Optimierung des Wirkungsgrades virtueller Infrastrukturen

Dissertation

an der

Fakultät für Mathematik, Informatik und Statistik
der
Ludwig-Maximilians-Universität München

vorgelegt von

Tobias Lindinger

Tag der Einreichung: 10. Dezember 2009

Optimierung des Wirkungsgrades virtueller Infrastrukturen

Dissertation

an der

Fakultät für Mathematik, Informatik und Statistik
der
Ludwig-Maximilians-Universität München

vorgelegt von

Tobias Lindinger

Tag der Einreichung: 10. Dezember 2009
Tag des Rigorosums : 09. Februar 2010

1. Berichterstatter: **Prof. Dr. Heinz-Gerd Hegering**, Universität München
2. Berichterstatter: **Prof. Dr. Arndt Bode**, Technische Universität München

Zusammenfassung

Virtualisierungstechniken erfreuen sich immer größerer Beliebtheit in vielen Bereichen der Informatik. Ursprünglich wiederentdeckt mit dem Ziel Ressourcen und Dienste zu konsolidieren, dienen Virtualisierungsansätze heute als Grundlage für moderne Grid- und Cloud-Computing-Infastrukturen und werden damit auch im Bereich des Hochleistungsrechnens eingesetzt. Derzeit existieren keine objektiven und systematischen Analysen bezüglich des Wirkungsgrades von Virtualisierungsansätzen, Techniken und Implementierungen, obwohl sie von vielen großen Rechenzentren weltweit eingesetzt und produktiv betrieben werden. Alle existierenden, modernen Hostvirtualisierungsansätze setzen derzeit auf eine Softwareschicht, die sich je nach Virtualisierungstyp zwischen Hardware und Gast-Betriebssystem bzw. zwischen Host- und Gast-Betriebssystem befindet. Eine Anwendung in einer virtuellen Maschine ist somit nicht mehr nur von der Leistung des physischen Systems abhängig, sondern ebenfalls von der Technologie des eingesetzten Virtualisierungsproduktes und nebenläufigen virtuellen Maschinen. Je nach Anwendungstyp kann es daher sinnvoll sein, einen anderen Virtualisierungsansatz zu wählen und auf den Typ der nebenläufigen virtuellen Maschinen zu achten, um den Wirkungsgrad eines lokalen Systems sowie den der globalen Infrastruktur zu optimieren.

Um dieses Ziel zu erreichen, werden in einem zweistufigen Ansatz zunächst theoretisch Virtualisierungsansätze analysiert und Parameter identifiziert, deren Einfluss auf den Wirkungsgrad in einem zweiten Schritt empirisch quantifiziert wird. Für die Durchführung dieser quantitativen Analyse ist eine Anpassung verbreiteter Leistungsmaße, wie z.B. Durchsatz und Antwortzeit, für den Kontext der Virtualisierung erforderlich, da sie sich klassisch gesehen auf das Betriebssystem einer Maschine beziehen, eine virtuelle Maschine jedoch von der Architektur her eher einer klassischen Anwendung entspricht. Die Messung dieses Leistungsmaßes in virtuellen Umgebungen stellt eine weitere Herausforderung dar, da Zeitmessung in virtuellen Maschinen aufgrund von Scheduling durch den Hypervisor generell fehlerbehaftet ist und somit alternative Messmethoden konzipiert werden müssen. Basierend auf den durchgeführten Analysen und Messungen wird anschließend ein Leitfaden entwickelt, der dabei hilft, die zur Virtualisierung einer Infrastruktur benötigten Ressourcen qualitativ sowie quantitativ abzuschätzen und eine Verteilung der virtuellen Maschinen anhand ihres charakteristischen Ressourcenbedarfes auf physische Systeme vorzunehmen, so dass vorhandene physische Ressourcen optimal ausgenutzt werden können. Die Automatisierung des erstellten Leitfadens durch die Entwicklung und prototypische Implementierung eines globalen Ressourcen-Schedulers auf der Basis eines gewichteten Constraint Solvers rundet die Arbeit ab. Der verwendete Ansatz besitzt zwar eine theoretisch exponentielle Laufzeitkomplexität, liefert in der Praxis aufgrund einer entwickelten Greedy-Heuristik jedoch bereits nach extrem kurzer Laufzeit herausragende Ergebnisse. Die optimierten Verteilungen lassen sich anschließend mittels weniger Live Migration realisieren, da bereits bei der Berechnung einer Verteilung auf deren räumliche Nähe zur bestehenden Verteilung geachtet wird.

Abstract

Virtualization techniques are enjoying increasing popularity in many areas of computer science. Though originally revisited with the aim to consolidate resources and services, today virtualization approaches are employed as a basis for modern grid and cloud computing infrastructures, and thus finding their way into the field of high- performance computing. In spite of the extensive use of virtualization technology, a quantitative, objective analysis of the efficiency of different virtualization approaches, techniques and implementations is still lacking. All existing modern approaches for virtualizing systems currently rely on a software layer, which is located between hardware and host operating system or between host and guest operating system depending on the virtualization technique. An application in a virtual machine is no longer dependent only on the performance of the physical system, but also on the technology of the deployed virtualization product and concurrent virtual machines. Depending on the type of application and the type of concurrent virtual machines used, it may be beneficial to employ a different virtualization technique in order to increase efficiency of a local system and the global infrastructure.

To achieve this objective, virtualization approaches are analyzed theoretically as a first step, before identifying parameters and quantifying their influence empirically as a second step. For the implementation of the quantitative analysis an adaptation of common performance measures, such as throughput and response time is required for the context of virtualization, as these metrics are classically related to the operating system of a machine, a virtual machine, however, shows more similiarity to a classic application. Measurements of performance in a virtual environment present additional challenges, because timing is generally flawed in virtual machines because of scheduling by the hypervisor, thus making necessary alternative measurement methods. Based on the analysis of virtualization approaches and on measurement results, this work presents a guide that helps to estimate the resources that are required for the virtualization of an given infrastructure qualitatively and quantitatively and thus to make a distribution of virtual machines based on their typical resource requirements on physical systems, so that existing physical resources can be utilized optimally. The guide is complemented by the prototype of a global resource scheduler based on a weighted constraint solver. While the base algorithm does exhibit exponential time complexity it was possible to apply a greedy heuristic and to achieve useful results after an extremely short term. The optimized distributions can be achieved by means of minimum numbers of virtual machine migrations, since the proximity to the existing distribution has been already taken to the calculation of the optimized distribution.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Inhalt

Trend zur Virtualisierung durch Green-IT und Konsolidierung

Zurzeit entsteht in vielen Bereichen der Gesellschaft ein Trend zum Umweltschutz. So entwickeln zum Beispiel Automobilfirmen schadstoffarme Autos mit geringem Verbrauch, Städte führen Feinstaubplaketten für Autos ein und Verbraucher werden in medienwirksamen Aktionen zum Stromsparen aufgefordert. Dieser Trend ist auch in der Informatik nicht zu übersehen. Unter dem Schlagwort „Green-IT" vermarkten derzeit etliche Hersteller Lösungen zur Einsparung von Energie in größeren Rechenzentren, allen voran die Hersteller von Virtualisierungslösungen.

Ein durchschnittlicher Server in einem typischen Rechenzentrum - etwa im Bereich Webhosting - stellt aus Sicherheits- oder Kompatibilitätsgründen häufig nur einen Dienst bereit und ist daher nur zu einem sehr geringen Prozentsatz ausgelastet. Jeder dieser Server muss in der Regel von seiner Leistungsfähigkeit etwas überdimensioniert werden, um kurzzeitige Lastspitzen abfangen zu können. Gleichzeitig verbraucht ein Server, der nur zu 20% ausgelastet ist, aber ca. 80% der Energie, die er unter Voll-Last benötigen würde. Diese Situation ist sowohl ökologisch als auch ökonomisch nicht zufriedenstellend und ist ein Grund für den Einsatz von Lösungen zur Servervirtualisierung in Rechenzentren.

Mit Hilfe von Virtualisierungstechnologien ist es möglich, mehrere virtuelle Server zur selben Zeit, aber unabhängig voneinander, auf einem physischen Server zu betreiben. Die Folgen sind eine wesentlich höhere und damit auch effizientere Auslastung der physischen Server sowie eine Homogenisierung der Infrastruktur, die von den Herstellern der Virtualisierungslösungen unter dem Schlagwort „Konsolidierung" vermarktet wird. In Summe ist im virtualisierten Fall zusätzlich eine wesentlich geringere Anzahl an physischen Maschinen notwendig, was wiederum die Stellfläche und Kühlleistung im Rechenzentrum reduziert und damit neben den ökologischen Gesichtspunkten auch eine finanzielle Einsparung bedeutet. Ein weiterer Aspekt, den die Homogenisierung der Infrastruktur mit sich bringt, ist ein vereinfachtes Management mit erweiterter Funktionalität, da alle Server personalschonend über eine einheitliche Managementplattform administriert werden

können und Hochverfügbarkeitslösungen sowie Lastausgleichsmechanismen in den Management-Applikationen in der Regel enthalten sind.

