usw.
Mehrere dieser Tests wurden in einem einfachen Bash-Skript bzw. einer Batch-Datei zusammengefasst. Das Ergebnis wurde jeweils in eine Ausgabedatei umgelenkt. Damit sichergestellt ist, dass die Ausgabe auch tatsächlich auf die Festplatte geschrieben wurde, und nicht noch im Hauptspeicher liegt - was den Test (wenn auch nur geringfügig) beeinflussen könnte - wurde nach jedem Test ein `sync` ausgeführt. Unter Linux ist dieser Befehl standardmäßig vorhanden. Für Windows gibt es von Sysinternals [Sysb] ein entsprechendes Programm.

Iometer (Netz)

Iometer besitzt wie bereits erwähnt eine leicht zu handhabende grafische Oberfläche. Das Programm `iometer.exe`, welches nur für Windows erhältlich ist, ist gleichzeitig externer Zeitgeber und "Ort der Berechnung". Dieses Programm läuft auf dem Managementrechner. Der sogenannte `dynamo`, der die Last erzeugt, wird in der VM gestartet und verbindet sich zur Client-GUI. Auf dem Managementrechner wird über die GUI ein zweiter `dynamo` gestartet, der den Endpunkt der Netzwerkverbindung darstellt. Als Testnetz wird das dedizierte Netz gewählt, damit die Übertragungen des Benchmarktools selbst den Test nicht beeinflussen. Hierzu werden in der GUI die NICs dieses Netzes als Quelle und Senke gewählt.
Der Aufruf für den dynamo ist dabei `./dynamo -m <IP der VM> -i <GUI IP> -n <name>`. Bei der `<GUI IP>` handelt es sich dabei um die IP des Managementrechners, auf der die `iometer.exe` läuft. `<name>` ist ein beliebiger Text zur einfacheren Identifikation des Tests in der Ausgabedatei. Für die Tests wurde ein einheitlicher Ablauf festgelegt und in der GUI eingerichtet. Diese Konfiguration lässt sich als `.icf` Datei abspeichern. Es wurden sowohl Lese- als auch Schreibeoperationen durchgeführt. Jeweils wurden Pakete in den Größen 4, 32 und 256 Kilobyte und 1 und 4 Megabyte verschickt und empfangen. Jede dieser Testgrößen

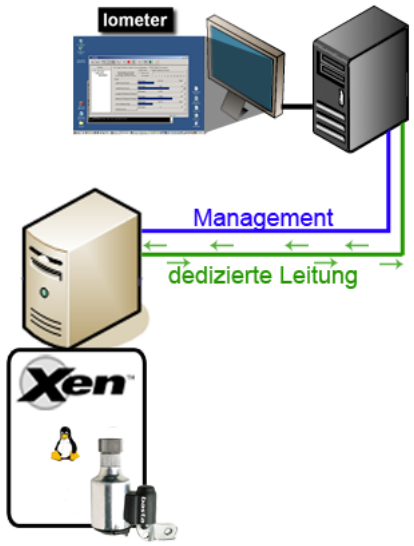


Abbildung 4.3.: Aufbau für den Netz Test

wird drei Minuten getestet und von Iometer gemittelt (arithmetisches Mittel). Die Lese- und Schreibetests laufen also jeweils 15 Minuten und werden beide in einer `.csv` ("Comma separated value") Datei gespeichert.

Iometer (Disk)

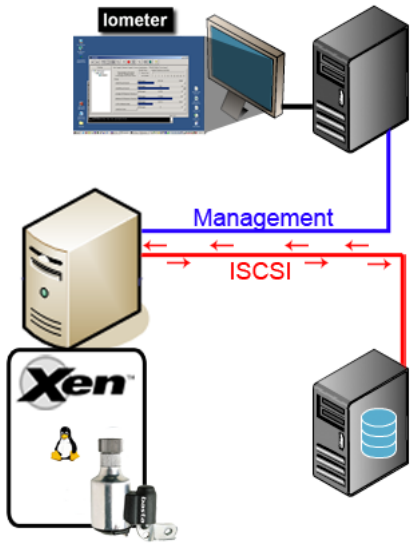


Abbildung 4.4.: Aufbau für den Disk Test

Für den Festplattentest wird, wie schon erwähnt ein Image das auf einer iSCSI Lun liegt, benötigt. (siehe Kapitel 2.1.4) Dieses Image wird normal wie das normale Festplattenimage der VM eingebunden.
Auf der VM wird wie beim Netztest 4.1.1 ein dynamo gestartet, der sich zur iometer.exe

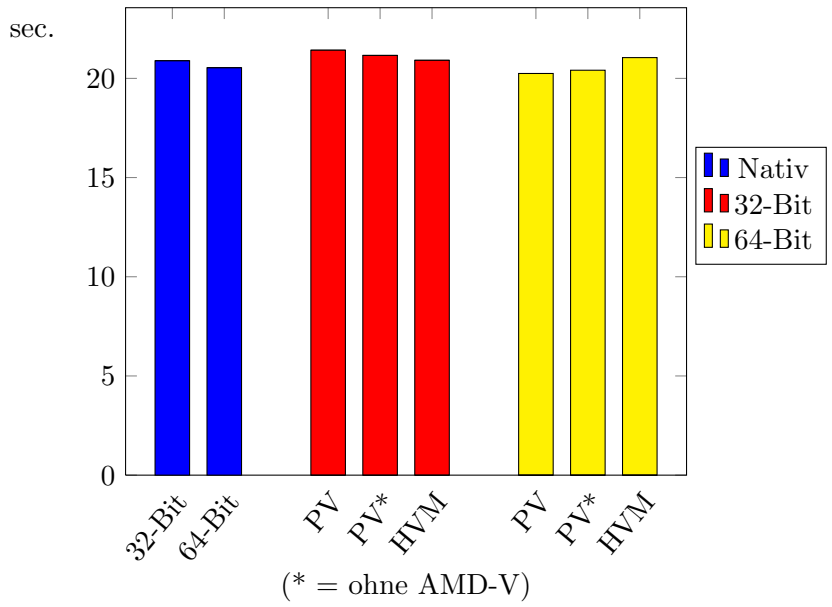
verbindet.
Für diesen dynamo wird in der GUI dann der Disktest ausgewählt und als Ziel die ganze Testpartition ausgewählt. Alternativ kann auch auf dem Image ein Dateisystem erstellt werden und dieses als Ziel gewählt werden.
Wie bei dem Netztest werden Lese- und Schreibeoperationen durchgeführt. Zusätzlich werden beide Operationen jeweils sequentiell und randomisiert durchgeführt. Die Testgrößen sind identisch zu den Disktests 4, 32, 265 Kilobyte und 1 und 4 Megabyte. Jeder Test läuft auch jeweils drei Minuten, was bei fünf Tests je Einheit eine Stunde Laufzeit ergibt. In der Ausgabedatei werden die Tests nach den vier Einheiten, also Lese- und Schreibetests jeweils sequentiell und randomisiert gruppiert.

4.1.2. Auswertung

CPU/linpack

Da alle Arraygrößen ein gleiches Verhalten zeigen, beschränken wir uns hier auf 2000×2000 als Beispielsgröße um das beobachtete Verhalten aufzuzeigen. Wie bei jedem Test werden die Ergebnisse für Linux und Windows gesondert betrachtet.
Bei Linux benötigt der Test in für die Berechnungen in der Arraygröße 2000×2000 nativ sowohl bei 32-Bit als auch bei 64-Bit im Durchschnitt etwa 20.5 Sekunden. Sowohl bei 32-Bit VMs als auch 64-Bit VMs ist nur ein minimaler Verlust zu erkennen. So liegt der Wirkungsgrad hier bei über 90%. Auch im Vergleich der unterschiedlichen VM Konfigurationen ist keine Tendenz zu erkennen.

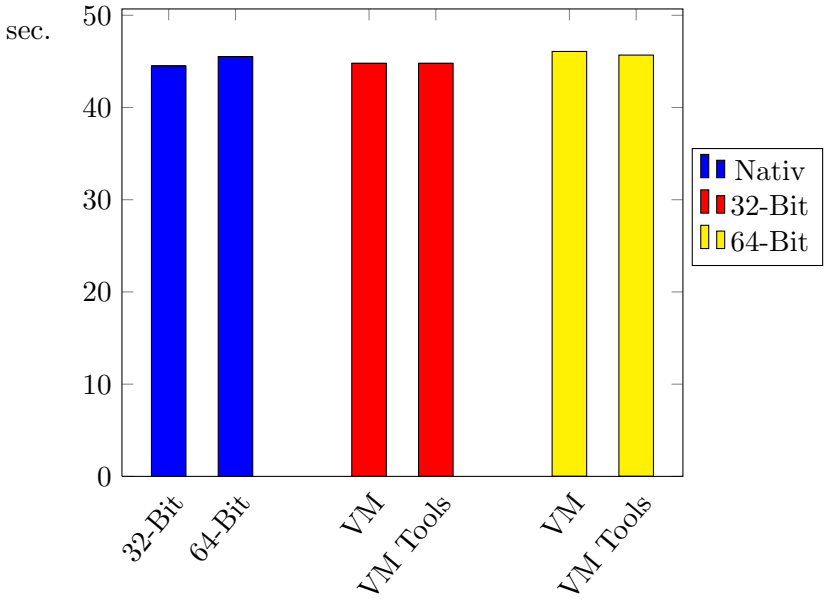
Tabelle 4.1.: Linpack Single Linux



Im direkten Vergleich benötigen bereits die nativen Maschinen unter Windows über die doppelte Zeit für die Berechnungen wie Linux. Sie benötigen in etwa 44.5 bzw. 45.5 Sekunden. Allerdings ist dies ein Wert der von den virtuellen Maschinen so gut wie gar nicht überboten

wird. So liegt auch hier der Wirkungsgrad bei über 90%.
Allgemein lässt sich also beobachten, dass sich reine CPU Rechenzeit sehr perfomant auf VMs in Xen übertragen lässt.

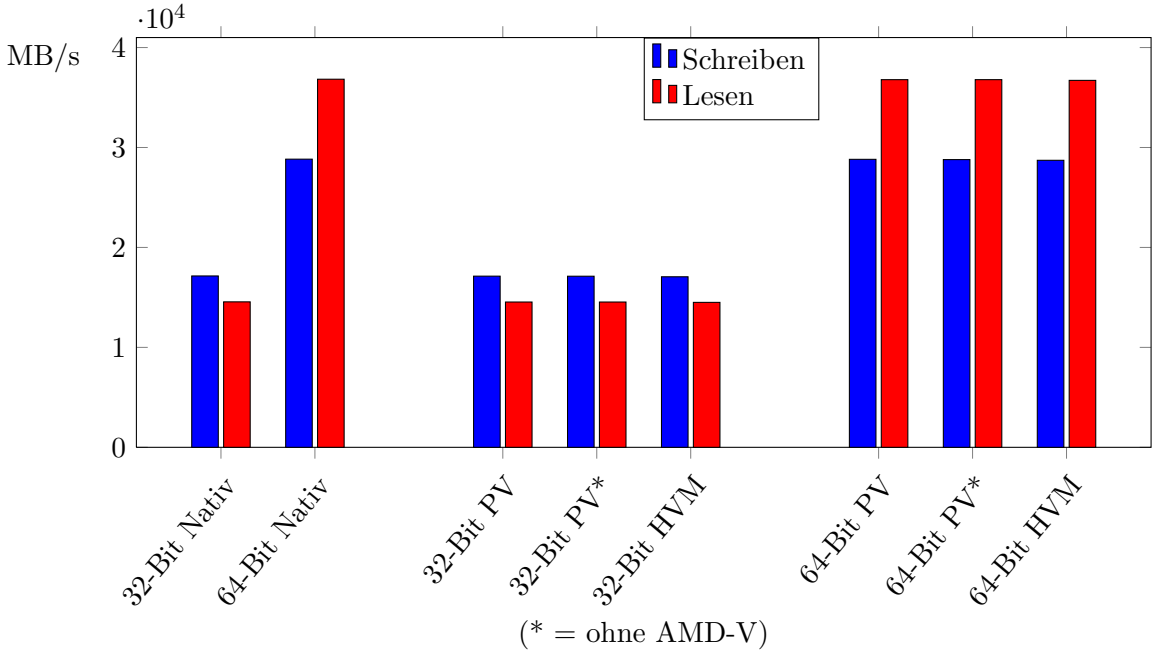
Tabelle 4.2.: Linpack Single Windows



RAM/ramspeed

Mit Hilfe von ramspeed wurde eigentlich nicht ein einzelner Test, sondern drei voneinander unabhängige Untertests durchgeführt. Dabei konnte die Schreib- und Lesegeschwindigkeit des Level 1 Caches, des Level 2 Caches und des Hauptspeichers ermittelt werden.
Bei den nativen Tests unter Linux ist beim Level 1 Cache zu beobachten, dass dem 64-Bit System ein viel höheres, fast doppelt so hohes, Ergebnis wie bei dem 32-Bit System erreicht wurde. Das liegt daran dass für das 64-Bit System ein eigens auf die erweiterte Berechnungsbandbreite der CPU angepasstes Benchmarkprogramm verwendet wurde. Bei den Tests des Level 2 Caches und des Hauptspeichers führt diese Anpassung allerdings nicht zu einer Verbesserung der Geschwindigkeit.
Allgemein und für alle drei Tests ist zu sagen, dass der Wirkungsgrad immer sehr nahe der 100% Marke ist, wirkliche Verluste sind nicht zu erkennen.

Tabelle 4.3.: Level 1 Cache - Linux



Bei Windows ist schon unter Nativ der bei Linux zu beobachtende Unterschied zwischen 32-Bit und 64-Bit Architektur nicht zu erkennen. Das liegt daran, dass ramspeed für Windows unglücklicherweise nur in einer 32-Bit Version erhältlich ist, die bei Linux angesprochene Optimierung auf die 64-Bit Architektur ist also nicht vorhanden. Dadurch konnte leider auch nicht weiter getestet werden, ob sich eine mögliche Optimierung auf die virtuellen Maschinen überträgt.
Zu den Ergebnissen der Tests in den VMs lässt sich die gleiche Beobachtung wie bei Linux machen: Es ist kein Verlust festzustellen. Alle Konfigurationen haben einen konstant guten Wirkungsgrad.

Tabelle 4.4.: Level 2 Cache - Linux

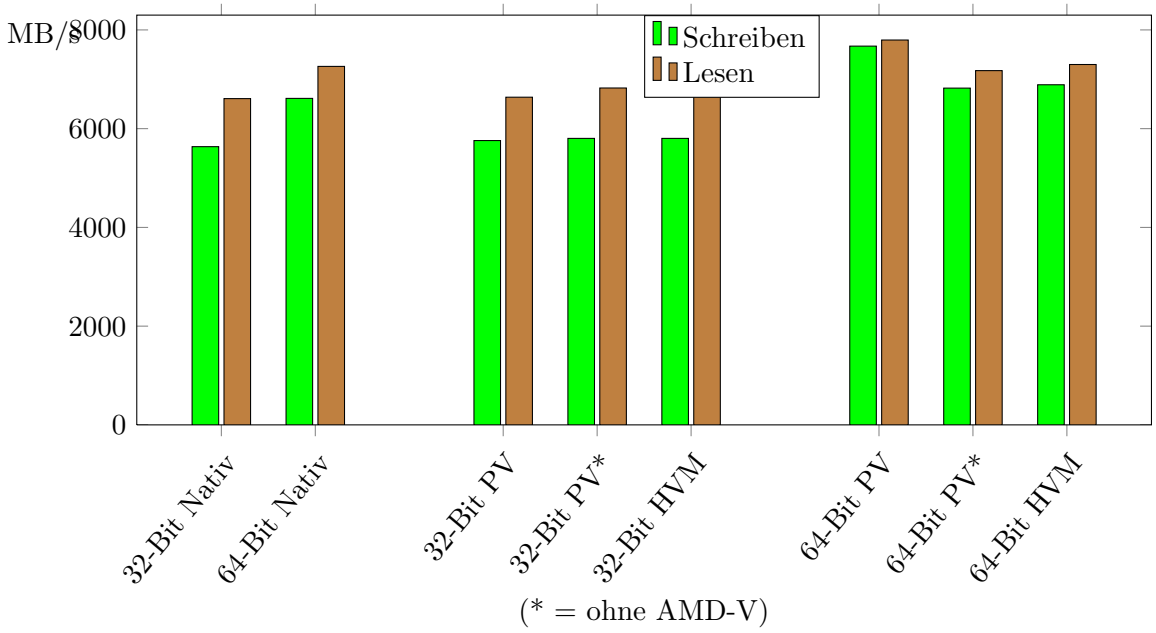


Tabelle 4.5.: Hauptspeicher - Linux

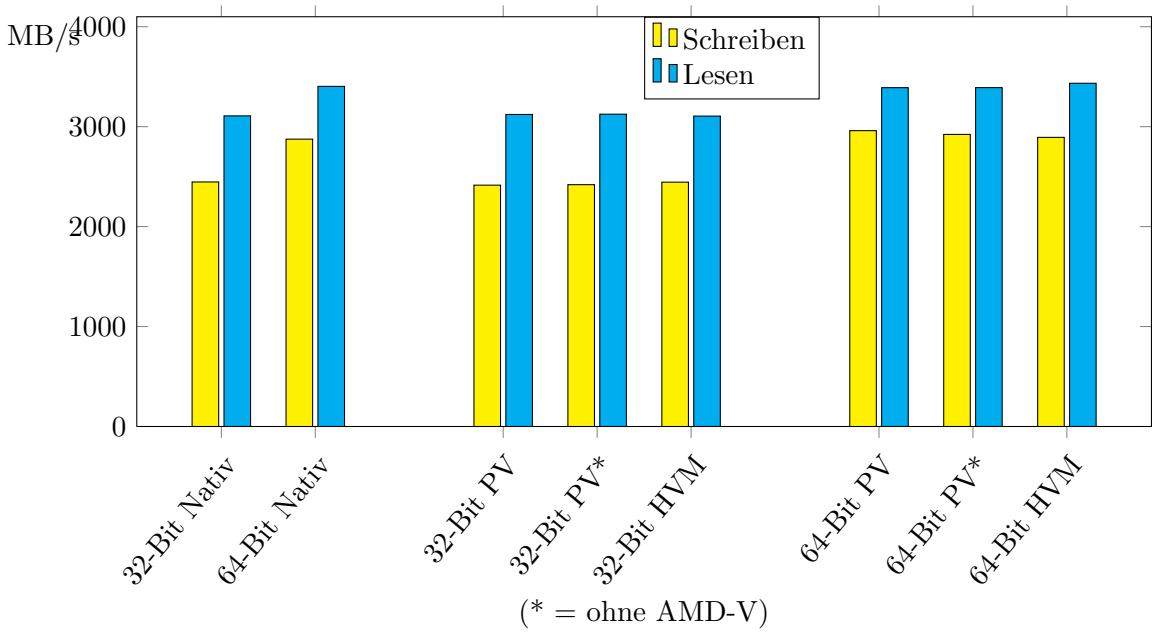


Tabelle 4.6.: Level 1 Cache - Windows

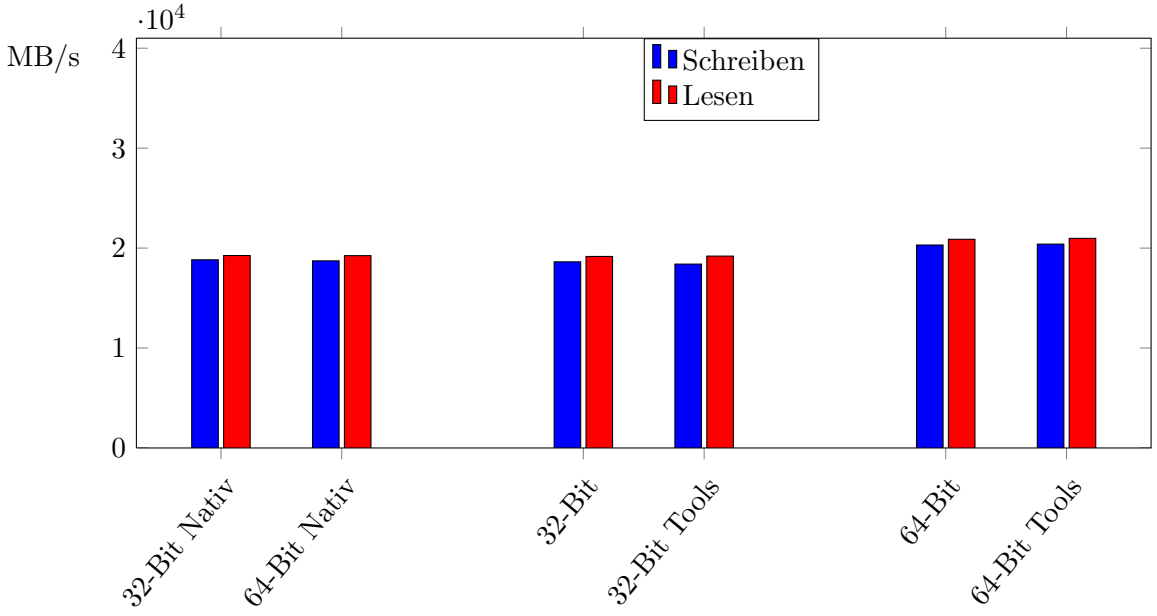


Tabelle 4.7.: Level 2 Cache - Windows

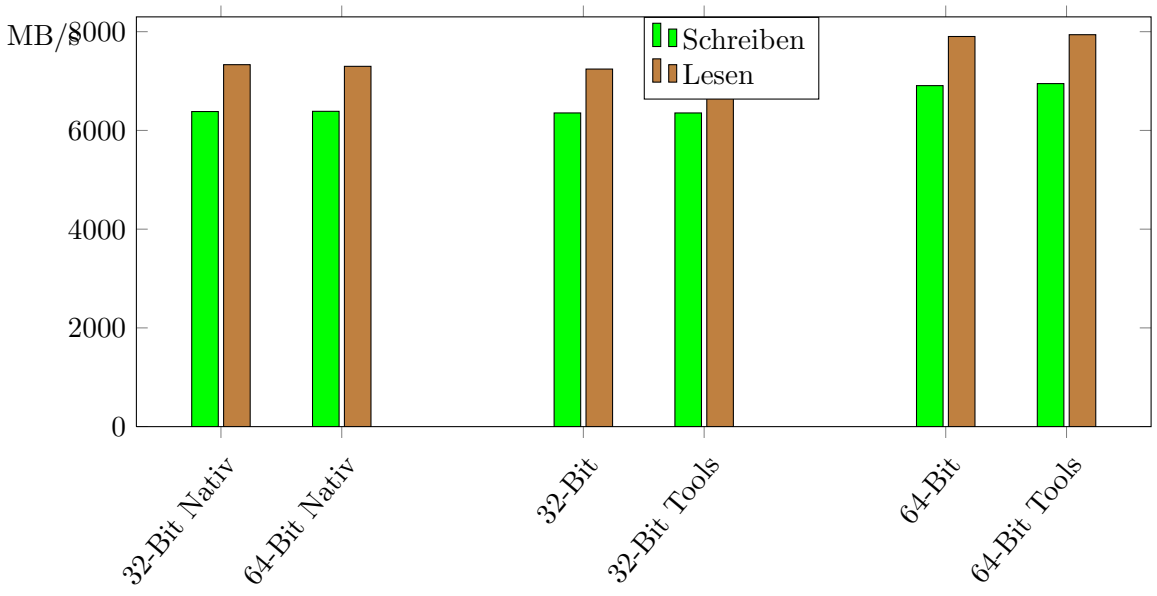
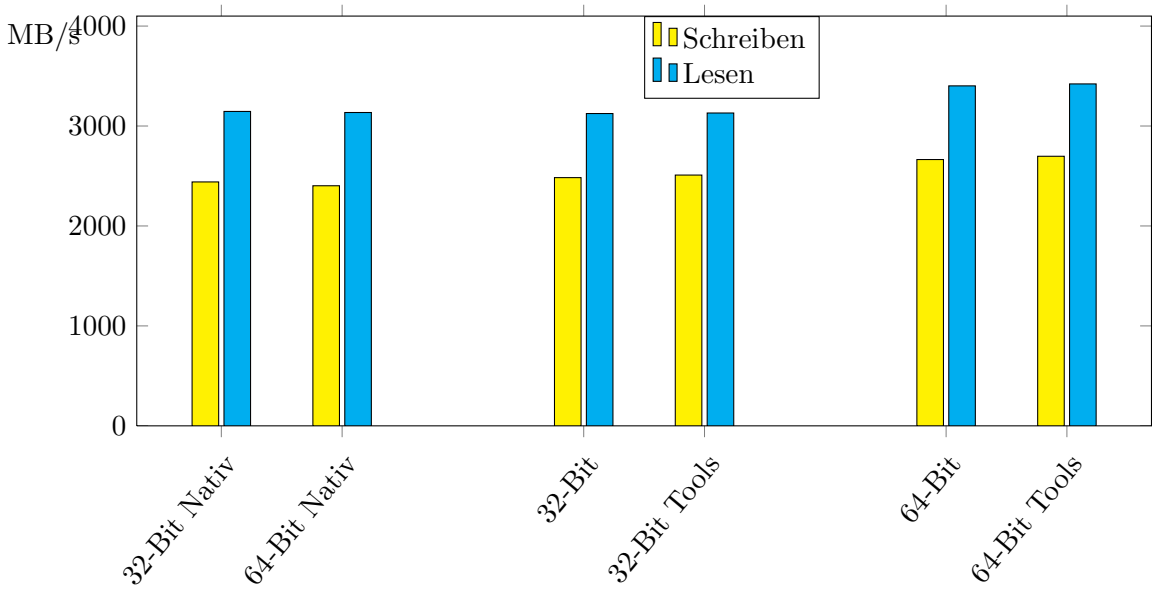


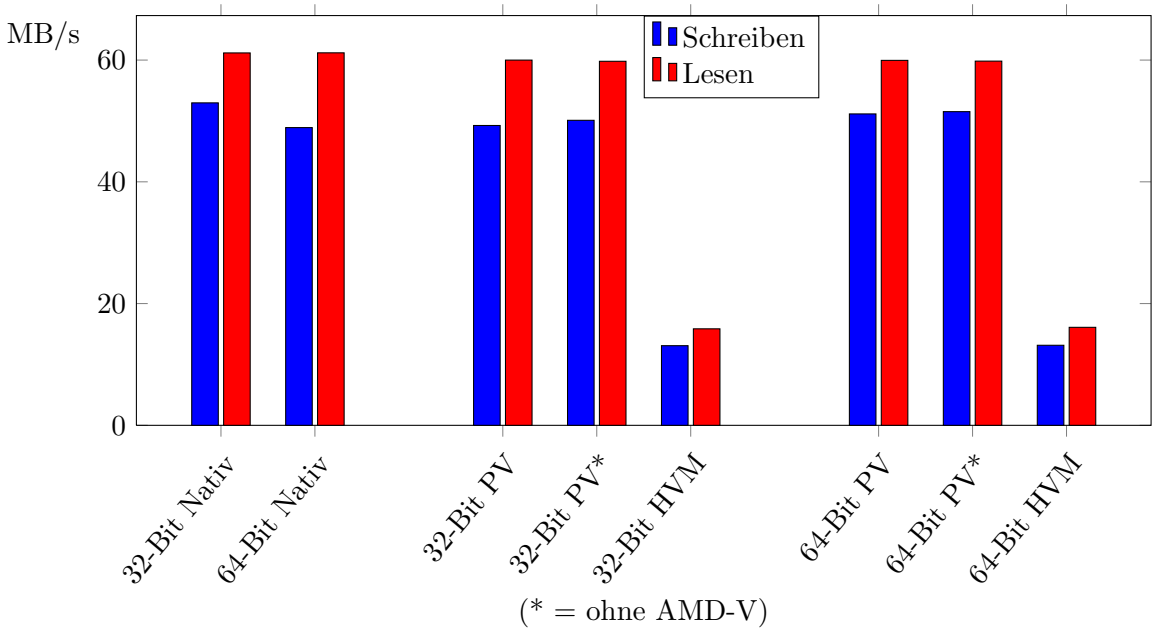
Tabelle 4.8.: Hauptspeicher - Windows



Netz/Iometer

Bei den Tests der Netzperfomanz hingegen, zeigt Xen bereits erste Schwächen auf. Bei den nativen Maschinen ist zu beobachten, dass allgemein mit einer höheren Lese- als Schreiberate zu rechnen ist, und dass die Werte hierfür optimalerweise bei etwa 50 MB/s bzw. 60 MB/s liegen.
Während sowohl bei 32-Bit als auch bei 64-Bit die paravirtuellen VMs sowohl mit als auch ohne aktiviertem AMD-V keine erkennbaren Verluste aufzeigen, bricht die Übertragungsrate bei vollvirtualisiertem Linux stark ein. Vollvirtualisiertes Linux unter Xen hat bei der Netzübertragung nur einen Wirkungsgrad von etwa 25%. Obwohl zu erwarten ist, dass vollvirtualisierte I/O Geräte wie Netzwerkkarten eine schlechtere Performanz liefern, da vom Hypervisor die Daten erst auf die tatsächliche Netzwerkkarte gelegt werden muss, überrascht dieses doch schon sehr schlechte Ergebnis.

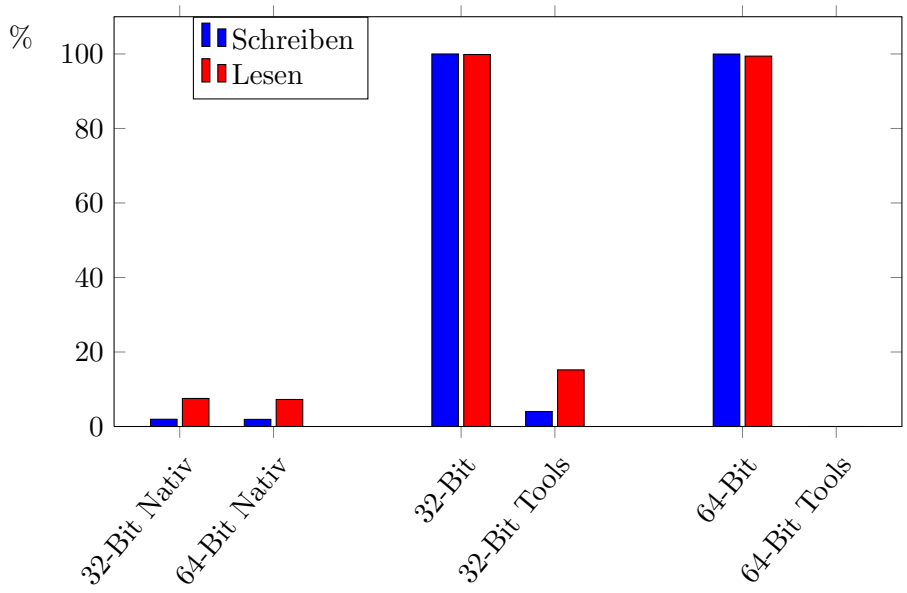
Tabelle 4.9.: Netz Single Linux



Hat man den Unterschied zwischen paravirtualisiertem und vollvirtualisiertem Linux im Hinterkopf, erwartet man bei Windows Gästen schon ein nicht optimales Ergebnis. Der Grund hierfür ist, dass Windows Gäste nicht paravirtualisiert betrieben werden können.
Das Ergebnis überrascht dann aber doch. Während die Rate bei 64-Bit auf “nur” 70% absinkt, fällt sie bei 32-Bit auf 13 MB/s bzw. 15.5 MB/s, also in etwa bei beidem 25%.
Der Grund hierfür findet sich in einem anderen Wert. Während unter Linux weder beim nativen Test auf dem Rechner, noch bei allen VM Test in der VM eine CPU Auslastung festgestellt werden konnte, haben bereits die nativen Windowsrechner eine - wenn auch nur leichte - CPU Auslastung von 2% bzw. 7%. Die Windowsgäste unter Xen hingegen, haben laut Iometer während dem Lese- und Schreibtest eine Auslastung von 100% (4.10).