1.1 Motivation und Zielsetzung

Kapazitätsplanung zum Migrationszeitpunkt ist aufwendig oder gar unmöglich

Auf den ersten Blick scheint der Einsatz von Virtualisierungslösungen im Rechenzentrum nur Vorteile zu bieten. Tatsächlich können solche Infrastrukturen mit der heutigen Technik weitestgehend problemlos betrieben werden, allerdings ist die Hürde der Migration hin zu einer virtuellen Infrastruktur relativ hoch. In der Regel eignet sich die vorhandene Infrastruktur nicht zum Betrieb einer virtuellen Infrastruktur, da nur wenige, aber sehr leistungsfähige Server benötigt werden, während in der Praxis meist viele, aber zu leistungsschwache Systeme vorhanden sind. Es muss also zunächst geeignete Hardware angeschafft werden. Leider existieren bislang keine allgemeinen Anhaltspunkte, die einen systematisch begründeten und zuverlässigen Schluss auf die Anzahl und Dimension der zu beschaffenden Systeme erlauben würde. Zwar existieren herstellerspezifische Tools, die die vorhandene Infrastruktur über Monate hinweg überwachen und protokollieren und anschließend vom Hersteller auf Grund seiner Erfahrungen ausgewertet werden können, allerdings liegen die Auswertungskriterien nicht öffentlich vor. Ein Vergleich mit anderen Virtualisierungslösungen ist daher kaum praktikabel, da für jedes zu evaluierende Produkt mehrere Monate Messung erforderlich wären. Aus Kostengründen ist man in der Regel an einer minimalen Infrastruktur interessiert, die gerade noch Spielraum für Hochverfügbarkeit bietet. Wählt man die Infrastruktur jedoch zu klein, ist sie überlastet und kann ihre Aufgabe nicht mehr zweckmäßig erfüllen.

Zielsetzung dieser Arbeit

Ziel dieser Arbeit ist es, auf theoretische und empirische Art und Weise Parameter zu identifizieren, welche die Leistungseigenschaften von virtuellen Infrastrukturen beeinflussen. Hierbei werden zunächst statische Parameter wie die Wahl der Virtualisierungslösung sowie die Anzahl und Art der zu virtualisierenden Systeme hinsichtlich des Ressourcenbedarfs und ihres Wirkungsgrades analysiert. Anhand dieser Untersuchungen sollen anschließend Anhaltspunkte erarbeitet werden, die eine Ressourcenplanung zum Migrationszeitpunkt vereinfachen beziehungsweise überhaupt erst ermöglichen. In einem zweiten Schritt werden dynamische Einflussfaktoren für eine effektive Konfiguration der virtuellen Maschinen untersucht. Die Ergebnisse dieses Schrittes sollen anschließend dazu dienen, eine effiziente Ressourcenverteilung unter den virtuellen Maschinen sowie eine sinnvolle Verteilung von virtuellen Maschinen auf physische Server zu errechnen und gegebenenfalls auch umzusetzen. Diese Grundlagen dienen anschließend der Entwicklung eines praxistauglichen Ressourcen-Schedulers.

1.2 Fragestellungen

Untersuchung des Leistungsbegriffes im Virtualisierungskontext

Um die soeben beschriebenen Ziele dieser Arbeit erreichen zu können, ist es erforderlich das Leistungsverhalten diverser Virtualisierungslösungen sowohl theoretisch als auch empirisch zu analysieren. Grundlage für jede Art der Leistungsbetrachtung ist dabei ein klar definierter Leistungsbegriff im Umfeld der Virtualisierung. Für den klassischen Fall des Leistungsbegriffes kann auf bestehende Literatur zurückgegriffen werden. Im Falle der Virtualisierung müssen einige Aspekte jedoch überdacht oder ergänzt werden. So bieten sich zum Beispiel, was die Messbarkeit der Leistung in virtuellen Maschinen angeht, mehrere Verfahren an, während die Techniken im klassischen Fall in ihrer Vielfalt beschränkt sind. Auch die Definition von Metriken zur Leistungsmessung ist nicht wie im klassischen Fall auf absolute Werte beschränkt, sondern muss Messwerte erlauben, die relativ zum Hostsystem sind. Diese und andere Aspekte müssen zunächst den Gegebenheiten der Virtualisierung angepasst werden, um als Arbeitsgrundlage für den Rest der Arbeit dienen zu können.

Analyse existierender Virtualisierungsansätze

Ein weiterer zu untersuchender Aspekt sind die gängigen Virtualisierungsansätze selbst. Es existieren zurzeit diverse Ansätze zur Schaffung virtueller Umgebungen, die sich in ihrer Art der technischen Realisierung stark unterscheiden. Eine Folge hiervon ist, dass die existierenden Virtualisierungsarten nicht nur sehr unterschiedliche Funktionalität bieten, sondern auch sehr unterschiedliche Anforderungen an physische Ressourcen stellen. Daher kann es möglich sein, dass für einen speziellen Einsatzzweck ein bestimmter Virtualisierungsansatz geradezu optimal skaliert, während ein anderer versagt. Da der Grund für dieses Verhalten in der Architektur der Virtualisierungsansätze zu suchen ist, sind diese ein wichtiger zu untersuchender Faktor, wenn es darum geht, virtuelle Infrastrukturen zu optimieren. Hierbei ist es sowohl von Interesse, Flaschenhälse in der Implementierung bzw. der Architektur zu kennen als auch deren Vorteile.

Einfluss von Konfigurationsparametern

Nicht nur die Virtualisierungsansätze selbst sind ein interessanter zu untersuchender Faktor, sondern auch Faktoren, die auf die Virtualisierungsschicht Einfluss haben bzw. Gebrauch von ihr machen. Unter Einflussfaktoren zählen hierbei jede Art von Konfigurationsparametern, die eine Virtualisierungslösung bietet. Dies können Parameter wie zum Beispiel Schedulingalgorithmen, Anzahl und Art der bereitgestellten CPUs, die Anwesenheit von Hardware zur Unterstützung von Virtualisierung - allen voran Techniken wie AMD-V oder Intel VT-x - oder die Größe des virtuell bereitgestellten Arbeitsspeichers sein. Zur Nutzung der Virtualisierung ist die Fähigkeit von Betriebssystemen und Anwendungen zur Parallelisierung und die Anforderungen an die Hardware ein entscheidender Punkt, der wiederum eventuell durch die Hardware-Konfiguration der virtuellen Maschine entsprechend zum Vor- oder Nachteil werden kann. Die Einflussfaktoren auf diesem Gebiet sind, wie man bereits sieht, sehr vielfältig und in den meisten Fällen wohl auch nicht

unabhängig voneinander. Als Resultat dieser Teilfragestellung sollen am Ende Richtlinien zum Design effizienter virtueller Infrastrukturen entstehen.

Ressourcen und Lastverteilung in virtuellen Infrastrukturen

In virtuellen Infrastrukturen steht man zusätzlich vor dem Problem, dass eine systematisch fundierte Verteilung von virtuellen Maschinen auf physische Server ein bisher ungelöstes Problem ist. Zwar existieren Mechanismen, die für ein Loadbalancing zwischen den Servern sorgen, indem sie virtuelle Maschinen zwischen den Hosts bei Bedarf verschieben, jedoch basieren diese Techniken lediglich auf Erfahrungswerten, nicht aber auf allgemein gültigen Kriterien. Des Weiteren ist das Migrieren von virtuellen Maschinen im laufenden Betrieb eine sehr teure Operation, da sie sehr ressourcenlastig ist. Eventuell würde in konkreten Fällen schon eine lokale Anpassung der Hardwarezuordnung genügen, um einen Engpass zu eliminieren. Ein Ergebnis dieser Arbeit soll es daher sein, eine fundierte Basis für einen Ressourcen-Scheduler zu bieten, der aufgrund der in dieser Arbeit gewonnenen Erkenntnisse eine sinnvolle Entscheidungsgrundlage für seine Aufgabe hat.

Erhöhung des Wirkungsgrades virtueller Infrastrukturen

Letztendlich beschäftigen sich die in dieser Arbeit diskutierten Fragestellungen alle mit der Erhöhung der Effizienz von Virtualisierungstechniken. Dabei werden jedoch nicht die Virtualisierungstechniken an sich überarbeitet oder verbessert, sondern es wird lediglich versucht, vorhandene Rechenleistung möglichst effektiv zu nutzen, sprich den Wirkungsgrad der Virtualisierungsschicht so weit wie möglich zu erhöhen. Die Erkenntnisse dieser Arbeit können damit ohne weitere Investitionen dazu führen, dass die Leistungsfähigkeit der virtuellen Infrastruktur steigt. Die in dieser Dissertation untersuchten Dimensionen - Virtualisierungsansätze, Leistungsmessung, Analyse der Einflussfaktoren sowie Optimierungsansätze - werden auch in Abbildung 1.1 dargestellt.