Daher hofft man, dass die Tools, die so genannten PV Treiber, die man unter Xen bei

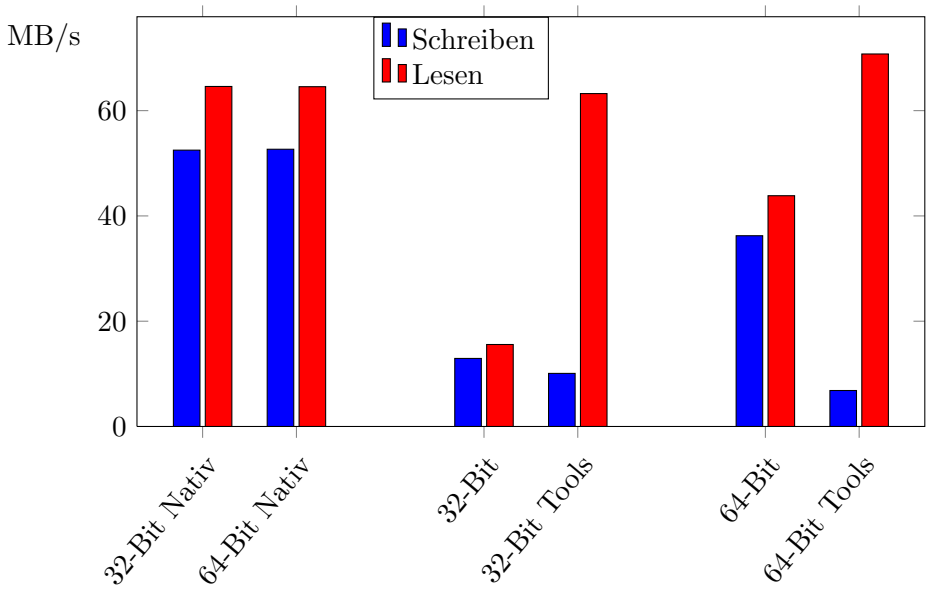
Tabelle 4.10.: Netz Windows - CPU Auslastung



Windows VMs installieren kann, dieses Problem beheben und den Übersetzungsaufwand reduzieren, damit auch unter Windows annehmbare Werte bei der Übertragung erzielt werden können. Wenn man allerdings das Diagramm betrachtet 4.11, merkt man, dass diese PV Treiber anscheinend ihre Produktreife noch nicht erreicht haben.
Während die Leserate stark ansteigt, und einen Wirkungsgrad von 100% erreicht, fällt die Schreiberate noch weiter ab. Zur Erinnerung, es findet zu keiner Zeit ein gleichzeitiges Lesen und Schreiben statt, es kann hier also auch nicht zu einer Priorisierung einer Operation gekommen sein.

Für Heimbedarf, wo man im Allgemeinen eher einen geringen Upload ins Internet vom ISP ("Internet Service Provider") zur Verfügung gestellt bekommt, aber einen im Verhältnis großen Download besitzt, und diese VM nur Netzverkehr ins Internet betreibt, kann mit Hilfe dieser Tools ein normaler Betrieb stattfinden. Möchte man aber mehr Daten von dieser VM senden, beispielsweise im lokalen Netz, oder mit einem größeren Upload, kommt man mit einem Windows Gast unter Xen also nicht zu einem zufriedenstellenden Ergebnis. Allgemein lässt sich beobachten, dass bei Auslastung der Netzwerkkarte, vollvirtualisierte Maschinen unter Xen gegenüber paravirtualisierten nicht ansatzweise konkurrenzfähig sind. Selbst die momentan für Windowsgäste verfügbaren Tools können diesen Abstand nicht beheben.

Tabelle 4.11.: Netz Single Windows



Disk/Iometer

Bei den sequentiellen Disktests werden Daten sequentiell geschrieben oder gelesen, das bedeutet, es wird auf Daten zugegriffen, die auf dem Datenträger physisch direkt hintereinander liegen. Dabei sind die Unterschiede zwischen Lesen und Schreiben sehr gering. Lesezugriffe sind geringfügig schneller. Der Unterschied ist allerdings lediglich 2.5 bzw. 4.5 MB/s. Beim Schreiben gibt es bei den Linuxgästen keine Verluste. Allein beim Lesen gibt es leichte Verluste von kaum mehr als 10%.

Beim randomisiertem Zugriff allerdings, das bedeutet beim Zugriff auf verschiedene, vom Bereich auf den als letztes zugegriffen wurde unabhängige Datenbereiche, zeigen sich größere Verluste.
Hier ist schon bei den Nativen ein größerer Unterschied festzustellen. Erwartungsgemäß liegen beide Werte niedriger als beim sequentiellen Zugriff, da eine Festplatte kein RAM ("Random Access Memory") ist, sprich der Schreib-/Lesekopf sich viel bewegen muss und dafür Zeit benötigt. Allerdings ist die Schreibrate viel niedriger als die Leserate. Während die Leserate 20 statt 30 Megabyte pro Sekunde beträgt, werden randomisiert nur knapp 14 statt wie sequentiell 28 oder 26 MB/s geschrieben.
Ein ähnliches Verhalten zeigt sich auch bei den virtuellen Maschinen. Nur ist jetzt die Leserate nahezu unverändert, während die Schreibrate auf etwa 60% einbricht.
Dass die Leserate nur um höchstens 5% einbricht, deutet darauf hin, dass der Flaschenhals der randomisierte Zugriff ist, der einen stärkeren Verlust verursacht als durch die Virtualisierung entstanden wäre.

Nachdem bisher vollvirtualisiert (auch logischerweise) immer schlechtere Ergebnisse geliefert hat, und die Tools für Windows unbefriedigende Werte zeigten, wäre es anzunehmen, dass

Tabelle 4.12.: Disk Single Linux sequentiell

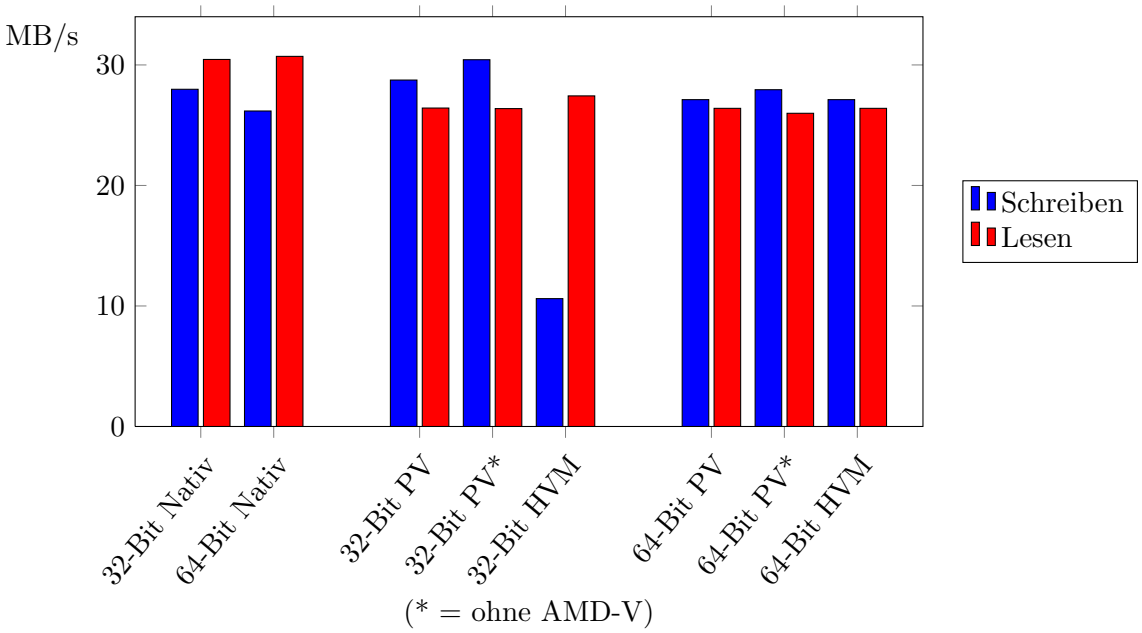
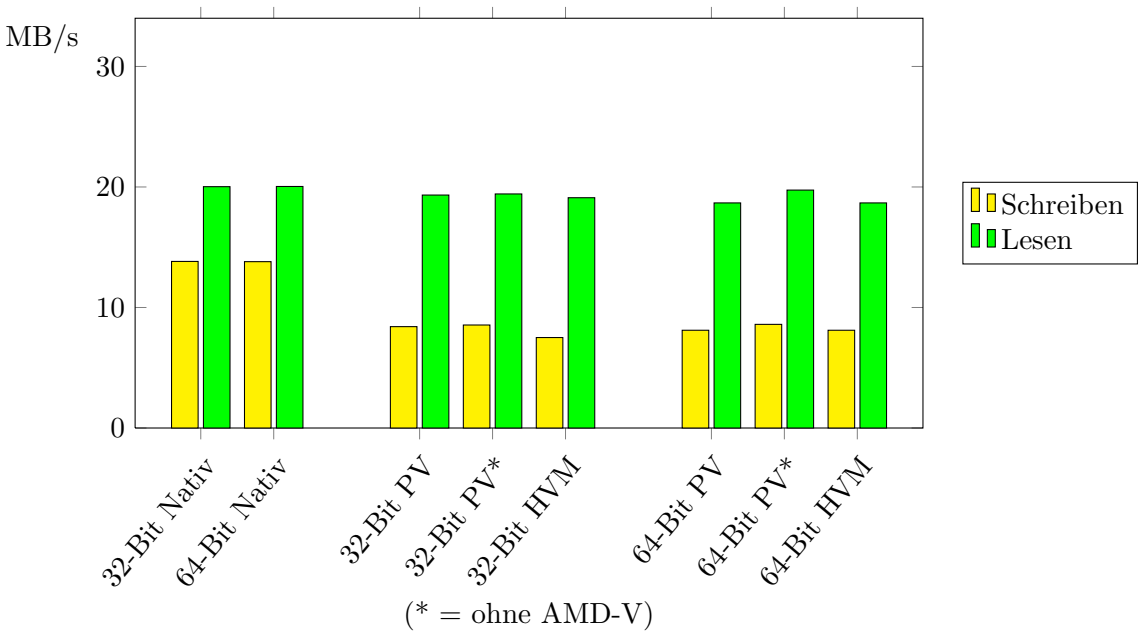


Tabelle 4.13.: Disk Single Linux randomisiert



der Disktest auf Windowsgästen noch schlechtere Werte liefert als die unter Linux.
Da überrascht das Ergebnis umso mehr. Bei keinem Test ist ein Wert schlechter als Nativ zu beobachten. Die Ergebnisse der nativen Tests unter Windows liegen in etwa auf dem gleichen Niveau wie die unter Linux, man kann also auch nicht von einem Flaschenhals beim

Betriebssystem sprechen.
Allerdings ist wie bei bisher allen Tests mit Iometer eine - wenn auch nur leichte - CPU Auslastung zu beobachten.

Tabelle 4.14.: Disk Single Windows - sequentiell

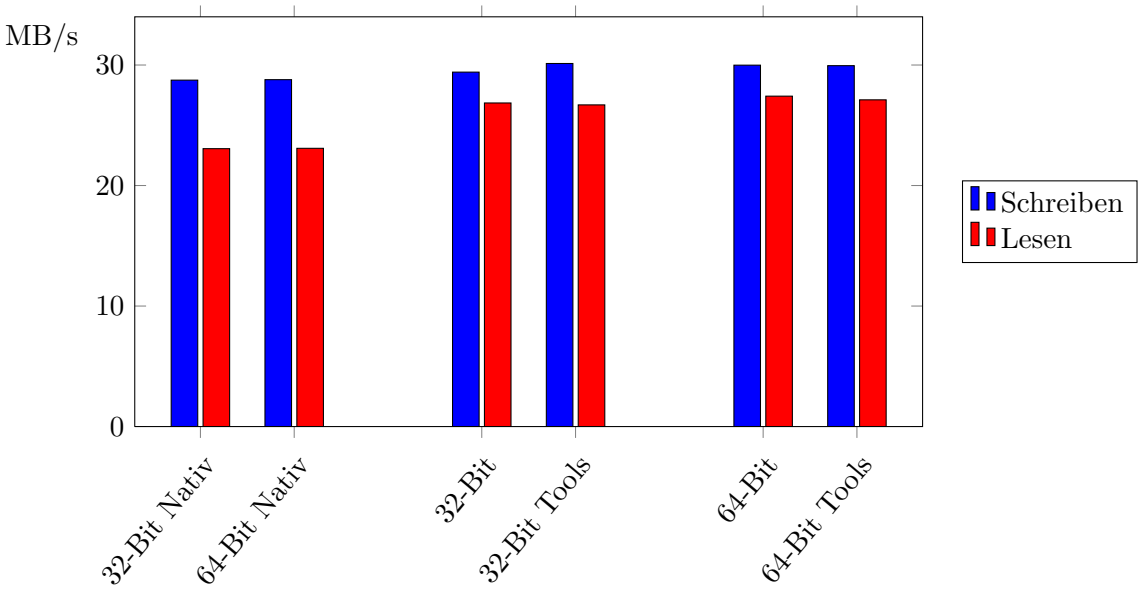


Tabelle 4.15.: Disk Single Windows - randomisiert

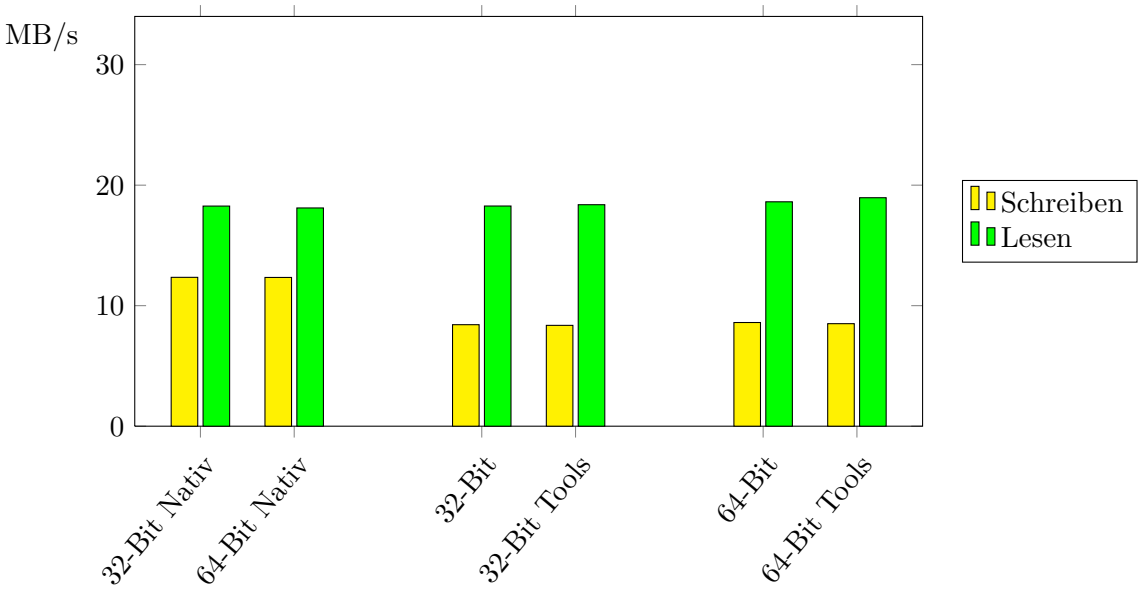
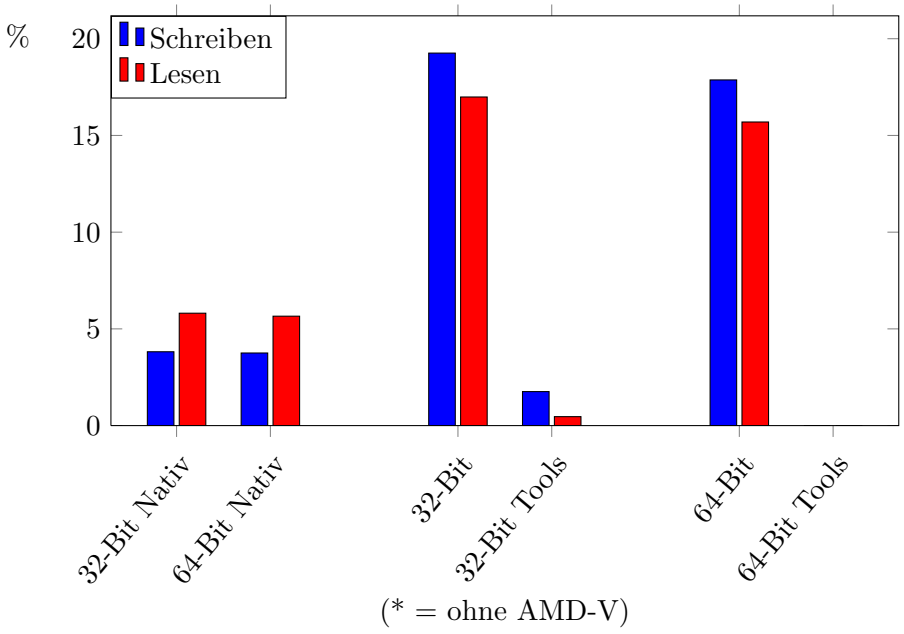
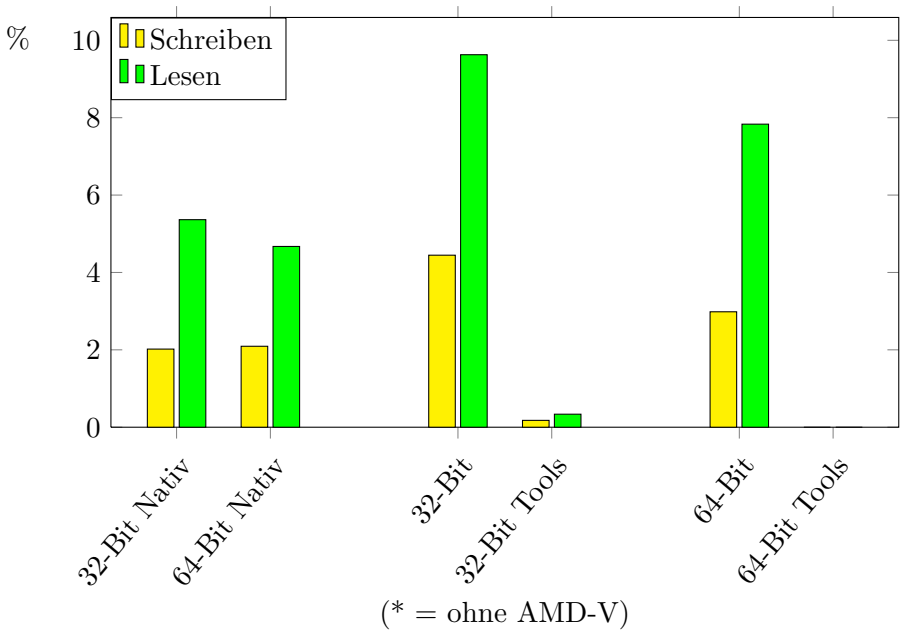


Tabelle 4.16.: Disk Single Windows - CPU Auslastung



(* = ohne AMD-V)

Tabelle 4.17.: Disk Single Windows - randomisiert - CPU Auslastung



(* = ohne AMD-V)

4.2. Parallele Benchmarks

Nachdem von unterschiedlichsten Konfigurationen von VMs vielfältige Tests durchgeführt wurden, war das Ziel als nächstes, im Rahmen der Möglichkeiten zu überprüfen, inwieweit Xen skalierfähig ist. Das bedeutet, wie sich das Ergebnis ändert, wenn mehrere virtuelle Maschinen gleichzeitig laufen, und gleichzeitig auf unterschiedliche Art und Weise Last erzeugen. Beispielsweise gleichzeitig auf das gleiche Medium, zum Beispiel den Hauptspeicher oder das Netz, zugreifen. Da es nicht möglich war, diese Tests unter allen möglichen Konfigurationen durchzuführen, wurde eine Beispielskonfiguration gewählt, unter der alle Tests durchgeführt werden sollten. Dabei wurde selbstverständlich die Konfiguration gewählt, die bei den bisherigen Benchmarks den höchsten Wirkungsgrad erreicht hat.
Konkret für den Test gewählt wurde Linux paravirtualisiert in 32-Bit mit aktiviertem AMD-V.
Liste an durchgeführten Tests: Ohne zu vergessen, dass die Tests an einem Heimcomputer mit beschränkten Ressourcen durchgeführt werden, wurde versucht möglichst stark zu skalieren um realistische Bedingungen zu schaffen. Um aber auch noch ausreichend Ressourcen in den einzelnen VMs zu haben, wurde sich darauf beschränkt zwei bzw. drei virtuelle Maschinen gleichzeitig auf einem Hypervisor laufen zu lassen. Weiterhin wurde versucht viele unterschiedliche in der Praxis auftretende Szenarien abzudecken und dabei aber auch darauf zu achten, nicht zu viele Parameter gleichzeitig zu verändern, weil sonst nicht mehr festgestellt werden könnte, welche Einstellung zu dem erreichten Ergebnis geführt hat.

4.2.1. Durchführung

Konkret durchgeführt wurden folgende Tests mit den bekannten und bisher verwendeten Benchmarktools.

CPU, RAM, Netz, Disk

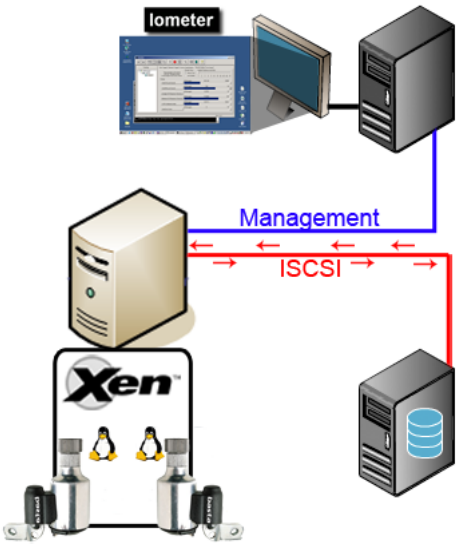


Abbildung 4.5.: Aufbau des 2-Disk Tests als Beispiel für die parallelen synthetischen Tests

Als erstes wurde getestet wie effizient der CPU Scheduler von Xen arbeitet. Dazu wurden

mehrere, um genau zu sein zwei bzw. drei, virtuelle Maschinen gestartet. Auf jeder VM wurde dann wie bei den vorherigen Tests ein linpack gestartet, der in den bekannten Arraygrößen 10 Durchläufe berechnet hat. Der linpack Server jeder VM lief dabei auf dem Managementrechner. Dabei wurde die Zeitmessung wie gegeben über das dedizierte Netz abgefragt.
Das gleiche Prinzip wurde bei den parallelen RAM Tests angewendet, anstelle eines linpack lief hier aber in jeder VM ein ramspeed, dadurch war kein dediziertes Netz nötig. nur dass hier anstatt einem linpack, auf jeder VM ein ramspeed lief und kein dediziertes Netz nötig war.
Für die parallelen Netztests wurde auf dem Managementrechner die Iometer GUI und in jeder VM ein Dynamo gestartet. Wie bei den vorherigen Netztests war das Testnetz dabei das dedizierte Netz zum Managementrechner, auf dem für jede VM ein passiver Dynamo als anderes Ende der Verbindung lief.
Ähnlich liefen die Disktests ab. Nur wurde dazu je VM ein Image vom iSCSI Target in jede VM nach bisherigem Prozedere eingebunden und ein Dynamo mit einem Disk Worker in jeder VM gestartet.
Wie erwähnt wurden alle vier Tests dabei mit zwei und drei VMs durchgeführt. Interessant hierbei ist, ob die Verteilung zwischen den VMs fair ist, das bedeutet ob alle VMs circa die gleichen Werte erzielen oder ob eine VM stark bevorteiligt wird, während den anderen kaum noch Bandbreite zur Verfügung gestellt wird, und natürlich, ob die aufsummierte Übertragungsrate gleich der Übertragungsrate bei einem Test mit nur einer VM ist, sprich ob kein Verlust durch den Mehrbetrieb entstanden ist.

Netz bzw. Disk unter Last

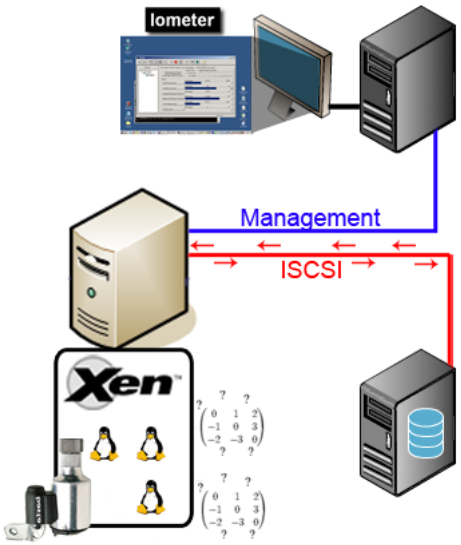


Abbildung 4.6.: Aufbau für den Disk Test unter Last

Als nächstes sollte untersucht werden, wie sich die Übertragungsrate bei einem normalen Netz- bzw Disktest ändert, wenn die CPU des Hosts, also auch die der Test-VM zugewiesene vCPU komplett ausgelastet ist. Da der Hostrechner, wie im Kapitel 3.1 erwähnt, eine Dual-Core CPU besitzt, wurden zusätzlich zu der Test-VM zwei weitere VMs gestartet. Beide sind

ebenfalls mit einer vCPU ausgestattet. Auf diesen beiden "Last-VMs" wurde jetzt jeweils ein linpack Client gestartet. Damit sollte die physische CPU komplett ausgelastet sein und für die Test-VM CPU-mäßig schlecht möglichste Voraussetzungen geschafft sein.
Da es in den Last-VMs nicht auf die Ergebnisse des linpack Tests ankommt, und daher auch keine genau Zeitmessung nötig ist, wurde der jeweilige linpack Server ebenfalls in der Last-VM gestartet. Dem linpack Client wurde dementsprechend auch `localhost` als Parameter für den Zielrechner mitgegeben.
Dann wurde in der Test-VM ein normaler Netz- und ein normaler Disk-Test wie unter 4.1.1 bzw. 4.1.1 durchgeführt.

Iometer Disk und Netz

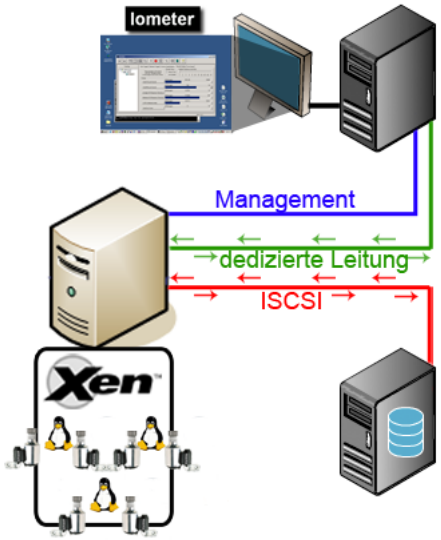


Abbildung 4.7.: Aufbau für den Disk und Netz Test

Als ausführlichster Test soll nun zum ersten Mal das Zusammenspiel von mehreren Tests untersucht werden.
Bis jetzt wurden zwar schon mehrere VMs gleichzeitig betrieben und getestet, allerdings wurde noch weiter nicht untersucht, wie sich einzelne Komponenten in einem Verbund verhalten. Im Test 4.2.1 wurde schon ein Schritt in diese Richtung gemacht, in diesem Test wird noch ein Schritt weiter gegangen.
Hier sollen erstens mehrere VMs, zweitens zwei Tests gleichzeitig durchgeführt werden. Bei diesem Test werden also auf drei VMs jeweils ein Disk- und ein Netztest durchgeführt. Die Ergebnisse dieses Tests können dann einerseits mit dem 3-Netztest, dem 3-Disktest, und beiden Single Testvarianten verglichen werden.

Inter VM Kommunikation

Als letztes wurde noch ein Test durchgeführt, der nicht direkt mit vorherigen Tests verglichen werden kann. Er hat dafür umso mehr einen Realitätsbezug und kann mit die Effizienz von Virtualisierungen darlegen. Dabei soll die Netzübertragung zwischen zwei VMs auf dem gleichen Hypervisor untersucht werden. Theoretisch ist die Rate dieser Übertragung nahezu

nicht begrenzt, da die Daten weder auf Haupt- noch auf Hintergrundspeicher gespeichert werden muss, und keine physische NIC angesprochen werden muss. Die Übertragung verläuft allein auf dem Hostrechner. Eine solche Übertragung kann in der Praxis daher oft angewendet werden, wenn Rechner zwar logisch voneinander getrennt werden sollen, aber trotzdem auf definierten Wegen miteinander kommunizieren müssen.