1.3 Einordnung der Arbeit

Es existieren keine objektiven Leistungs-Untersuchungen von Virtualisierungsansätzen

Trotz der Tatsache, dass Virtualisierung sowohl für die Forschung als auch für Rechenzentren ein aktuelles Thema ist, existieren keine objektiven und öffentlich zur Verfügung stehenden Vergleiche zwischen den Virtualisierungsansätzen und Produkten. Zwar veröffentlicht jeder Hersteller von Virtualisierungslösungen zu Marketingzwecken Vergleiche mit Konkurrenzprodukten, jedoch mangelt es diesen Studien offensichtlich an Objektivität, da es exakt genau so viele Sieger zu geben scheint wie Teilnehmer. Dies gilt sowohl für Leistungs- als auch für Funktionalitätsvergleiche. Auch Unternehmen und Rechenzentren evaluieren natürlicherweise Virtualisierungslösungen für den praktischen Einsatz. Diese Erfahrungen dürften erheblich neutraler ausfallen als die der entsprechenden Hersteller, meist haben Unternehmen aber nicht die Zeit oder das Interesse, wirklich alle in Frage kommenden Lösungen auch ernsthaft zu evaluieren, und sind außerdem auf ihre Sichtweise der Dinge eingeschränkt. Natürlich existieren auch Studien und Messungen aus objektiven Quellen, wie z.B. [Stra 06], in der Performanzeinflüsse des

Intel VT-x Befehlssatz evaluiert werden. Leider beschränken sich die Analysen meist auf die Untersuchung eines oder einer kleinen Anzahl an Parametern. Es wäre schön, wenn man diese Einzelmessungen wie ein Puzzle zu einem großen Bild zusammensetzen könnte, doch leider sind die Messverfahren und die umgebenden Einflussfaktoren nicht standardisiert, so dass die einzelnen Puzzleteile alle zu unterschiedlichen Puzzles gehören. Nichtsdestotrotz sind diese Arbeiten als Grundlage dieser Arbeit enorm hilfreich, denn sie geben bereits erste Hinweise, welche Parameter bzw. welche Parameterkombinationen eine detailliertere Betrachtung wert sind. Zumindest den Leistungsaspekt betreffend soll diese Arbeit eine gewisse Vergleichbarkeit der Virtualisierungsansätze und Produkte zum Ergebnis haben, welche bis heute nicht gegeben ist.

Bestehende Scheduler haben konzeptionelle Defizite

Ein weiterer zum großen Teil unerforschter Aspekt ist die Menge der Faktoren, die die Virtualisierung von Systemen in ihrer Effizienz beeinflussen. Dies ist jedoch eine erforderliche Grundlage zum effizienten Design und Betrieb von virtuellen Infrastrukturen. Darüber hinaus unterstützen bislang alle größeren Hersteller von Virtualisierungsprodukten die Migration von Maschinen im laufenden Betrieb auf andere Hosts und dynamische Ressourcenzuordnung innerhalb eines Hosts, um einen Lastausgleich zu ermöglichen. Nach welchen Kriterien diese Zuordnung von Ressourcen zu virtuellen Maschinen erfolgt, hat bisher kein Anbieter veröffentlicht. Es ist nicht auszuschließen, dass diese Verfahren lediglich auf empirisch gefundenen Schwellwerten basieren. Dieser Verdacht erhärtet sich, wenn man einen solchen Ressourcen-Scheduler im laufenden Betrieb beobachtet. Der Scheduler ist einen großen Teil der Zeit damit beschäftigt, Maschinen von einem Host auf einen andern zu verschieben, was insbesondere bei stark ausgelasteten virtuellen Maschinen eine erhebliche zusätzliche Last impliziert. Kurzfristig arbeitet der Scheduler im Falle eines Ressourcenengpasses also sogar kontraproduktiv. Des Weiteren kann beobachtet werden, dass eine Maschine zwischen zwei Hosts pendelt, wenn diese in Summe in etwa die identische Last tragen und die Kapazität der Hosts an ihre Grenzen stößt. In diesem Bereich ist es offensichtlich nötig, zunächst Handlungsoptionen zu identifizieren und deren Auswirkung und Realisierungskosten zu betrachten. Hierzu gehört es unter anderem auch, ein Oszillieren von virtuellen Maschinen zwischen zwei oder mehreren Hosts mit geeigneten Mitteln zu verhindern. Auf dieser Basis kann anschließend ein fundiertes Konzept für einen Ressourcen-Scheduler erstellt werden, der in den erwähnten Situationen angemessen reagiert.

1.4 Vorgehensmodell

Leistungsbegriffe und Virtualisierungsansätze als Grundlage

Grundlage für die Arbeit in dieser Dissertation ist die Definition eines klaren Leistungsbegriffes für virtuelle Infrastrukturen. Darauf aufbauend können anschließend weitere Aspekte in dieser Richtung untersucht werden, allen voran die Messbarkeit des Leistungsverhaltens in zunächst klassischen Umgebungen. Diese Erkenntnisse können anschließend auf virtuelle Umgebungen transformiert

werden. Gegenstand der Untersuchung in diesem Bereich sind zum Beispiel, an welchen Schnittstellen Leistung gemessen werden kann und welche Werkzeuge, Protokolle oder Daten-Schemata sich für diese Messungen eignen. Voraussetzung für die Entwicklung eines Leistungsbegriffes im Virtuellen ist es jedoch, dass die verschiedenen Ansätze der Virtualisierung, insbesondere die Host-Virtualisierung, untersucht werden. Hier ist unter anderem interessant, auf welcher Ebene die Virtualisierung vorgenommen wird und welche Unterschiede hinsichtlich Funktionalität, aber vor allem auch im Punkte Leistung sich hierdurch zwischen den einzelnen Virtualisierungsarten ergeben. Primär interessiert in diesem Stadium jedoch lediglich die Architektur der verschiedenen Ansätze. Das Kapitel 2 befasst sich daher mit Virtualisierungskonzepten an sich, während der Anfang des Kapitels 3 sich mit den Grundlagen der Leistungsmessung auseinandersetzt. Der virtuelle Leistungsbegriff wird voraussichtlich eine Obermenge des klassischen Leistungsbegriffes bilden, da alle Metriken und Messverfahren, die in physischen Systemen eingesetzt werden können, auch in virtuellen Maschinen durchführbar sind. Durch die Virtualisierung kommen lediglich weitere Schichten in der Architektur hinzu, so dass z.B. zusätzliche Metriken und Messverfahren denkbar erscheinen.

Theoretische Leistungsanalyse von Host-Virtualisierungsansätzen

Mit den Grundlagen aus den vorangehenden Kapiteln ist es möglich, erste Abschätzungen über das Leistungsverhalten von Host-Virtualisierungslösungen zu tätigen und ansatzweise potentielle Flaschenhälse zu identifizieren. Auf Grund dieser Analyse werden jedoch nur qualitative Aussagen über das Leistungsverhalten möglich sein, da das Gesamtverhalten von sehr vielen Faktoren abhängig ist und diese theoretisch nicht vollständig analysierbar sind. Nichtsdestotrotz können diese theoretischen Erkenntnisse als Filter dienen, um nicht eine schier unendlich scheinende Anzahl an Permutationen über alle denkbaren Einflussfaktoren empirisch messen zu müssen. Eine umfangreiche, aber systematische Messreihe erscheint als einzige Möglichkeit, Stärken und Defizite im Kontext des Leistungsverhaltens quantitativ zu analysieren, weshalb sich die Abschnitte 3.2 bis 4.4 mit diesem Thema beschäftigen. Um den Wirkungsgrad verbessern zu können, ist zunächst eine Analyse notwendig, die potentielle Einflussfaktoren auf das Leistungsverhalten identifiziert. In einem weiteren Schritt können anschließend Einzel- sowie Kreuzmessungen der entsprechenden Parameter vorgenommen werden. Voraussichtlich werden auch vergleichbare Messungen an physischen Systemen aufschlussreiche Erkenntnisse bieten. Die Auswertung und Korrelation der diversen Messungen schaffen die Basis für die Lösung vieler ungelöster Fragestellungen, wie sie schon im Abschnitt 1.1 beschrieben sind.

Idealzustand und Optimierungsansätze

Auf derselben Grundlage ist es möglich, verschiedene Ansätze zu entwickeln, um Virtualisierung noch effizienter betreiben zu können. Grob lassen sich diese Ansätze in die beiden Punkte „Verbesserung der Virtualisierungstechniken" und „Effizienzerhöhung" einteilen. Für den ersten Punkt existieren bereits eine Reihe an Lösungen, die von den Herstellern von Virtualisierungslösungen und diversen freien Communities ständig weiterentwickelt und verbessert werden, während der zweite Punkt weitestgehend noch nicht untersucht ist. Der Fokus dieser Arbeit liegt daher klar auf zweiterem Ansatz und wird im Kapitel 5 behandelt.

Möglichkeiten zur Erhöhung des Wirkungsgrades

Exemplarisch wird sich diese Arbeit mit zwei Ansätzen zur Steigerung des Wirkungsgrades auseinandersetzen. Die statische Optimierung erörtert hierbei die Fragestellungen, die sich zum Zeitpunkt der Migration hin zu einer virtuellen Infrastruktur stellen und auf Leistungsaspekten beruhen. Beispielhaft sei hier die Anzahl der in etwa benötigten physischen Server genannt, um die alte Infrastruktur performant virtualisieren zu können. Die dynamische Optimierung untersucht, welche der untersuchten Einflussparameter zur Laufzeit geändert werden können und bei welchen Symptomen welche Änderungen zum Leistungserhalt bei kurzfristig höherer Last sinnvoll eingesetzt werden können. Dies entspricht dem Entwurf des Regelwerkes für einen Ressourcen-Scheduler. Beide Vorgehensmöglichkeiten werden in Kapitel 6 behandelt. Das gesamte Vorgehensmodell dieser Arbeit wird in Abbildung 1.2 modelliert.