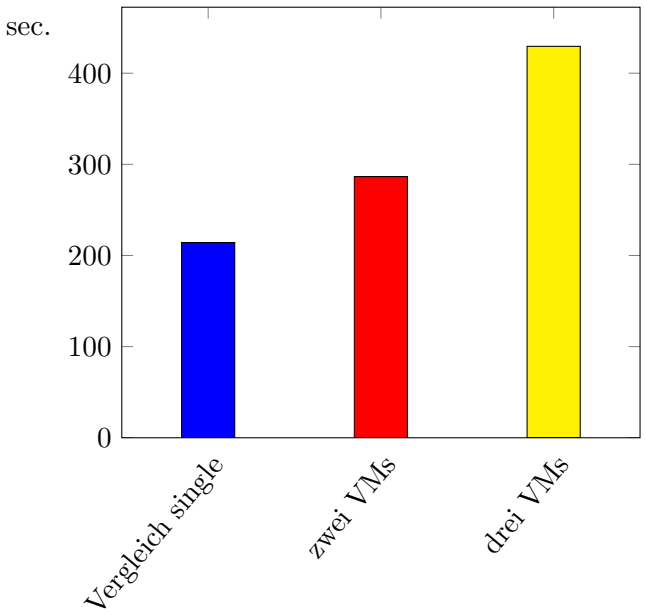
4.2.2. Auswertung

Bei der Auswertung der parallelen Tests werden - wenn nicht anders angegeben - die Durchschnittswerte aller getesteten VMs aus diesem Test mit dem vergleichbaren Ergebnis der Wirkungsgradbenchmarks verglichen. Das bedeutet es wird aus dem jeweiligen Test aus Kapitel 4.1.2 das Ergebnis der paravirtualisierten Ubuntu VM mit 32-Bit und aktiviertem AMD-V genommen, da die parallelen Benchmarks wie oben erläutert in dieser Konfiguration durchgeführt wurden.

linpack parallel

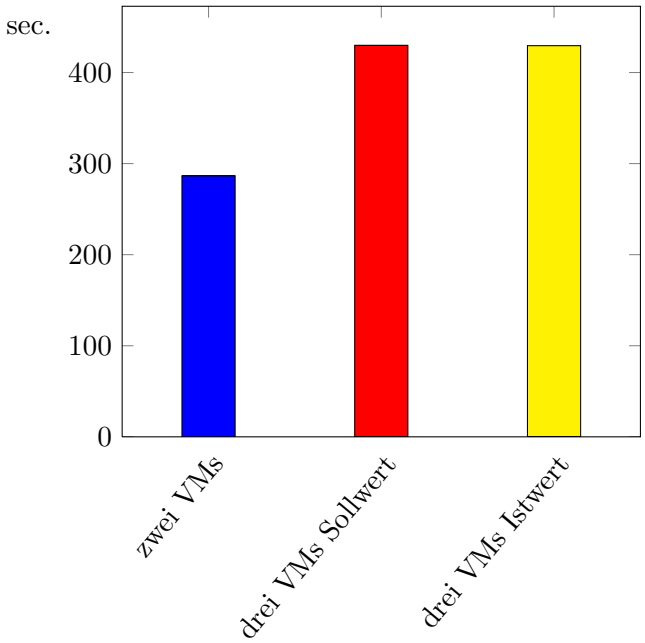
Um die Ergebnisse der parallelen CPU Tests im Vergleich mit dem Single Test zu verstehen und richtig zu deuten, ist eine Vorüberlegung notwendig.
Einer VM ist immer nur eine vCPU zugewiesen, es wird also auch immer nur ein Core der VM zur Berechnung zugeteilt. Der Hostrechner besitzt allerdings zwei Kerne. Während bei dem Single Test immer nur ein Core arbeitet (auch wenn der arbeitende und der schlafende in beliebiger Frequenz wechseln können), arbeiten bei dem 2-linpack Test immer beide Kerne. Theoretisch sollten also bei dem 2-linpack die Ergebnisse herauskommen, wie bei dem Single Test.
Erst bei dem 3-linpack Test gibt es mehr vCPUs als Kerne. Dadurch muss immer eine der drei vCPUs - und dadurch auch eine der drei VMs - warten, während die anderen beiden arbeiten. Auch hier gilt wieder dass eine vCPU nicht direkt einem Core zugewiesen ist, und die Zuordnung in beliebiger Frequenz wechseln kann.
Vorneweg gibt es zu erwähnen, dass die unterschiedlichen Testgrößen identische Ergebnisse vorzeigen, deswegen beschränken wir uns hier auf die Darstellung des Ergebnisses mit der Arraygröße 4000×4000.

Tabelle 4.18.: Durchschnitt linpack parallel



Vergleicht man in diesem Diagramm zunächst den Durchschnittswert beim Test mit zwei VMs mit dem einzelnen, kann man einen Verlust von fast 25% beobachten. Dies bedeutet, allein die Tatsache dass der CPU Scheduler nun zwischen zwei virtuellen Maschinen hin und her wechseln muss, wirkt sich derart stark auf die Dauer der Berechnung aus. Selbstverständlich gibt es vorher bereits zwei VMs, die `dom0` wird ja ebenfalls virtualisiert. Allerdings läuft während dem Test in der dom0 kein Prozess der viel Berechnungszeit benötigt, deswegen ist anzunehmen dass der CPU Scheduler nur sehr sporadisch der dom0 den Fokus übergibt, während im Test mit je einem linpack in zwei unprivilegierten VMs ein hochfrequenter Wechsel von Nöten ist.
Das Ergebnis des 3-linpack Tests muss vorher noch an einen Sollwert unter der Annahme es existierte für jede vCPU ein Core angepasst werden, da sonst keine Deutung möglich ist. Wie erwähnt gab es in den bisherigen Tests nicht weniger Kerne als vCPUs, so dass immer alle VMs gleichzeitig arbeiten konnten. Jetzt muss zu jeder Zeit eine der drei VMs warten, weshalb es zu einer Erhöhung der durchschnittlichen Arbeitszeit um 50% kommt.
Möchte man also 2-linpack und 3-linpack vergleichen, muss man den Wert von 2-linpack mal 1.5 nehmen, dieser Wert hätte dann die Bedeutung “diesen Wert sollte 3-linpack ohne zusätzlichen Verlust erreichen (=Sollwert)” und ihn mit dem tatsächlich erreichtem Wert (=Ist-Wert) vergleichen.

Tabelle 4.19.: Durchschnitt linpack parallel - angeglichen



Dabei sieht man dass der Ist- und der Soll-Wert exakt identisch sind. Es ist also beim Übergang von zwei auf drei VMs kein weiterer Verlust zu beobachten.
Dies lässt sich damit erklären, dass der CPU Scheduler sowieso schon durchgehend arbeitet, und es unerheblich ist, ob zusätzlich zu den arbeitenden Maschinen sich noch eine Maschine im Wartezustand befindet. Durch den Creditscheduler entsteht auch bei der Auswahl der als nächstes zu aktivierenden Maschine kein zusätzliches Aufwand.

ramspeed parallel

Bei den parallelen CPU Cache und Hauptspeicher Tests kann man zwei unterschiedliche Beobachtungen machen.
Als erstes fällt bei der Betrachtung des Tests mit zwei VMs im Vergleich zum Single Test auf, dass der Durchschnitt nahezu identisch ist. Beim Level 1 und Level 2 Cache sind keine Unterschiede zu erkennen, einzig beim Hauptspeicher fällt ein Unterschied um ca. $x\%$ auf. In der Summe hat sich die Lese- und Schreiberate im Vergleich zum Single Test also nahezu verdoppelt. Wie unter 3.1 erwähnt, besitzt der Hostrechner einen Level 1 und Level 2 Cache pro Kern, sowie zweimal zwei RAM Steckplätze die jeweils per Dual-Channel verbunden sind. Deshalb gibt es sowohl beim Cache als auch beim Hauptspeicher zwei Busse, die unabhängig voneinander angesprochen werden können, und die jeweils die volle Übertragungsrate bieten. Beim Cache lässt sich bei parallelen Zugriff also tatsächlich die doppelte Rate übertragen. Einzig beim Hauptspeicher ist nicht ganz eine Verdopplung zu erkennen, allerdings doch auch ein ziemlicher Anstieg.
Vergleicht man als nächstes die Summe der Geschwindigkeit zwischen dem Test mit zwei und mit drei VMs, fällt auf dass hier kein weiterer Anstieg zu verzeichnen ist. Es handelt sich also tatsächlich um zwei und nicht mehr Busse, die parallel angesprochen werden können. Zwar ist die Geschwindigkeit beim Level 2 Cache und auch beim Hauptspeicher geringfügig geringer als beim Test mit zwei VMs, im Vergleich mit einer VM ist aber immer noch ein starker Anstieg zu verzeichnen. Unter entsprechenden Voraussetzungen kann im Mehr-VM-Betrieb also die Vorhandene Hardware effektiver ausgenutzt werden.

Tabelle 4.20.: Level 1 Cache - Durchschnitt

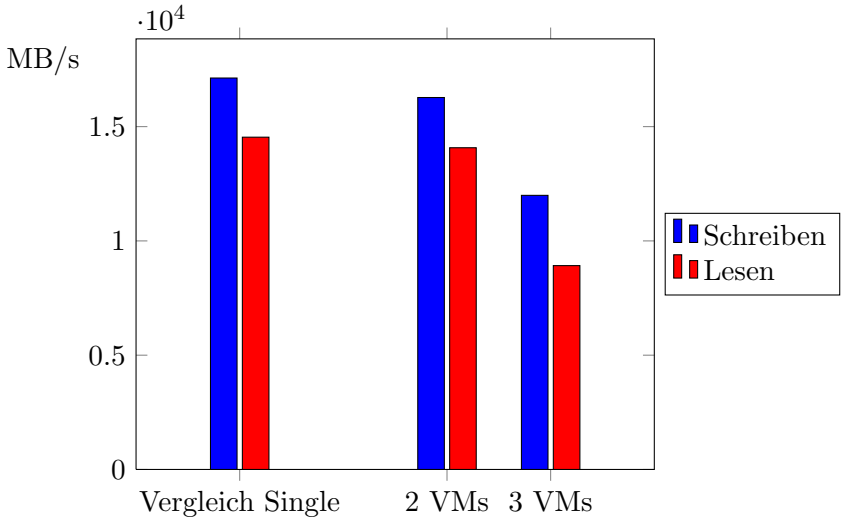


Tabelle 4.21.: Level 2 Cache - Durchschnitt

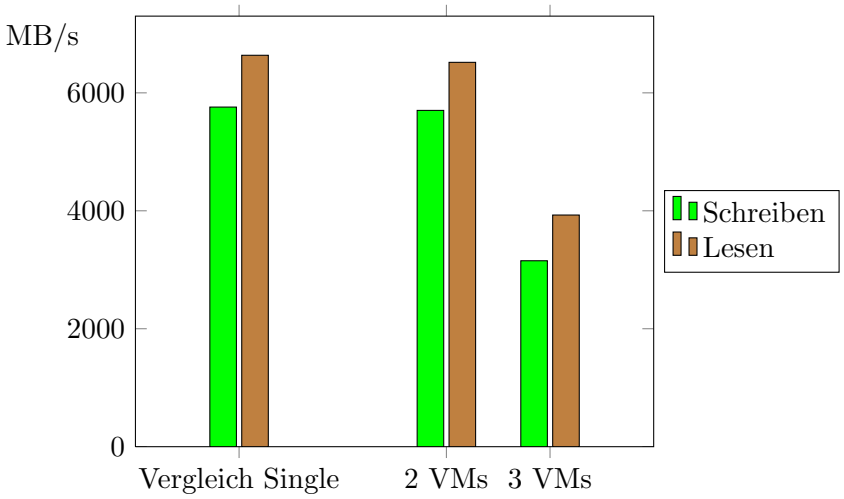


Tabelle 4.22.: Hauptspeicher - Durchschnitt

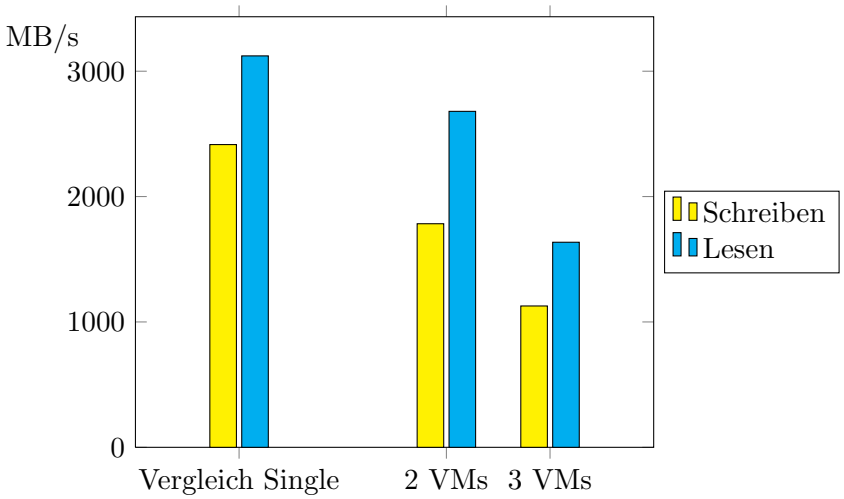


Tabelle 4.23.: Level 1 Cache - Summe

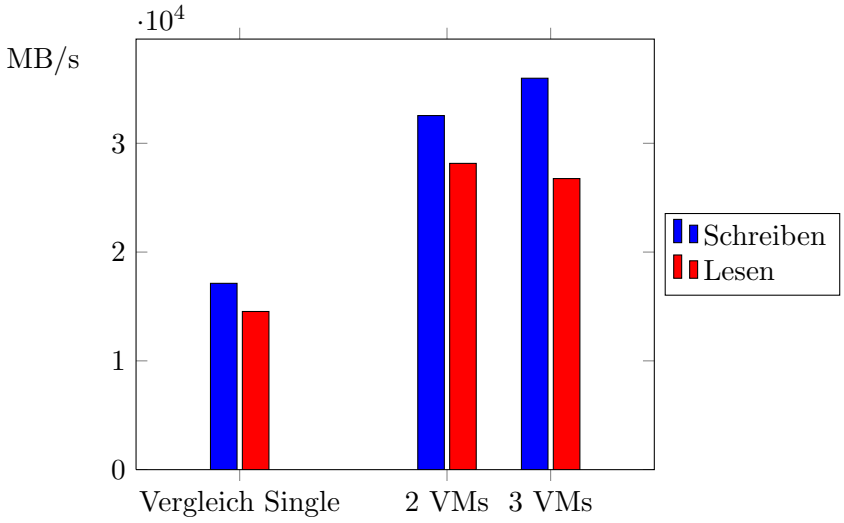


Tabelle 4.24.: Level 2 Cache - Summe

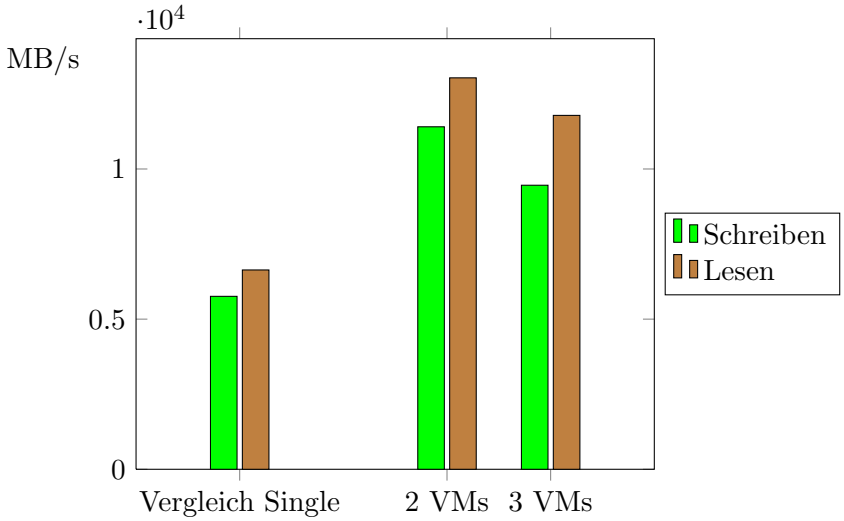
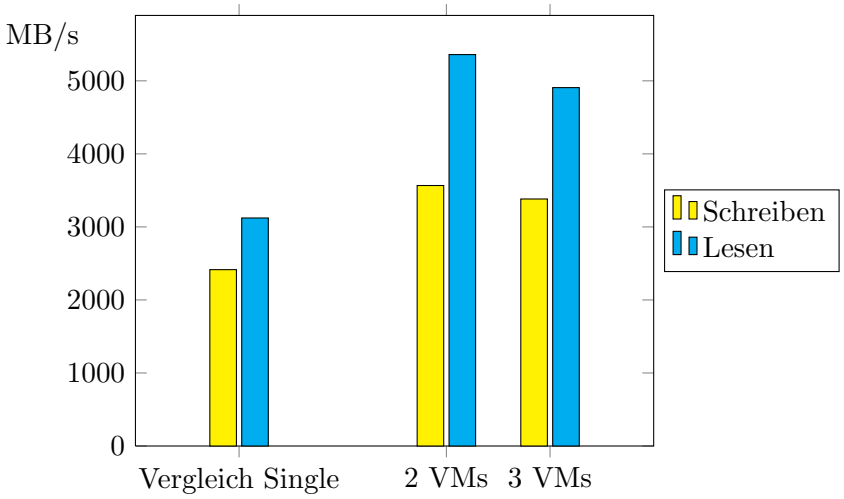


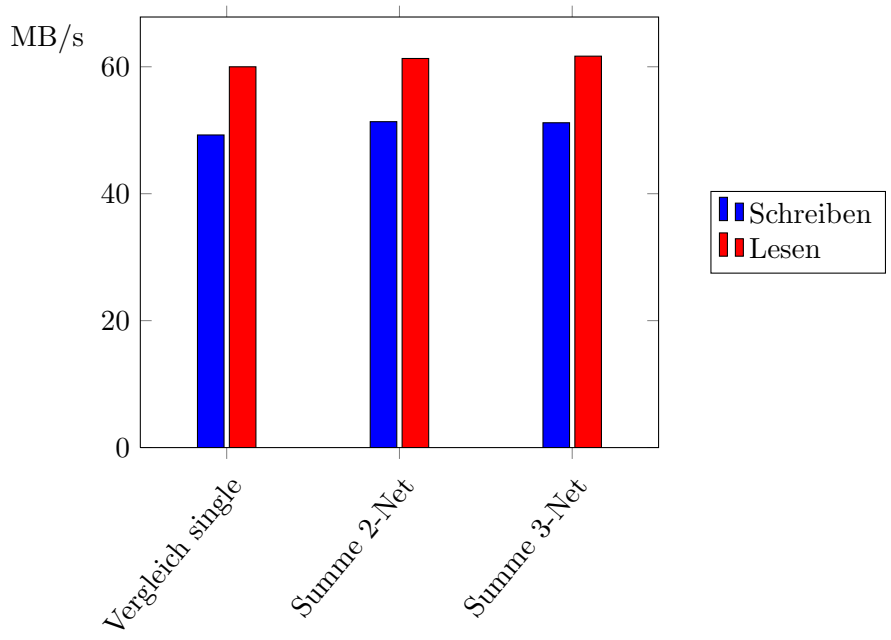
Tabelle 4.25.: Hauptspeicher - Summe



Netz parallel

Die parallelen Netztests unterscheiden sich nicht mit von dem Singletest. Selbstverständlich ist die Übertragungsrate bei den einzelnen Tests geringer, da die zur Verfügung stehende Bandbreite auf die einzelnen Dynamos von Iometer aufgeteilt wird, in der Summe ist aber sowohl bei 2-Net als auch bei 3-Net kein Verlust gegenüber dem Singletest zu erkennen 4.26. Betrachtet man die einzelnen VMs, so kann man erkennen dass Xen die zur Verfügung stehende Bandbreite fair auf alle VMs aufteilt 4.27.

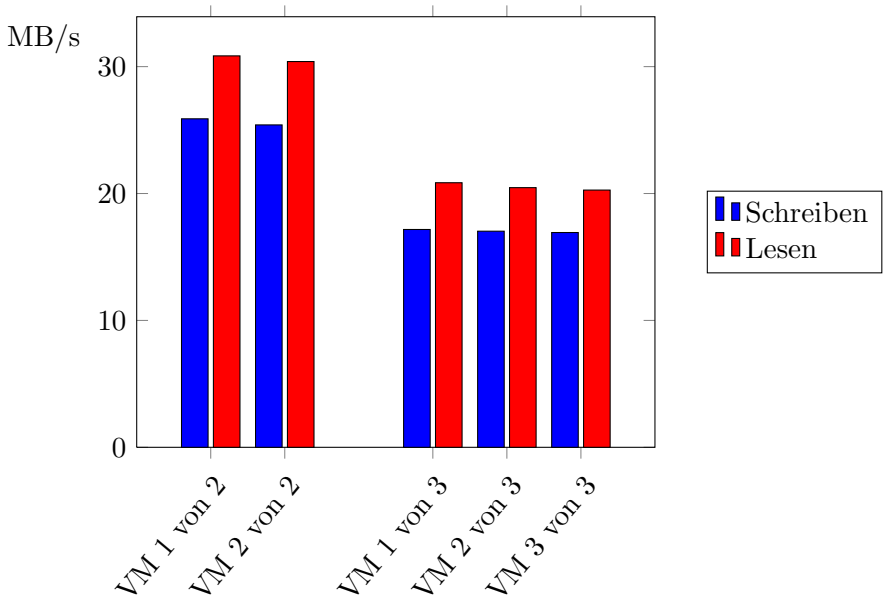
Tabelle 4.26.: Netz parallel - Summe



Disk parallel

Wie bei den parallelen Netztests, muss bei dem Test die Summe der Ergebnisse aller VMs mit dem Singletest verglichen werden.
Dabei fallen gleich mehrere Unterschiede auf. Randomisierte Operationen fallen stark ab. Während der Unterschied zwischen Single und 2-Disk beobachtbar aber nicht zu stark ist, ist der Abfall von 2-Disk zu 3-Disk viel größer. Bei den sequentiellen Operationen zeichnet sich ein anderes Bild ab. Während sequentielles Schreiben von Single zu 2-Disk und von 2-Disk zu 3-Disk konstant und merklich abfällt, steigt sequentielles Lesen bei 2-Disk und 3-Disk gegenüber dem Single wenn auch nur leicht an. Der Abfall lässt sich mit der später noch erwähnten Fairness erklären. Der Vorteil von sequentiellen Operationen gegenüber randomisierten besteht darin, dass der Schreib-/Lesekopf bereits in der Nähe der als nächstes zu schreibenden bzw. lesenden Position steht, es wird also keine Zeit zusätzlich zum Ausrichten des Kopfes benötigt. Springt der Fokus oft zu einer anderen VM, muss mit großer Wahrscheinlichkeit auch der Kopf der Festplatte springen. Umso mehr Fokuswechsel es gibt, umso mehr ähneln sequentielle Operationen also randomisierten. Nun warum ist aber dann

Tabelle 4.27.: Netz parallel - Vergleich der VMs

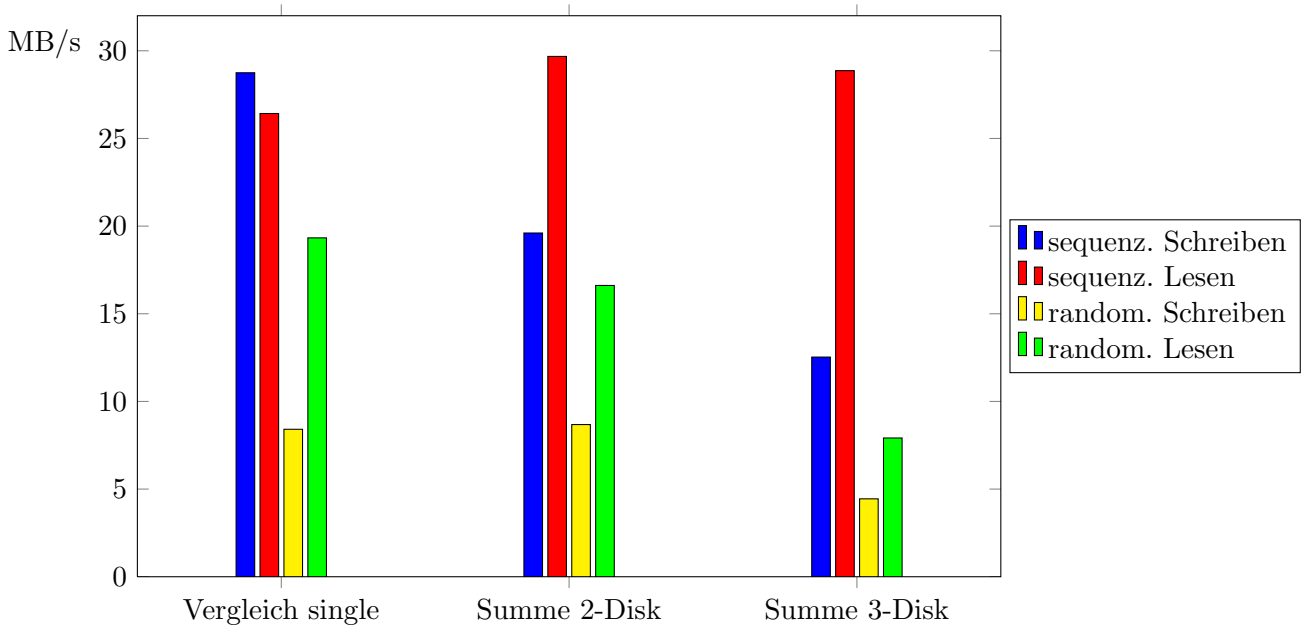


die Rate des sequentiellen Lesens bei den parallelen Tests höher? Es wird an Cachingstrategien des iSCSI Targets und des Initiators liegen. Die jeweils zu lesenden Daten sind zwar höchstwahrscheinlich nicht ganz identisch, sie liegen zeitlich aber garantiert nicht weit auseinander. Die Wahrscheinlichkeit dass manche davon sich noch in einem Festplattencache befinden, sind also recht hoch. Dabei muss man allerdings auch bemerken dass das Caching nicht dafür nicht sonderlich effektiv ist, hier könnte man noch deutlich stärkere Anstiege beobachten 4.28. Sieht man sich die Ergebnisse der einzelnen VMs bei dem 2-Disk und 3-Disk Test an (4.29), fällt auf dass fast alle Werte nahezu fair sind.
Einzig sequentielles Lesen zeigt ein auf den ersten Blick verblüffendes Ergebnis. Bei beiden Tests gibt es eine VM die ein schlechteres Ergebnis als die anderen erzielt hat. Die Anderen (bei 2-Disk ist das logischerweise nur eine VM, bei 3-Disk sind es aber schon beide Anderen) zeigen ein gleich gutes, viel besseres Ergebnis.
Dieses Verhalten lässt sich durch das bereits angesprochene Caching erklären. Eine VM muss die Daten erstmal direkt von der Festplatte lesen, alle anderen können dann (zumindest teilweise) auf die noch im Cache zur Verfügung stehenden Daten zurückgreifen.

Netz bzw. Disk unter Last

Bei der nächsten Kategorie an Tests betrachten wir das das Verhalten unter voller CPU Last.
Bei dem Netztest unter Last fällt auf, dass sowohl beim sequentiellen Schreiben als auch beim Lesen kaum Verluste zu beklagen sind. Die Verluste belaufen sich auf ca. 10%. Gemessen an der Tatsache dass Iometer während dem Test so wenig wie möglich Rechenzeit zugewiesen bekommt, ein vertretbarer Wert. Anders sieht es bei den Leseoperationen aus. Diese brechen um bis zu 30% 4.30 ein.
Unter Last sind nun auch zum ersten Mal beim Netzteil Verluste zu beobachten. Während

Tabelle 4.28.: Disk parallel - Summe



Linux paravirtualisiert unter Xen sowohl bei den Wirkungsgradbenchmarks beim Netztest keine Verluste zeigt (4.9) und auch beim gleichzeitigem Zugriff mehrerer VMs kein Verlust erkennbar ist (4.26), wirkt sich volle CPU Auslastung stark auf die Übertragungsrate aus. Während die Schreibrate um 20% einbricht, sinkt die Leserate sogar um 45% (4.31).

Tabelle 4.29.: Disk parallel - Vergleich der VMs

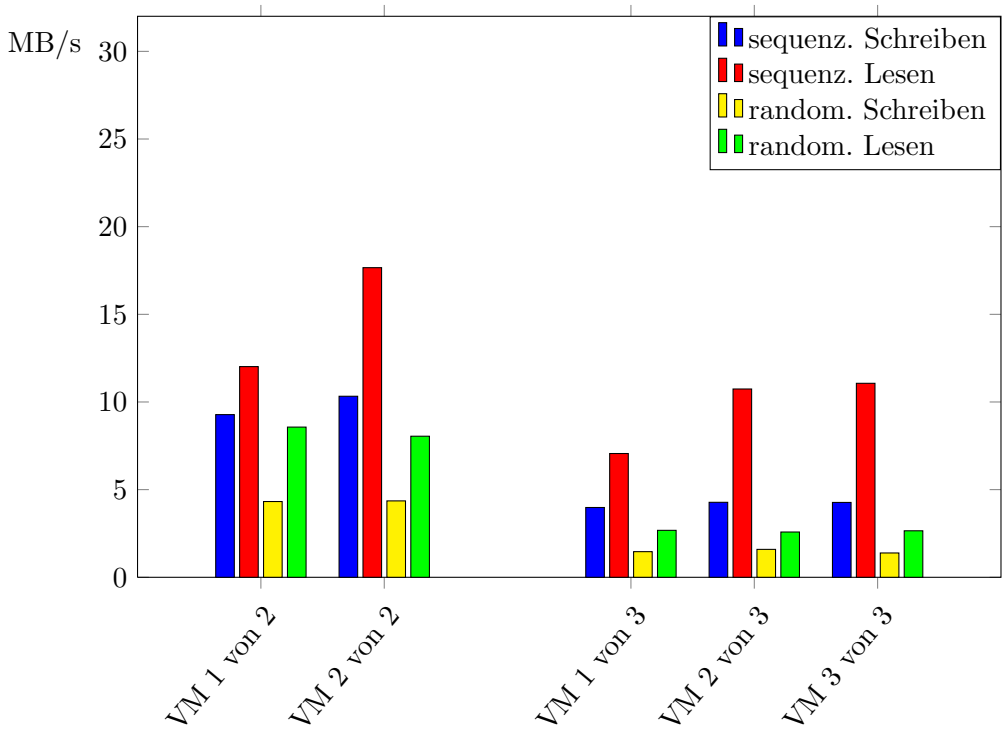


Tabelle 4.30.: Disk unter Last

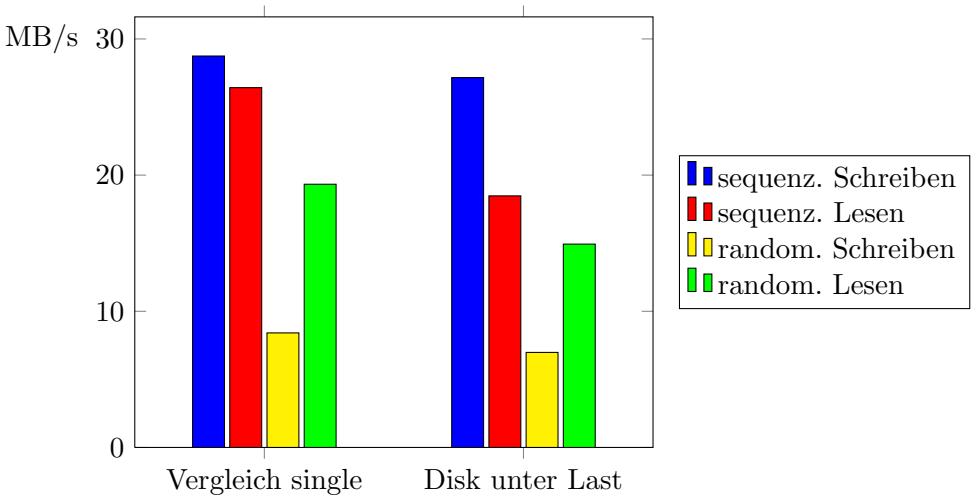
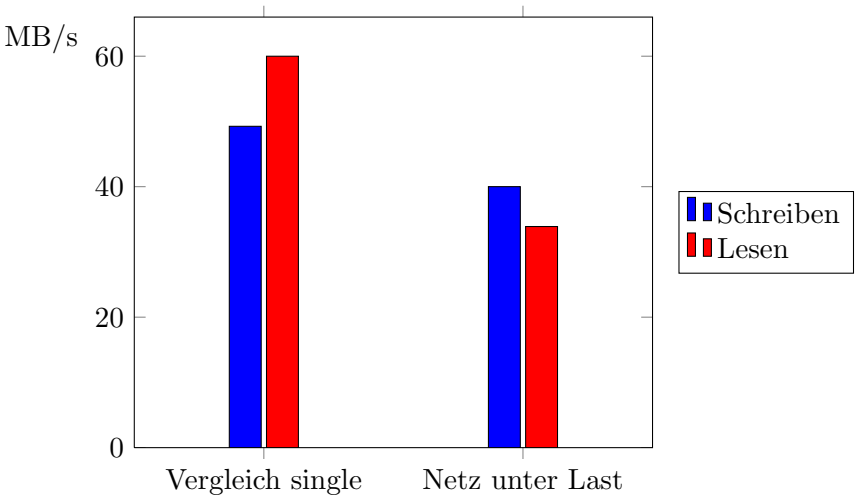