Abbildung 1.1: In dieser Arbeit behandelte Fragestellungen

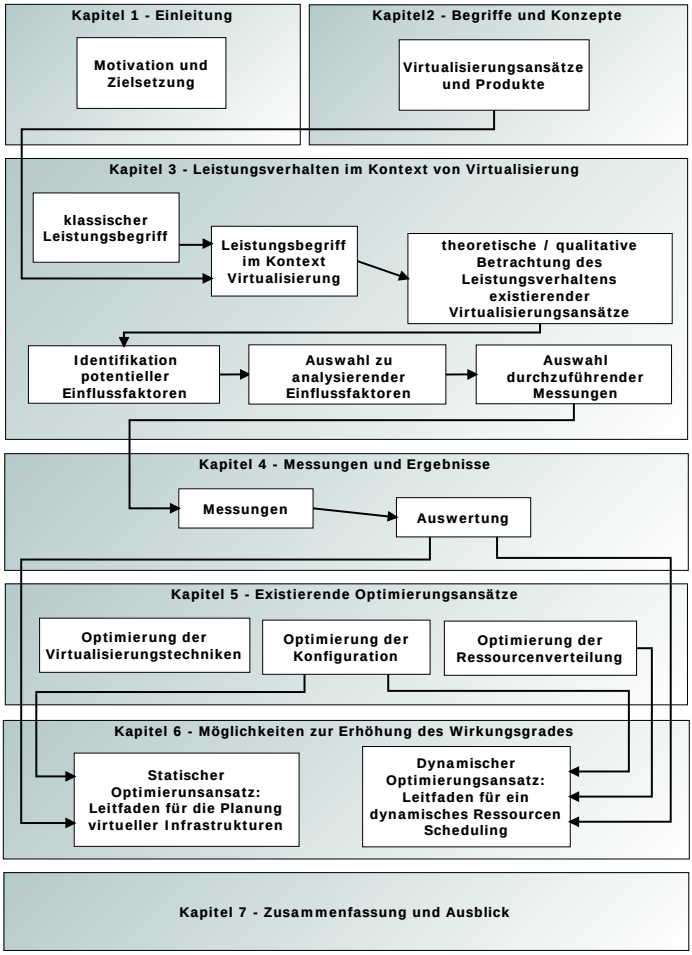


Abbildung 1.2: Vorgehensmodell innerhalb dieser Dissertation

2 Konzepte zur Virtualisierung

Inhalt

Geschichtliche Entwicklung der Host-Virtualisierung

Auch wenn Virtualisierung zum jetzigen Zeitpunkt ein sehr aktuelles Thema für Hersteller von Soft- und Hardwareprodukten, Unternehmen und Rechenzentren darstellt, so ist die zugrundeliegende Idee alles andere als neu: Physische Ressourcen werden in logische Instanzen aufgeteilt und den Anwendungen als funktionsfähige Ressourcen isoliert von einander zur Verfügung gestellt. Bereits im Jahre 1959 wurde ein erster Ansatz in diese Richtung unternommen. Christopher Strachey führte mit seiner Abhandlung ,,Time Sharing in Large Fast Computers" [Stra 59] das Multiprogramming ein. Zum ersten Mal war es damit möglich mehrere Prozesse isoliert von einander zeitlich verzahnt auszuführen. Dieser erste Virtualisierungsansatz benötigt zur gleichzeitigen Ausführung mehrerer Programme zusätzliche Ressourcen, um die Kontextinformationen der Prozesse zu verwalten und Prozesswechsel durchzuführen. Durch die zeitlich effizientere Auslastung der Einzelkomponenten der Maschinen sank die durchschnittliche Antwortzeit der Prozesse dagegen stark. Mitte der 60er Jahre wurde dieses Konzept ergänzt

durch das sogenannte Paging, welches eine Abbildung von virtuellem Hauptspeicher auf physische Speichermedien durch Adressübersetzung erlaubte. Mit dieser Technik stand einem Rechner plötzlich mehr virtueller Hauptspeicher zur Verfügung als er physisch besaß. Die Differenz von virtuellem und physischem Hauptspeicher wurde auf günstigerem permanenten Speicher gehalten. 1972 brachte IBM den ersten Mainframe mit integrierten Virtualisierungsfunktionen auf den Markt. Das System mit dem Namen VM/370 konnte mehrere Betriebssysteminstanzen zur selben Zeit ausführen, indem es vorhandene Ressourcen hardwareseitig in Partitionen aufteilte, die jeweils einer Instanz exklusiv zur Verfügung standen, den Rest übernahm ein sogenanntes „Control Program", das wir heute Virtual Machine Monitor oder auch Hypervisor nennen würden. Weitere Evolutionsstufen dieses Systems waren die Modelle S/370 und ESA/390, welche die Vorfahren der heute als 64 Bit Variante vermarkteten Z-Series darstellten.

Probleme bei der Virtualisierbarkeit der x86-Architektur

Leider lassen sich nicht alle beliebigen Architekturen auf diese einfache Art und Weise virtualisieren. Zu den problematischen Architekturen gehört unter anderem die x86-Architektur. Eine CPU beziehungsweise Instruction Set Architecture (ISA) ist nach G.J. Popek und R.P. Goldberg nämlich genau dann virtualisierbar, wenn alle privilegierten Instruktionen eine Exception auslösen, sobald sie in einem nicht-privilegierten Kontext ausgeführt werden. Alle sensiblen Operationen, das heißt Operationen, deren Wirkung vom Kontext des Prozessors abhängt, müssen privilegiert sein. Wörtlich fordern Popek und Goldberg:

„Für jeden üblichen Rechner der dritten Generation kann genau dann ein VMM gebaut werden, wenn für diesen Rechner die Menge der sensitiven Instruktionen eine Untermenge der privilegierten Instruktionen ist." [PoGo 73]

Genau dieser Punkt ist jedoch bei der x86-Architektur nicht erfüllt. Es existiert ungefähr ein Duzend nicht-privilegierter, aber sensitiver Befehle, weshalb die x86-Architektur nach G.J. Popek und R.P. Goldberg nicht virtualisierbar ist. Dass es dennoch möglich ist, zeigte VMware 1999 mit der Veröffentlichung der ersten Version der VMware Workstation, die eine Technik namens „Scan Before Execution" (SBE) verwendete. Dabei wird Code, den eine virtuelle Maschine ausführen will, vor der Ausführung auf sensible Befehle hin gescannt und diese durch eine Reihe unproblematischer Operationen ersetzt. Dieses Vorgehen ist bei neueren Prozessoren der x86-Architektur nicht mehr nötig, da sie mit den Technologien Intel VT-x und AMD-V mittlerweile der Goldberg-Bedingung genügen.

Anwendungsbereich von Virtualisierung weitet sich aus

Mit der Entwicklung neuer Virtualisierungstechnologien hat sich auch die Bedeutung des Begriffes Virtualisierung geändert. Eine gängige Definition für Host-Virtualisierung, die sich bis heute gehalten hat, lautet etwa: „Virtualisierung bezeichnet Methoden, die es erlauben, Ressourcen eines Computers aufzuteilen" [Wiki 08, Stichwort: Virtualisierung]. Es existieren jedoch mittlerweile Anwendungsbereiche, bei denen physische Ressourcen nicht aufgeteilt, sondern vereinigt werden sollen. Ein Beispiel hierfür ist zum Beispiel das anwendungstransparente Umschalten zwischen redundanten FibreChannel Links mit dem Ziel Loadbalancing und Failover Mechanismen zu realisieren. Logisch existiert dabei im

Betriebssystem genau ein FibreChannel Adapter, der je nach Anforderung aber an unterschiedliche physische Adapter gebunden werden kann. Techniken wie diese werden zur Zeit unter anderem von Fujitsu-Siemens unter dem Namen I/O-Virtualisierung entwickelt. Eine bessere und vom Kontext Host-Virtualisierung unabhängige Definition des Begriffes Virtualisierung könnte daher folgendermaßen formuliert werden:

Definition des Begriffes Virtualisierung

Definition. Virtualisierung ist die unter Umständen rekursiv durchführbare Abbildung von m Schnittstellen der Schicht s auf n funktional gleichartige Schnittstellen der Schicht s-1 unter der Zuhilfenahme weiterer Ressourcen mit dem Ziel der Abstraktion.

Diese zugegeben abstrakte Definition lässt sich auf alle Teilbereiche der Virtualisierung, angefangen vom Multitasking, über Paging bis zur Techniken der Host-Virtualisierung, Multiplexingverfahren und I/O-Virtualisierung anwenden. Da sich diese Arbeit im Wesentlichen mit kompletten Systemen beschäftigt, werden im Folgenden die einzelnen Techniken der Host-Virtualisierung detaillierter vorgestellt und verglichen. Zuvor wird jedoch die Funktionsweise eines physischen Systems skizziert, um die bei der Host-Virtualisierung zu lösenden Probleme zu verstehen. Gegen dieses System kann im Folgenden eine Overhead-Betrachtung erfolgen.

2.1 Architektur eines klassischen physischen Systems

Typischer Aufbau eines Systems in der x86-Architektur

In diesem Kapitel werden zunächst einige Funktionen der PC-Architektur tech-

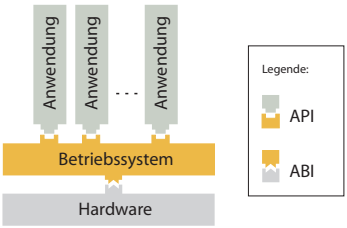


Abbildung 2.1: Architektur eines physischen Systems

nisch detailliert erläutert. Die folgenden Abschnitte zeigen anschließend, welche Probleme bei der Virtualisierung dieser Funktionen entstehen und wie die existierenden Virtualisierungsansätze versuchen, diese zu lösen. Diese Gegenüberstellung zeigt nicht nur die unterschiedlichen Herangehensweisen der verschiede-

nen Virtualisierungsansätze, sondern kann auch ein abweichendes Verhalten hinsichtlich ihrer Performanz erklären. Heute übliche physische Systeme bestehen normalerweise aus einer Hardwareplattform, auf der genau ein Betriebssystem in Form von Software aufsetzt. Sofern es sich bei einem System nicht um einen Großrechner handelt, hat sich eine Plattform auf dem Markt durchgesetzt, die sowohl in Rechenzentren als auch an Einzelarbeitsplätzen eingesetzt wird: die PC-Architektur, besser bekannt unter dem Namen x86-Architektur. Diese Bezeichnung ist streng genommen zwar nicht korrekt, da x86 einen Prozessorbefehlssatz bezeichnet und keine Systemarchitektur. Da dieser Begriff im Kontext von Virtualisierung jedoch ständig zum Einsatz kommt, wird er auch in dieser Arbeit als Synonym für den eigentlich richtigen Begriff PC-Architektur verwendet.

Einschränkung auf die x86-Architektur

Auch wenn viele andere Plattformen existieren, wie etwa im Embedded Bereich, orientiert sich diese Arbeit an der x86-Architektur, da diese neben ihrer starken Verbreitung auch Plattform für die zur Zeit entwickelten Host-Virtualisierungslösungen ist und gegenüber anderen Architekturen eine Vorreiterrolle auf diesem Gebiet einnimmt. Früher oder später werden allerdings andere Architekturen, zum Beispiel aus dem Embedded Bereich, in der Entwicklung nachziehen und vor ähnlichen Herausforderungen stehen wie jetzt die x86-Architektur.

ABI der x86-Architektur als Schlüssel zur Virtualisierung

Die Kommunikation der verschiedenen Schichten eines Systems erfolgt über fest definierte Schnittstellen, die es ermöglichen, dass auf einer Architektur diverse Betriebssysteme lauffähig sind. Diese Schnittstelle zwischen Betriebssystem und Architektur nennt man üblicherweise Application Binary Interface (ABI). Für die x86-Architektur existieren im Wesentlichen diverse Windows und Unix Varianten als Betriebssystem, die selbst wiederum standardisierte Schnittstellen für ihre Applikationen bieten. Diese Schnittstellen heißen Application Programming Interface, oder kurz API. Für die Host-Virtualisierung ist vor allem die ABI der x86-Architektur von Bedeutung, da hier eingegriffen werden muss, um die x86-Architektur virtualisieren zu können. Insbesondere die Adressübersetzung und die Ausführung privilegierter CPU-Befehle sind hier interessant, weshalb sie im Folgenden kurz erläutert werden. Abbildung 2.1 skizziert den Aufbau eins x86-basierten Systems.

2.1.1 Adressübersetzung in der x86-Architektur

Protected Mode: Segmentierung und Paging

Moderne Betriebssysteme der x86-Architektur verwenden zum Speicherzugriff in der Regel den Protected Mode der x86-Architektur. Es existieren zwar grundsätzlich auch andere Modi, diese existieren aber nur noch aus Kompatibilitätsgründen zu älteren Betriebssystemen. Der Protected Mode ist ein Speicherzugriffsverfahren, das Segmentierung und Paging in sich vereint und hardwareseitig Unterstützung für gewisse Teiloperationen bereitstellt. Abbildung 2.2 skizziert den generellen Ablauf der einzelnen Übersetzungsschritte.

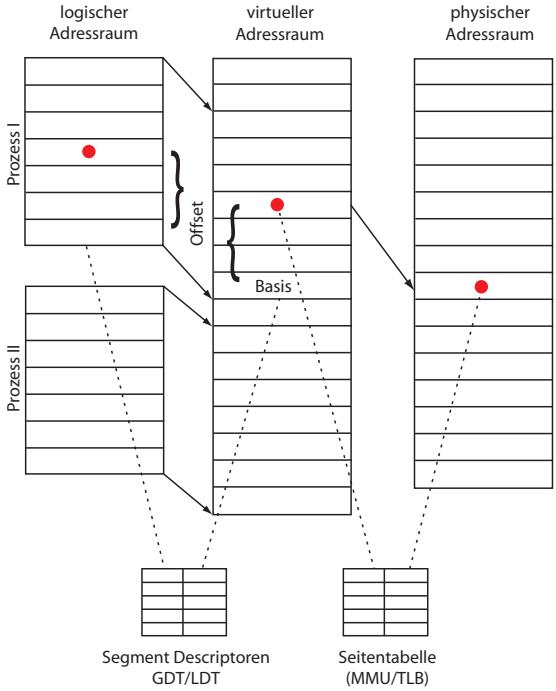


Abbildung 2.2: Adressübersetzung in einem x86-System

Realisierung der Adressübersetzung

Zunächst wird jedem Prozess ein eigener logisch linearer Adressraum zur Verfügung gestellt. Dieser wird in einen linearen virtuellen Adressraum abgebildet, der die Adressräume aller Prozesse des Betriebssystems beinhaltet. Um den Speicher eines Prozesses in diesem virtuellen Adressraum adressieren zu können, benötigt man die Anfangsadresse (Basis / Segment) des linearen Speicherblocks im virtuellen Adressraum. Anschließend kann man die Speicheradresse mit demselben Offset adressieren wie im logischen Adressraum. Dieser Vorgang heißt Segmentierung und ermöglicht eine einfachere Art der Programmierung, da der Programmierer von einem linearen Speicheraufbau ausgehen kann, obwohl dieser physisch nicht vorhanden ist. Anschließend erfolgt eine Abbildung des virtuellen Adressraums in den realen Speicher. Dieser Vorgang wird durch Paging realisiert. Dazu wird der virtuelle Speicher in viele gleichgroße, meist 4 KB fassende Seiten unterteilt und diese auf freie Speicherblöcke (Frames) im Hauptspeicher oder auf

Page Tables und Directory Tables

den Sekundärspeicher abgebildet. Die notwendigen Übersetzungsinformationen werden in Tabellen abgelegt, so dass eine Seite im Hauptspeicher jederzeit exakt einer virtuellen Adresse zugeordnet werden kann. Aus Effizienzgründen wird die Übersetzung nicht in einer einzigen, sondern in zwei hierarchisch aufeinanderfolgenden Tabellen realisiert. Diese Tabellen heißen Page Directory und Page Table, werden aber der Einfachheit halber nur als Seitentabelle bezeichnet. Jedem Prozess ist ein eigenes Page Directory zugeordnet, das selbst Referenzen auf bis zu 1024 Page Tables mit je 1024 Einträgen enthalten kann. Aus diesem Grund sind 4 GB die maximal adressierbare Speichergröße für einen Prozess. Abbildung 2.3 stellt den Aufbau der Tabellenstruktur schematisch da. Um das Paging möglichst effizient zu realisieren, wurden die eigentlichen Übersetzungsschritte in Hardware realisiert. Die hierfür zuständige Komponente heißt MMU und besitzt einen eigenen Cache, um häufig benutzte Übersetzungen von Adressen zwischenzuspeichern. Dieser Cache heißt Translation Lookaside Buffer (TLB). Dieser muss bei jedem Prozesswechsel geleert werden, da ein Wechsel des Prozesses auch einen Wechsel des Page Directories zur Folge hat und der Cache damit für den aktuellen Prozess ungültige Adressen enthalten würde. Das Löschen des Caches geschieht automatisch, wenn der Inhalt des CR3 Registers, das auf das aktuelle Page Directory referenziert, neu geschrieben wird.

Implementierung in Hardware

2.1.2 Das Ring Konzept der x86-Architektur

Kernel und User Mode und ihre Realisierung in Ringstrukturen

Wie bereits im letzten Abschnitt erwähnt, arbeiten moderne Betriebssysteme im Protected Mode, nur während des Bootvorganges arbeiten sie teilweise im Real Mode, da in diesem Modus der Zugriff auf alle Ressourcen des Systems möglich ist. Im Protected Mode hingegen existieren vier Sicherheitsstufen, genannt Ring 0 bis Ring 3. Mit steigender Nummer des Rings nehmen die Privilegien ab. Im Ring 0 ist alles erlaubt, hier laufen die Kernprozesse des Betriebssystems, daher heißt dieser Ring auch Kernel Mode. Ring 1 und 2 sind zwar vorhanden, werden bei der x86-Architektur aber nur von sehr wenigen exotischen Betriebssystemen zur Ausführung von Treibern verwendet. Windows und Linux Systeme machen keinen Gebrauch von diesen Ringen. Ring 4 stellt schließlich den Kontext für Anwendungen, den sogenannten User Mode, bereit. Siehe hierzu auch Abbildung 2.4. Innerhalb des Prozessors existieren Befehle, die nur im Ring 0, also im Kernel Mode ausgeführt werden können. Diese Befehle nennt man auch privilegiert. Das Ausführen von privilegierten Befehlen außerhalb von Ring 0 wird vom Prozessor geblockt und führt zu einer Exception. Will eine Anwendung eine privilegierte Operation ausführen, so muss es einen Systemaufruf initiieren. Hierzu existieren zwei Möglichkeiten:

Ausführung privilegierter Befehle durch Anwendungen

- Die Nummer des Systemaufrufes wird in des Register EAX geschrieben und anschließend der Befehl „int” oder binär 0x80 aufgerufen, was einen Interrupt auslöst, der vom Betriebssystem behandelt werden kann.
- Die Nummer des Systemaufrufes wird in das Register EAX geschrieben und

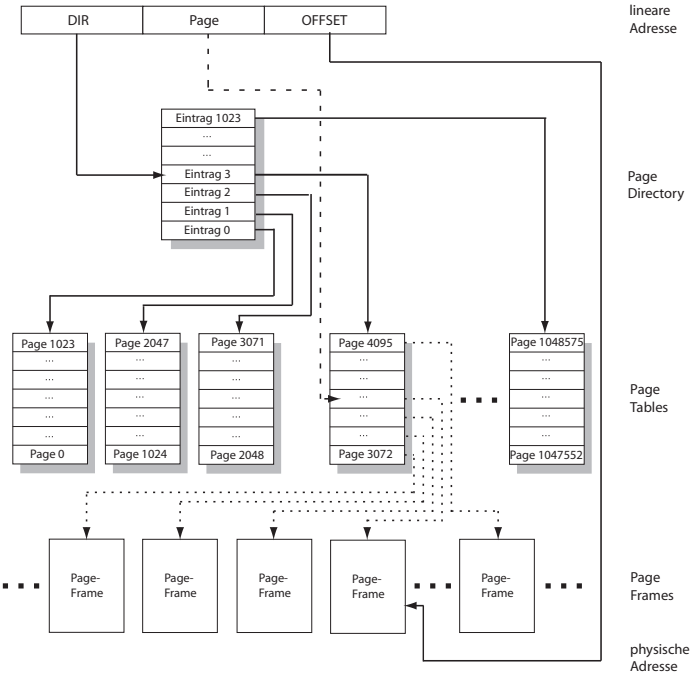


Abbildung 2.3: Pagingverfahren in einem x86-System [AW 08]

anschließend je nach Prozessortyp SYSCALL oder SYSENTER aufgerufen.

Die beiden letzteren Befehle sind speziell für Systemaufrufe entwickelt worden und sind effizienter als der generische „int" Befehl, sind aber erst seit dem Intel Pentium II / AMD K6 verfügbar. [Rudo 08], [Wiki 08, Stichwort: Ring (computer security), System call]

2.2 Herausforderungen bei der Virtualisierung von Maschinen

Herausforderungen bei der Virtualisierung der x86-Architektur

Bei der Host-Virtualisierung von x86-Systemen treten durch die verletzte Popek-/Goldberg-Bedingung einige Probleme auf, die alle Virtualisierungsansätze konzeptionell zu lösen haben. Eine Ausnahme bildet hier die Native Virtualisierung,

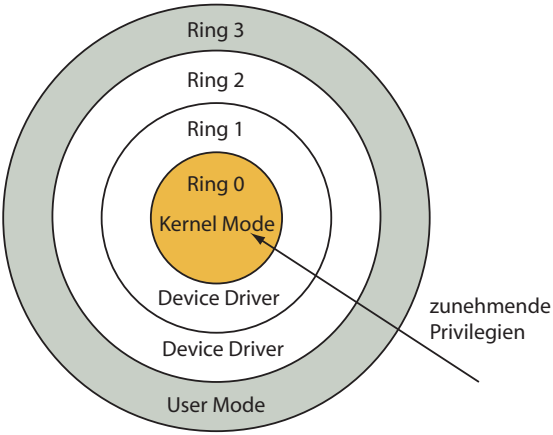


Abbildung 2.4: Das Ring Konzept der x86-Architektur

da sie von CPU-Befehlssätzen Gebrauch macht, die die x86-Architektur nach Popek/Goldberg virtualisierbar machen. Neben dem eigentlichen Scheduling zwischen den virtuellen Betriebssysteminstanzen muss nach [BDF+ 03] jeder Host-Virtualisierungsansatz Konzepte für die folgenden Problemstellungen bereitstellen.

Hauptspeicherzuordnung

1. Hauptspeicherzugriff

   a) Adressübersetzung
   Der Speicherbereich einer virtuellen Maschine muss auf realen Speicher abgebildet werden. Hierfür müssen zusätzlich zu Segmentierung und Pagingverfahren Übersetzungstabellen bereit stehen. Eine besondere Herausforderung in diesem Zusammenhang ist, dass hardwareseitig implementierte Caches wie der Translation Lookaside Buffer (TLB), der die Übersetzungsergebnisse der Memory Management Unit (MMU) zwischenspeichert bei jedem Kontextwechsel zwischen verschiedenen virtuellen Maschinen gelöscht wird.

   b) Isolation
   Der Speicherbereich des Virtual Machine Monitor (VMM) muss vor Zugriffen von virtuellen Maschinen geschützt werden, um Manipulation zu vermeiden. Des Weiteren muss durch den VMM eine Überprüfung stattfinden, dass eine virtuelle Maschine auch tatsächlich nur ihren eigenen Speicherbereich liest und schreibt.

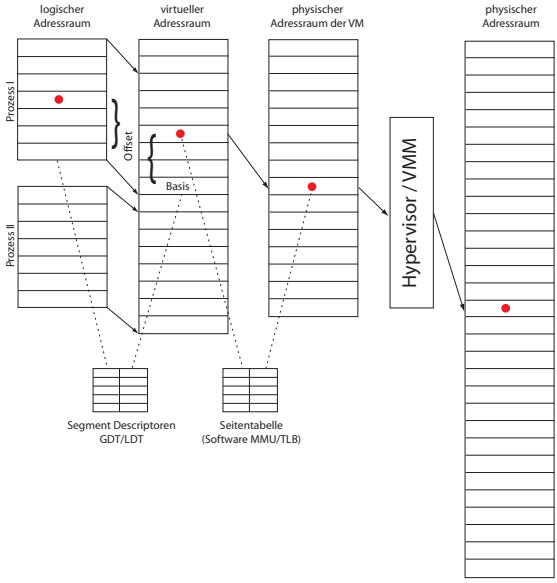


Abbildung 2.5: generische Adressübersetzung in einem virtualisierten System

2. CPU

Logische Bindung von CPUs am virtuelle Maschinen

a) Zugriffschutz
Alle virtuellen Maschinen müssen mit weniger Privilegien ausgeführt werden als der VMM / Hypervisor selbst, um ein Aushebeln der Zugriffskontrollen, die durch den VMM implementiert werden, zu unterbinden.

b) Exceptions
Kritische Exceptions, die während der Ausführung von Code innerhalb von virtuellen Maschinen auftreten, müssen vom VMM abgefangen und behandelt werden.

c) Systemaufrufe
Systemaufrufe einer Anwendung in der virtuellen Maschine müssen vom Betriebssystem der zugehörigen virtuellen Maschine bearbeitet werden. Ohne zusätzliche Mechanismen wird dieser Systemaufruf vom hierarchisch untersten Betriebssystem / Hypervisor d.h. dem privilegiertesten behandelt. Dies würde zu nicht ausführbaren Sys-

temaufrufen im VM-Kontext führen. Daher muss ein Mechanismus gefunden werden, so dass Systemaufrufe in virtuellen Maschinen auch von diesen behandelt werden können.

d) Interrupts
Hardware Interrupts müssen vom VMM an das betroffene virtuelle Gastbetriebssystem weitergeleitet werden.

e) Zeit
Die Unterscheidung zwischen realer Zeit und virtueller Zeit, die einer virtuellen Maschine tatsächlich zur Befehlsausführung bleiben, muss möglich sein, um ein korrektes Accounting zu ermöglichen.

Zuordnung von peripheren Komponenten

3. Zugriff auf Geräte

   a) Geräte wie Festplatten und Netzwerkkarten müssen den virtuellen Maschinen während ihrer Laufzeit entweder physisch exklusiv, mindestens aber logisch exklusiv zur Verfügung stehen. Logische Geräte werden meist durch Emulation vom VMM bereitgestellt.

Einige dieser Punkte sind verhältnismäßig einfach realisierbar. Am schwierigsten sind die Probleme der Hauptspeicherzuordnung und der Systemaufrufe, da sie essentielle Funktionen von Betriebssystemen betreffen und ihr Konzept zur Lösung des Problems starke Auswirkungen auf das Leistungsverhalten der jeweiligen Implementierung mit sich bringt. Im Folgenden werden die bekanntesten Konzepte und Lösungsansätze für die eben erwähnten Probleme der Host-Virtualisierung vorgestellt und ihre Unterschiede herausgestellt.