Tabelle 4.31.: Netz unter Last



Netz und Disk parallel

Der Netz- und Disktest ist der größte im Laufe dieser Arbeit durchgeführte Test. Mit ihm wurde versucht, eine gleichzeitige, starke Auslastung von Festplatte und Netz zu simulieren. Sieht man sich die Ergebnisse des Netz- (Teil-) Tests an, fällt von der Tendenz ein ähnliches Verhalten wie bei dem Netztest unter Last (4.31) auf. Sowohl Lese- als auch Schreiberate brechen ein, ein Verhalten dass wie am Netztest (4.26) erkennbar, nicht an dem multiplen Zugriff auf das Netzmedium liegen kann. Ähnlich wie beim Lasttest sinkt die Leserate stärker als die Schreibrate. Beide Verluste sind aber geringer. Während ein Verlust von weniger als 10% beim Schreiben nicht gravierend ist, sinkt die Leserate immerhin um fast 20% (4.32)
Zusätzlich zu dem Verlust in der Summe der übertragenen Daten kann man zum ersten Mal eine ungleiche Verteilung der Übertragungsrate einzelnen VMs beobachten. Dieser Unterschied liegt im Gegensatz zum Disktest in Kapitel 4.1.1 nicht allein am Caching der Festplatte, da auch der Schreibvorgang betroffen ist. Bei dem zweiten Teiltest dieses Benchmarks, dem Disktest, gibt es größere Verluste. Im Vergleich mit dem parallelen Disktest erkennt man ähnliche Ausprägungen. Die Rate der sequentiellen Operationen fällt allerdings noch weiter ab und ist beim Schreiben schon in etwa auf dem Wert der randomisierten Operationen abgefallen. Häufige Fokuswechsel der VMs lassen diese Operationen randomisierten Operatationen ähneln, da der Schreib-/Lesekopf nach einem Wechsel nicht an der vorherigen Stelle fortfahren kann und sich neu ausrichten muss. Das Verhalten das auch schon beim normalen parallelen Disktest zu beobachten war, wird durch und zusätzliche Prozesswechsel innerhalb der VMs, die die Dauer bis zum Erneuten Lese- bzw. Schreibeversuch noch erhöhen, noch verstärkt. Die Rate der Leseoperationen bricht allerdings nicht so stark ein und liegt noch in etwa 70% über dem randomisiertem Wert. Das Caching scheint also diese Rate noch weiterhin zu verbessern, auch wenn die bereits erwähnten, häufigen Fokuswechsel die Effektivität senken. Die Werte der randomisierten Operationen sinken beide im Vergleich mit 3-Disk nicht ein. Der randomisierte Zugriff ist in diesem Fall also der Flaschenhals für die Übertragungsrate.
Betrachtet man die einzelnen Werte der VMs, erkennt man zum ersten Mal eine ungleiche Verteilung beim Netztest (4.33), während die Verteilung beim Disktest erstaunlicherweise fair ist (4.35). Einzig beim sequentiellen Lesen ist eine VM gegenüber den beiden anderen benachteiligt. Wie beim parallelen Disktest 4.2.2 bereits erklärt, lässt sich das auf das Caching der iSCSI Platte zurückführen. Erkennbar ist hier allerdings auch, dass die Effektivität des Cachings weiter zurückgegangen ist, da der Unterschied der benachteiligten VM gegenüber den beiden anderen geringer als beim reinen Disktest ist.

Tabelle 4.32.: Netz Summe aus “Disk und Netz”

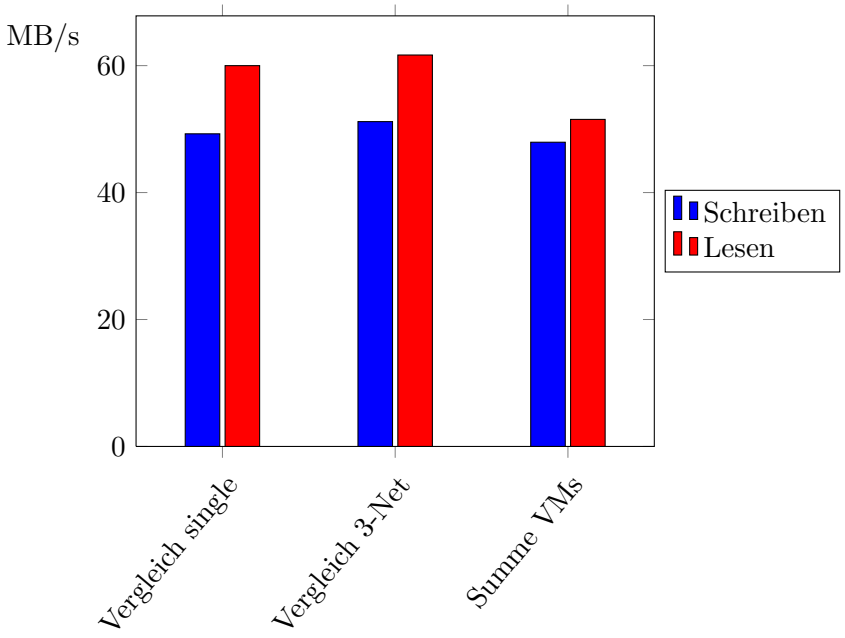


Tabelle 4.33.: Netz aus “Disk und Netz” - Vergleich der VMs

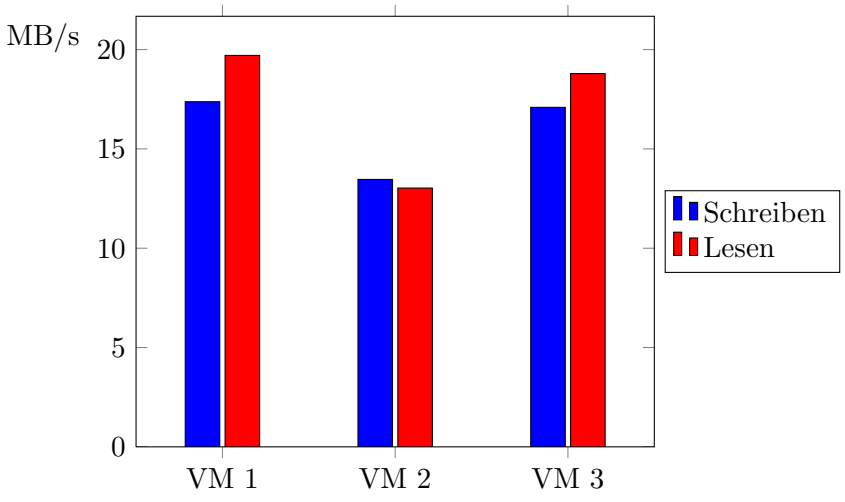


Tabelle 4.34.: Disk Summe aus “Disk und Netz”

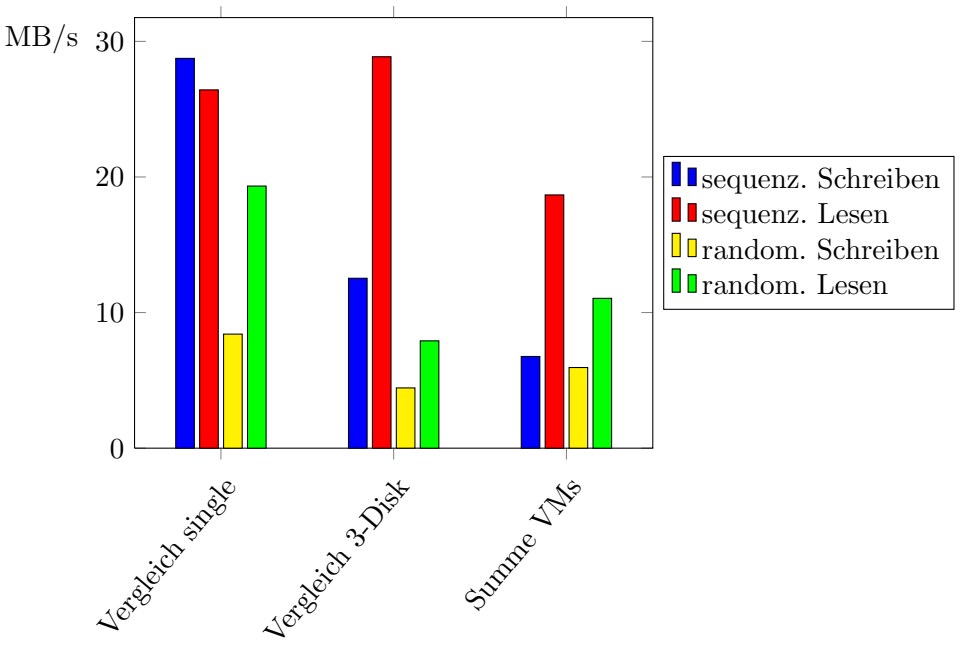
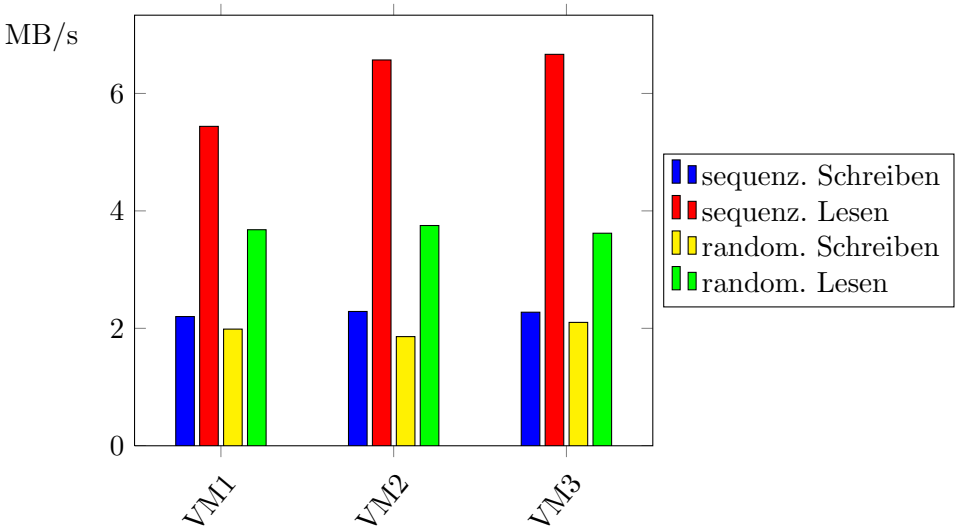


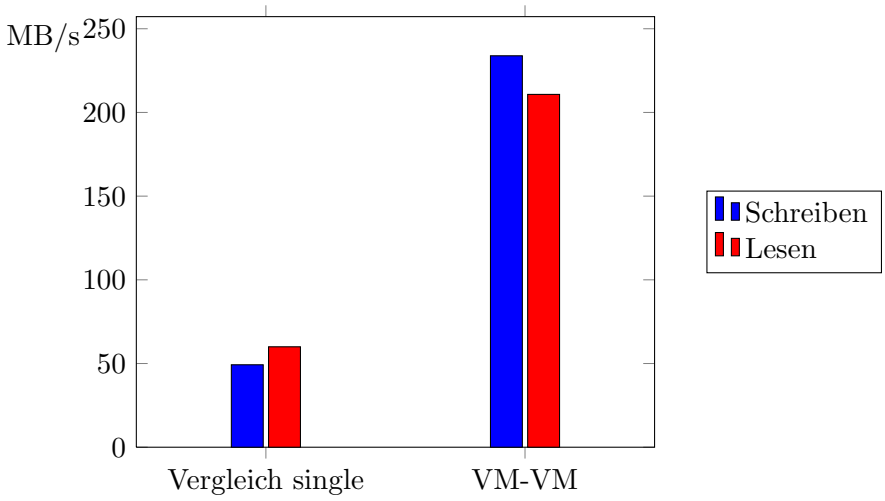
Tabelle 4.35.: Disk aus “Disk und Netz” - Vergleich der VMs



Inter VM

Der letzte durchgeführte Test ist ein alleinstehender Test, der nicht, oder nur schwer, mit einem Wirkungsgradbenchmark oder sogar nativem Test verglichen werden kann. Hier geht es um direkten Netzverkehr zwischen zwei VMs auf dem gleichen Host. In der Theorie darf dazu kein Peripheriegerät angesteuert werden, die Kommunikation sollte rein über Software erfolgen. Da bei dem Test mit Iometer nach dem Empfangen der Daten keine Schreiboperationen auf die Festplatte erfolgen, sondern nur überprüft wird ob die Daten angekommen sind, sollte die Übertragungsrate weit über die Rate von 1000BaseT ansteigen.
Tatsächlich beobachtet man einen Anstieg auf einen Wert, der bescheinigt dass die Daten nicht an das angeschlossene Gigabit angelegt wurden 4.36.

Tabelle 4.36.: Netz von VM zu VM



5. Fazit und Ausblick

Im letzten Kapitel soll kurz erläutert werden, welche Ziele erreicht wurden, welches Fazit man aus den Tests schließen kann und welche weiterführenden Arbeiten an diese Arbeit anknüpfen und sie erweitern können.

5.1. Erreichte Ziele

Im Rahmen dieser Arbeit wurde aufgezeigt, wie hoch der Verlust der Virtualisierung ist. Anders fomuliert kann man auch sagen, wie groß der Einfluss der Virtualisierung auf die Leistung der Systeme in den Tests ist.
Dabei wurde ebenfalls aufgezeigt, wo etwaige Schwächen von Xen liegen. Beispielsweise wurde ersichtlich dass I/O Operationen im Allgemeinen anfälliger für Einbrüche sind, als CPU Operationen oder Zugriffe auf den Hauptspeicher.