2.3 Emulation

Hohe Funktionalität bei geringer Performanz

Bei Emulation handelt es sich, was die Architektur betrifft, um die einfachste und flexibelste Art der Virtualisierung, unbestritten aber auch um die ressourcenhungrigste und langsamste Variante. Es wird versucht, jede Einzelkomponente einer Rechnerarchitektur in Software funktional nachzubilden, sprich zu emulieren. In Summe entsteht dabei eine virtuelle Plattform, auf der wiederum existierende Betriebssysteme lauffähig sind. Dabei spielt es keine Rolle, welche Architektur einem Emulator zugrunde liegt und welche emuliert werden soll. Das Konzept sieht vor, dass für jeden Befehl der emulierten Architektur eine funktional identische Übersetzung in der darunterliegenden Architektur existiert, die stattdessen ausgeführt werden kann. Da aus diesem Grund jeder Befehl während seiner Ausführung übersetzt werden muss, stellt es für den Emulator kein Problem dar, Adressübersetzung, Übersetzung von Systemaufrufen sowie die anderen oben genannten Anforderungen an einen Virtual Machine Monitor zu realisieren. Dieses Vorgehen birgt in sich den großen Vorteil, unabhängig von der zugrunde liegenden Architektur beliebige andere Architekturen zu emulieren. Ein Emulator kann durchaus in der Lage sein, auch mehrere unterschiedliche Architekturen zur selben Zeit zu emulieren und

Übersetzung zwischen divergenten Architekturen

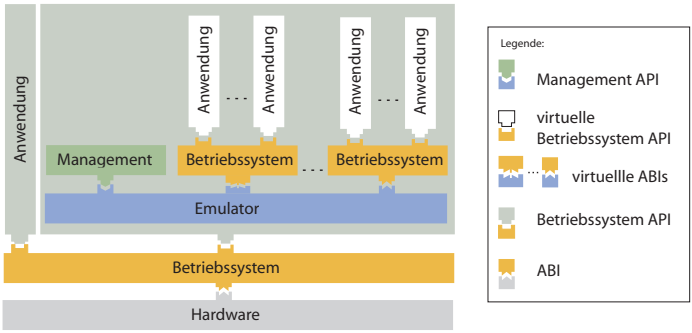


Abbildung 2.6: Emulation

auf die zugrundeliegende Architektur abzubilden. Dies ist auch in Abbildung 2.6 ersichtlich. Leider nimmt die Übersetzung der Befehle sehr viel Zeit in Anspruch, so dass sich dieses Verfahren nicht für den Produktivbetrieb einer großen Anzahl an Rechnern eignet, sondern lediglich für Entwicklungs- und Testzwecke interessant ist. Ganz nutzlos ist die Technik der Emulation trotzdem nicht, da viele Implementierungen von Hypervisorn, die sich primär einer anderen Technik der Host-Virtualisierung bedienen, einzelne Bestandteile der Systemarchitektur gezwungenermaßen emulieren müssen. Auf eine tiefergehende Leistungsanalyse der einzelnen Techniken wird in diesem Kapitel zunächst verzichtet. Diese erfolgt im Kapitel 3, nachdem ein entsprechender Leistungsbegriff für virtuelle Systeme definiert worden ist.

2.4 Paravirtualisierung

Portierung des Betriebssystems an die modifizierte ABI

Das Konzept der Paravirtualisierung ist die jüngste aller hier vorgestellten Techniken. Paravirtualisierung unterscheidet sich von den bereits genannten Konzepten dadurch, dass dem Betriebssystem der virtuellen Maschine nicht eine der physischen Plattform identische Schnittstelle zur Verfügung gestellt wird, sondern eine leicht veränderte Schnittstelle instanziiert wird. Bereit gestellt wird diese Schnittstelle vom Virtual Machine Monitor, der je nach Schicht, auf der die Virtualisierung durchgeführt wird, auch als Type 1 / Native /Bare Metal-Hypervisor (siehe auch Bild 2.7) oder auch Type 2 / Hosted-Hypervisor (siehe auch Bild 2.8) bezeichnet wird. Wie der Name bereits vermuten lässt, setzen Type 1 Hypervisor direkt auf Hardware auf, während Type 2 Hypervisor als Anwendung innerhalb eines Betriebssystems implementiert werden. Als Folge der Änderungen an der Schnittstelle zum Gastbetriebssystem muss dieses geringfügig an die neue

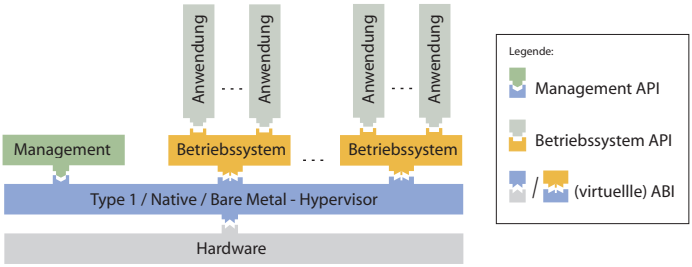


Abbildung 2.7: Architektur eines paravirtualisierten Systems - Type 1

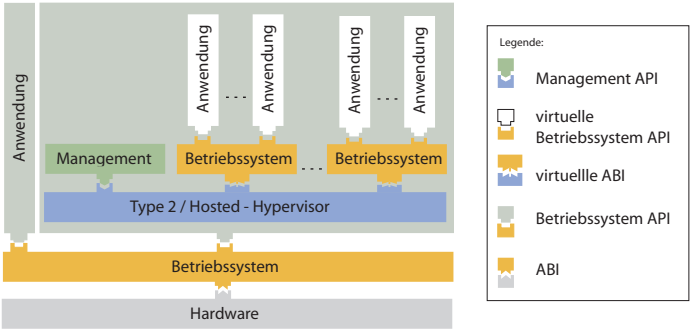


Abbildung 2.8: Architektur eines paravirtualisierten Systems - Type 2

Architektur angepasst werden. Diese Änderungen können von den Entwicklergemeinden nur an quelloffenen Betriebssystemen vorgenommen oder direkt vom Hersteller des Betriebssystems implementiert werden, so dass nicht alle Betriebssysteme auf paravirtualisierten Infrastrukturen lauffähig sind. Dieser Nachteil in der Kompatibilität zu Betriebssystemen wird jedoch durch eine ansonsten sehr effiziente Architektur aufgehoben. Da das Betriebssystem von der Virtualisierung weiß, können kritische Prozessorbefehle vermieden werden, indem das Betriebssystem bereits nativ nur nicht-privilegierte Prozessorbefehle verwendet und privilegierte Befehle direkt an eine zusätzliche Schnittstelle am Hypervisor absetzt, der sie an Stelle des Betriebssystems entsprechend ausführt. Systemaufrufe können behandelt werden, indem das Gastbetriebssystem einen Exceptionhandler beim Hypervisor registriert. Im Falle eines Systemaufrufes aus dem Gastsystem heraus wirft der Prozessor aufgrund zu geringer Privilegien eine Exception, die anschließend vom Hypervisor behandelt werden kann, indem er die Kontrolle

Vermeidung von Overhead bei Systemaufrufen

an den vom Gastbetriebssystem registrierten Exceptionhandler übergibt. Eine aufwendige Modifikation des Binärcodes von Gastbetriebssystem und dessen Anwendungen zur Laufzeit wie bei der Vollvirtualisierung entfällt damit. Um die Privilegien des Gastbetriebssystems herabzustufen, ist eine Neuzuordnung der Ringe in der x86-Architektur nötig. Das Betriebssystem darf nicht mehr auf der privilegiertesten Hierarchiestufe, dem Ring 0, ausgeführt werden, sondern muss dem Hypervisor weichen, der diesen Platz einnimmt. Das Betriebssystem kann stattdessen auf Ring 1 verschoben werden, da dieser im Normalfall nicht belegt ist. Das Ergebnis zeigt Abbildung 2.9. Diese Modifikation ist allerdings bedingt

Neubelegung der x86-Ringe

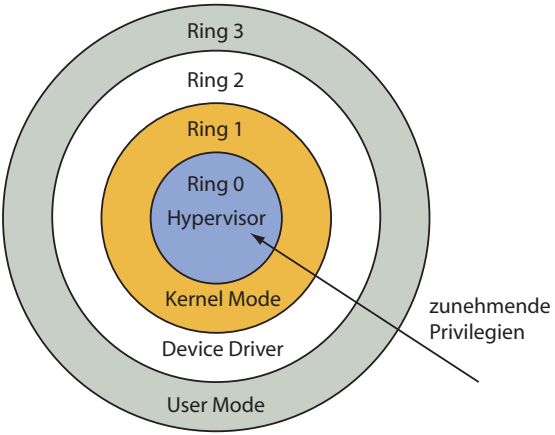


Abbildung 2.9: Das Ring Konzept der x86-Architektur modifiziert für Paravirtualisierung

durch die Architektur nur für Type 1 - Hypervisor möglich, welche für diese Arbeit aufgrund ihrer höheren Effizienz interessanter sind. Type II - Hypervisor arbeiten selbst auf einem Betriebssystem im Ring 3 als Applikation und besitzen daher eine zusätzliche Hierarchieschicht, die bei Kontextwechseln, Speicherzugriffen oder Systemaufrufen durchlaufen werden muss. Daher eignen sie sich hauptsächlich für den Einsatz am Desktop. Um den Hypervisor vom Gastbetriebssystem in diesen Systemen zu separieren, müssen zusätzlich virtuelle Privilegstufen eingeführt werden, da beide Komponenten in Ring 3 arbeiten, und das Kriterium der Isolation ansonsten verletzt wäre.