5.2. Fazit

Wichtig ist bei diesen Beobachtungen allerdings auch, dass diese Arbeit nicht eigenständig betrachtet wird, sondern dass die drei weiteren Arbeiten, die analog mit dieser entstanden sind, ebenfalls mit betrachtet werden, um die verschiedenen Hypervisor miteinander zu vergleichen. Auf Xen allein bezogen lässt sich feststellen, dass der Wirkungsgrad bei Vollvirtualisierung ungleich geringer ist, als bei Paravirtualisierung. Sollte es sich bei dem Gastsystem um Linux handeln, sollte man also darauf achten es zu paravirtualisieren.
Bei Windows als Gastsystem, welches sich ja nur vollvirtualisieren lässt, fällt auf, dass die "PV Treiber" keine zufriedenstellenden Ergebnisse liefern. Sie sind im aktuellen Zustand also nicht für einen Praxisbetrieb geeignet. Ohne die PV Treiber liefern Windows Gäste allerdings auch keine akzeptablen Ergebnisse, unter anderem aus dem erwähnten Grund. Deshalb ist es zu empfehlen auf eine andere Lösung als Xen zurückzugreifen, wenn man Windows Server virtualisieren möchte.

5.3. Ausblick

In Anlehnung und als Erweiterung und Fortführung dieser Arbeit sind einige Folgearbeiten möglich.
Da Xen mit paravirtualisiertem Linux die besten Ergebnisse liefern konnte, wäre es interessant es mit einem komplett anderem System zu vergleichen. OpenVZ verfolgt durch die Betriebssystemvirtualisierung ein anderes Konzept und versucht dabei die Virtualisierung so gering wie möglich zu halten, hat dabei allerdings andere Nachteile.
Ebenso wäre es interessant, die hier angewendeten Konzepte auf weitere Virtualisierer zu übertragen. Als Beispiel kann hier KVM erwähnt werden, das in letzter Zeit immer mehr an

Bedeutung gewinnt.
Ebenso können noch eine Reihe weiterer Testszenarien erstellt werden. Im Rahmen dieser Arbeit war die Zeit und der dadurch resultierende Arbeitsaufwand beschränkt, weshalb man sich auf eine sortierte Auswahl an Testszenarien beschränken musste. Gerade bei den Skalierungstests lassen sich leicht weitere Konfigurationen erstellen.
Ein weiterer beschränkender Faktor war gerade bei den Skalierungstests die verwendete Hardware. Deshalb wäre es natürlich besonders interessant, wenn die hier angewendeten Verfahren in einer noch realeren Umgebung durchgeführt werden würden.
Eine weitere Veränderung gegenüber diesem Szenario wäre, weitere Hardwarekomponenten zu untersuchen. Die Implementierung einer "Secure Virtual Machine" von Intel trägt den Namen "Intel VT", es ist damit das Gegenstück zu AMD-V. Ebenso besitzen neuere CPUs von Intel einen gemeinsamen Level 2 Cache für alle Kerne. Diese Beispiele sind sicher nur ein Bruchteile dessen, was es an nicht untersuchten Hardwarevarianten gibt, die Einfluss auf die Testergebnisse haben können. Gegen Ende dieser Arbeit wurde Xen 4.0 mit vielen neuen interessanten Eigenschaften veröffentlicht. Die Frage wie sich Xen in der Zukunft weiter entwickeln wird, ist natürlich auch unter dem Aspekt der Effizient sehr interessant. Auch ein Vergleich von KVM mit dieser neuen Xen Version ist gerade bei der Frage wie es bei der Linuxvirtualisierung in der Zukunft weitergehen wird, sehr wichtig.
Eine weitere Art von weiterführenden Tests wäre die Untersuchung der beobachteten Werte in tatsächlicher realer Umgebung. Dazu werden Anwendungsbenchmarks beispielsweise auf Datenbankserver oder ähnlichen in der Praxis verwendeten Diensten angewendet.
Zu guter Letzt sei noch erwähnt dass den VMs bis jetzt immer nur eine vCPU zugewiesen wurde. Paralleltests wie beispielsweise der Cachetest von RAMspeed, aber auch die parallelen Disk- und Netztests sowie selbstverständlich der CPU Test kann noch unter Mehrkernbetrieb in der VM getestet werden.
Wie man an Hand dieser Aufzählung sieht, gibt es eine Reihe von möglichen Fortführungen dieser Tests, wobei die hier erwähnten Ergänzungen sicher nur ein Bruchteil des Möglichen sind.

A. Anhang

Netz u. Disk (Disk) (MB/s)	sequenz. Schreiben	sequenz. Lesen	random. Schreiben	random. Lesen
Vergleich single	28.75	26.42	8.41	19.33
Vergleich 3-Disk	12.53	28.87	4.44	7.91
Summe Vms	6.76	18.68	5.95	11.05
VM1	2.20	5.44	1.99	3.68
VM2	2.29	6.57	1.86	3.75
VM3	2.27	6.67	2.10	3.62
Netz u. Disk (Netz) (MB/s)	Schreiben	Lesen		
Vergleich single	49.26	60		
Vergleich 3-Net	51.18	61.67		
Summe Vms	47.93	51.53		
VM 1	17.38	19.71		
VM 2	13.47	13.03		
VM 3	17.09	18.79		

Disk Linux (MB/s)	sequenz Schreiben	sequenz Lesen	random Schreiben	random Lesen
32-Bit nativ	27.98	30.46	13.82	20.03
64-Bit nativ	26.18	30.71	13.80	20.04
32-Bit	30.43	26.38	8.55	19.42
32-Bit AMD-V	28.75	26.42	8.41	19.33
32-Bit HVM	10.61	27.43	7.51	19.11
64-Bit	27.94	25.99	8.60	19.74
64-Bit AMD-V	27.12	26.4	8.11	18.68
64-Bit HVM	27.12	26.4	8.11	18.68
Disk Windows (MB/s)	sequenz Schreiben	sequenz Lesen	random Schreiben	random Lesen
32-Bit Nativ	28.75	23.06	12.35	18.26
64-Bit Nativ	28.79	23.09	12.34	18.11
32-Bit	29.41	26.85	8.42	18.27
32-Bit Tools	30.13	26.69	8.37	18.38
64-Bit	29.99	27.42	8.60	18.62
64-Bit Tools	29.95	27.11	8.51	18.96
Disk Windows (% CPU)	sequenz. Schreiben	sequenz. Lesen	random. Schreiben	random. Lesen
32-Bit Nativ	3.82	5.81	2.02	5.36
64-Bit Nativ	3.75	5.66	2.09	4.67
32-Bit	19.26	16.99	4.45	9.63
32-Bit Tools	1.76	0.46	0.18	0.34
64-Bit	17.87	15.69	2.98	7.83
64-Bit Tools	0	0	0	0

VM-VM (MB/s)	Schreiben	Lesen
Vergleich single	49.26	60
VM-VM	233.85	210.78

2-Disk (MB/s)	sequenz. Schreiben	sequenz. Lesen	random. Schreiben	random. Lesen
Vergleich single	28.75	26.42	8.41	19.33
Summe Vms	19.60	29.68	8.68	16.61
VM1	9.28	12.02	4.32	8.57
VM2	10.33	17.67	4.36	8.05
3-Disk (MB/s)	sequenz. Schreiben	sequenz. Lesen	random. Schreiben	random. Lesen
Vergleich single	28.75	26.42	8.41	19.33
Summe Vms	12.53	28.87	4.44	7.91
VM1	3.99	7.06	1.47	2.68
VM2	4.28	10.74	1.59	2.59
VM3	4.28	11.07	1.39	2.66

2-Net (MB/s)	Schreiben	Lesen
Vergleich single	49.26	60
Summe Vms	51.35	61.31
VM1	25.89	30.85
VM2	25.41	30.40
3-Net (MB/s)	Schreiben	Lesen
Vergleich single	49.26	60
Summe Vms	51.18	61.68
VM1	17.17	20.85
VM2	17.03	20.47
VM3	16.93	20.28

Disk unter Last (MB/s)	sequenz. Schreiben	sequenz. Lesen	random. Schreiben	random. Lesen
Vergleich single	28.75	26.42	8.41	19.33
Disk unter Last	27.17	18.48	6.98	14.93

Netz unter Last (MB/s)	Schreiben	Lesen
Vergleich single	49.26	60
Netz unter Last	40.01	33.90

Linpack parallel (Sekunden)	
single	214.2
2 parallel	286.52
3 parallel	429.47

Linpack Linux (Sekunden)	
32-Bit Nativ	20.89
64-Bit Nativ	20.53
32-Bit PV	21.16
32-Bit PV AMD-V	21.42
32-BIt HVM	20.91
64-Bit PV	20.41
64-Bit PV AMD-V	20.25
64-Bit HVM	21.05
Linpack Windows (Sekunden)	
32-Bit Nativ	44.5
64-Bit Nativ	45.51
32-Bit VM	44.79
32-BIt VM Tools	44.79
64-Bit VM	46.08
64-Bit VM Tools	45.69

Net Win (MB/s)	Schreiben	Lesen
32-Bit Nativ	52.5	64.6
64-Bit Nativ	52.67	64.56
32-Bit	12.93	15.58
32-Bit Tools	10.09	63.24
64-Bit	36.24	43.84
64-Bit Tools	6.83	70.77
Net Windows (% CPU)	Schreiben	Lesen
32-Bit Nativ	1.95	7.53
64-Bit Nativ	1.91	7.26
32-Bit	99.99	99.86
32-Bit Tools	4.02	15.2
64-Bit	99.97	99.42
64-Bit Tools	0	0
Net Linux (MB/s)	Schreiben	Lesen
32-Bit Nativ	52.97	61.17
64-Bit Nativ	48.91	61.19
32-Bit PV	49.26	60
32-Bit PV*	50.11	59.80
32-Bit HVM	13.08	15.85
64-Bit PV	51.16	59.95
64-Bit PV*	51.53	59.83
64-Bit HVM	13.15	16.1

RAMspeed parallel Summe (MB/s)	L1 Schreiben	L2 Lesen	L2 Schreiben	L2 Lesen	RAM Schreiben	RAM Lesen
single	17127.32	14538.85	5758.99	6637.38	2414.73	3122.15
2 Vms	32548.20	28153.50	11405.23	13035.16	3566.71	5359.60
3 Vms	35980.61	26756.78	9458.97	11784.89	3382.27	4907.25
RAM parallel Average (MB/s)	L1 Schreiben	L1 Lesen	L2 Schreiben	L2 Lesen	RAM Schreiben	RAM Lesen
Vergleich Single	17127.32	14538.85	5758.99	6637.38	2414.73	3122.15
VM 1/2	16253.24	14232.71	5973.02	6807.51	1784.92	2693.92
VM 2/2	16294.96	13920.80	5432.21	6227.65	1781.78	2665.67
VM 1/3	13301.14	8555.66	3349.90	4027.58	1124.97	1655.80
VM 2/3	11685.76	8916.35	3231.07	3865.53	1127.45	1636.75
VM 3/3	10993.71	9284.77	2878.01	3891.77	1129.85	1614.70
RAMspeed single Linux (MB/s)	L1 Write	L1 Read	L2 Write	L2 Read	RAM Write	RAM Read
32-Bit Nativ	17146.53	14554.83	5635.32	6608.65	2446.68	3108.04
64-Bit Nativ	28832.59	36831.66	6613.25	7261.21	2875.12	3403.24
32-Bit PV AMD-V	17127.32	14538.85	5758.99	6637.38	2414.73	3122.15
32-Bit PV	17120.63	14537.84	5802.97	6823.69	2419.65	3124.96
VM 32 HVM	17064.68	14503.53	5802.99	6650.97	2444.76	3105.91
64-Bit PV AMD-V	28815.99	36787.3	7671.85	7795.4	2960.09	3390.22
64-Bit PV	28788.65	36787.9	6821.33	7175.68	2922.51	3390.69
64-Bit HVM	28719.25	36715.81	6889.24	7299.54	2892.94	3434.31
RAMspeed single Windows (MB/s)	L1 Write	L1 Read	L2 Write	L2 Read	RAM Write	RAM Read
32-Bit Nativ	18829.044	19255.9	6382.28	7331.68	2440.52	3145.88
64-Bit Nativ	18721.47	19242.95	6387.08	7298.16	2402.26	3135.22
32-Bit	18623.48	19163.23	6354.01	7242.95	2483.63	3124.92
32-Bit Tools	18398.87	19199.7	6353.76	7269.34	2509.28	3130.2
64-Bit	20307.89	20879.6	6907.38	7902.44	2664.54	3401.67
64-Bit Tools	20397.56	20974.15	6948.38	7938.96	2697.33	3421.21

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[AMD05] Advanced Micro Devices, Inc.: *AMD64 Virtualization Codenamed ?Pacifica? Technology; Secure Virtual Machine Architecture Reference Manual*, may 2005. http://www.mimuw.edu.pl/~vincent/lecture6/sources/amd-pacifica-specification.pdf.

[Ent] Enterprises, Alasir: *The Alasir Licence*. http://alasir.com/licence/TAL.txt.

[ibm05] *IBM Systems Virtualization Version 2 Release 1*, Dec 2005. http://publib.boulder.ibm.com/infocenter/eserver/v1r2/topic/eicay/eicay.pdf.

[Kra07] Krapf, Andreas: *XEN Memory Management (Intel IA-32)*, Oct 2007. http://www-sop.inria.fr/everest/personnel/Andres.Krapf/docs/xen-mm.pdf.

[Mea] Meadowcourt: *meadowcourt.org - /downloads/*. http://meadowcourt.org/downloads/.

[MSD] MSDN: *Microsoft TechNet - DisableTaskOffload*. http://technet.microsoft.com/en-us/library/cc959732.aspx.

[Pal] Palivan, Oriana: *Xen Networking*. http://wiki.xensource.com/xenwiki/XenNetworking#head-1fc8531de90f02e42e6fdccc30232cf8f0254ad0.

[pcd] pcdrews: *IOmeter Patch*. http://sourceforge.net/tracker/index.php?func=detail&aid=1244848&group_id=40179&atid=427254.

[PUGa] Penguin User Group, PUG: *Xen-Installation*. http://www.pug.org/mediawiki/index.php/Xen-Installation-Seite-4#Credit_Scheduler.

[PUGb] Penguin User Group, PUG: *Xen-Installation*. http://www.pug.org/mediawiki/index.php/Xen-Installation-Seite-4#Arbeitsspeicherverwaltung.

[RMH] Rhett M. Hollander, Paul V. Bolotoff: *RAMspeed, a cache and memory benchmarking tool*. http://alasir.com/software/ramspeed/.

[Spe05] Spector, Stephen: *How are Hypervisors Classified?*, Jan 2005. http://www.xen.org/files/Marketing/HypervisorTypeComparison.pdf.

[Sysa] Sysinternals: *Sync v2.0*. http://technet.microsoft.com/de-de/sysinternals/bb897438.aspx.

[Sysb] Sysinternals: *Windows Sysinternals*. http://technet.microsoft.com/de-de/sysinternals/default.aspx.

[ubua] *Package: linux-image-2.6.24-24-xen.* `http://packages.ubuntu.com/hardy-updates/linux-image-2.6.24-24-xen`.

[ubub] *Package: ubuntu-xen-server.* `http://packages.ubuntu.com/jaunty/ubuntu-xen-server`.

[ubuc] *Package: xen-hypervisor-3.3.* `http://packages.ubuntu.com/jaunty/xen-hypervisor-3.3`.

[vmw] *Virtuelle Mschinen, virtueller Server, virtuelle Infrastruktur.* `http://www.vmware.com/de/virtualization/virtual-machine.html`.