Entsprechend effektiv können durch das Anpassen der ABI auch Speicherzugriffe durchgeführt werden. Da die Übersetzung von virtuellen Speicheradressen der Gastbetriebssysteme in reale Adressen direkt vom Hypervisor vorgenommen werden kann, entfällt eine Hierarchiestufe, die in der generischen Übersetzungsreihen-

Effiziente Adressübersetzung

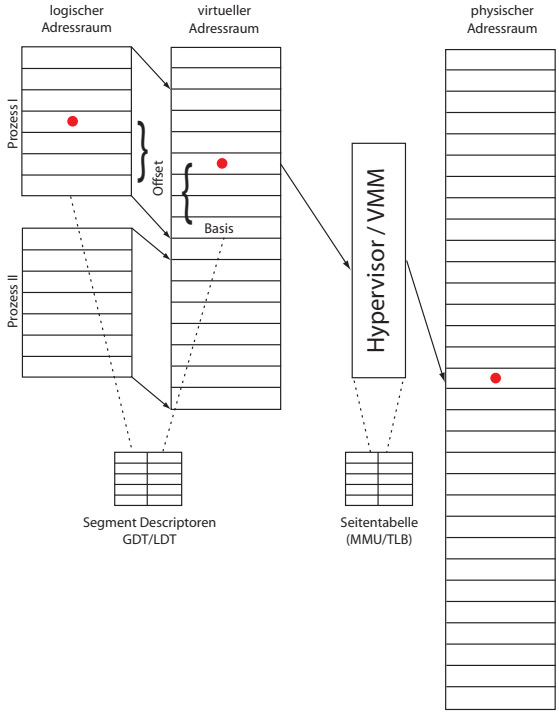


Abbildung 2.10: Adressübersetzung in einem paravirtualisierten System

Betriebssystem tritt rechte an den Hypervisor ab

folge (siehe Abbildung 2.5) notwendig ist, vollständig. Jeder von einem Gastsystem initiierte Speicherzugriff muss vom Hypervisor auf seine Gültigkeit geprüft und anschließend ausgeführt oder verweigert werden. Das Anlegen von neuen Page Tables kann vom Gastbetriebssystem wie gewohnt durchgeführt werden, allerdings muss nach dem Anlegen der Datenstruktur das Schreibrecht auf diese an den VMM abgetreten werden, so dass Page Tables ausschließlich vom Hypervisor initiiert durch Hypercalls verändert werden können und eine Speicherzugriffskontrolle erfolgen kann. Während eines Kontextwechsels zwischen virtuellen Maschinen muss der Translation Lookaside Buffer (TLB), welcher einen Cache der Memory Management Unit (MMU) realisiert, geleert werden, da sein Inhalt auf einer falschen Basisadresse für das Page Directory basiert und damit auf Adressen von anderen virtuellen Maschinen verweisen würde.

2.5 Vollvirtualisierung

Platzierung des Virtual Maschine Monitors

Bei der Vollvirtualisierung handelt es sich um eine effizientere Variante der Emulation, die lediglich Unterstützung für Gastsysteme bietet, die auf derselben Architektur basieren wie das zugrundeliegende physische System. Diese Einschränkung reduziert die Übersetzung der Befehle des Gastbetriebssystem in die darunterliegende Architektur in eine relativ einfache Abbildung, bei der lediglich einige kritische Instruktionen ersetzt werden müssen. Vollvirtualisierung ist ebenso wie Paravirtualisierung auf zwei Hierarchieebenen realisierbar. Die erste Möglichkeit besteht darin, den Hypervisor auf der Hardware zu platzieren. In diesem Fall spricht man von einem Type 1 - Hypervisor (siehe Abbildung 2.11). Die zweite

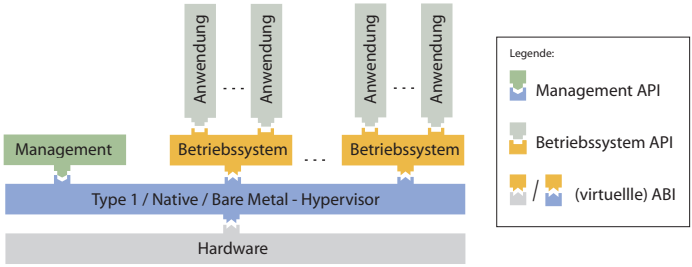


Abbildung 2.11: Architektur eines vollvirtualisierten Systems - Type1

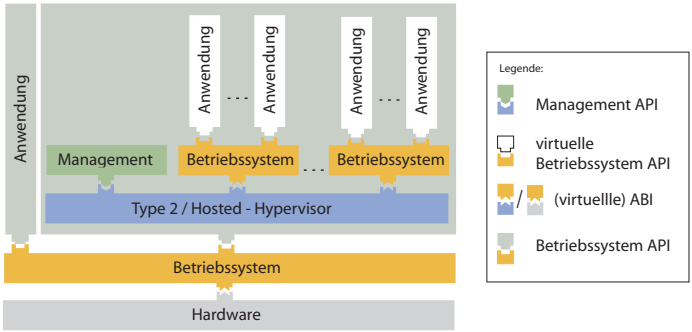


Abbildung 2.12: Architektur eines vollvirtualisierten Systems - Type2

Möglichkeit ist es, den Hypervisor als Applikation auf einem Betriebssystem zu

realisieren. In diesem Falle spricht man von einem Type 2 - Hypervisor (siehe Abbildung 2.12). In jedem Fall ist aber die vom Hypervisor bereitgestellte ABI identisch mit der des physischen Systems. Im Falle eines Typ 1 - Hypervisors werden die Gastbetriebssysteme in den Ring 3 verschoben und in diesem Kontext ausgeführt. Vor der Ausführung eines Codeblocks innerhalb einer virtuellen Maschine wird dieser zur Laufzeit durch den Virtual Machine Monitor auf Instruktionen untersucht, die nicht nativ ausgeführt werden dürfen. Diese werden durch einen Vorwärtssprung im Speicher ersetzt. An der referenzierten Speicherstelle wird alternativer Code, der meist keine privilegierten Instruktionen mehr enthält, eingefügt. Anschließend wird an die aufrufende Speicherstelle zurückgesprungen und die Ausführung dort fortgesetzt. Ein Übersetzungsblock endet meist an einer Stelle, an denen implizite Sprungbefehle stehen. Auf diese Art werden lediglich Instruktionen übersetzt, die tatsächlich ausgeführt werden. Nicht ausgeführter Binärcode oder Datenbereiche werden übergangen und sparen auf diese Art und Weise unnötige Übersetzungszeit.

„Binary Translation"

„Shadow Page Tables"

Für die Übersetzung von Speicheradressen der virtuellen Maschinen auf physische Adressen existieren mehrere Varianten. Die am häufigsten angewendete Art, die auch bei den VMware Produkten zum Einsatz gelangt, ist das Verwalten von „Shadow Page Tables" durch den Hypervisor. Diese vereinfachen die Abbildung von virtuellen Adressen innerhalb der virtuellen Maschinen in physische Adressen. Normalerweise sind für diese Übersetzung zwei Abbildungsschritte notwendig, deren Ergebnis jedoch in den „Shadow Page Tables" zwischengespeichert wird. Diese werden beispielsweise über den folgenden Mechanismus aktuell gehalten: Grundsätzlich darf das Gastbetriebssystem seinen Speicher selbst verwalten. Allerdings sind die Pagingtabellen des Gastbetriebssystems schreibgeschützt, so dass bei Updates vom Prozessor ein Trap ausgelöst wird. Dieser wird im Virtual Machine Monitor behandelt, anschließend werden die „Shadow Page Tables" aktualisiert. Auf diese Weise bleiben die Pagingtabellen synchron. Das Verfahren wird allgemein als „Trap and Emulate" bezeichnet. Lesender Zugriff auf die Pagingtabellen durch ein Gastbetriebssystem ist kein Problem mehr, da die Memory Management Unit (MMU) des Systems ebenfalls auf die Shadow Page Table des VMM zugreift, und damit die Adressen aller VMs kennt [Wald , AdAg 06, GSS 08]. Das Verfahren wird in Abbildung 2.13 dargestellt.

2.6 Native Virtualisierung

Optimierte Treiber mit erweiterter Funktionalität

Bei der Native Virtualisierung handelt es sich um einen hybriden Virtualisierungsansatz, der im Wesentlichen auf der Vollvirtualisierung basiert und aus Gründen der Performanz und der Funktionalität Elemente der Paravirtualisierung verwendet. Das Gastbetriebssystem selbst wird dabei nicht verändert, es kommen lediglich für die Virtualisierung optimierte Treiber im Gastsystem zum Einsatz. Diese sind darauf ausgelegt, möglichst wenige Kontextwechsel zwischen Virtual Machine Monitor und virtueller Maschine zu erzeugen, und damit den Overhead zu re-

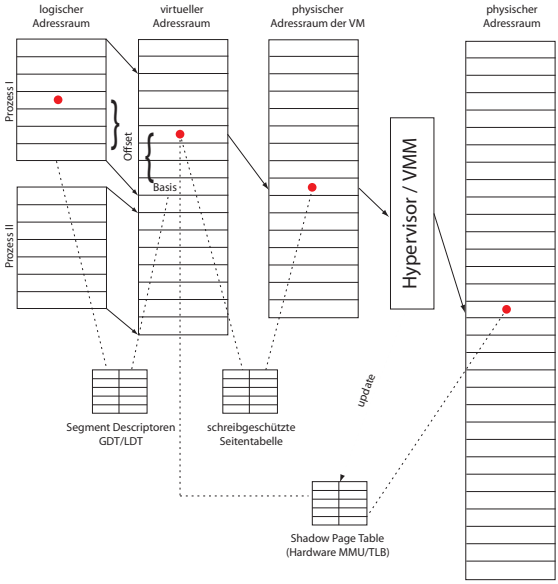


Abbildung 2.13: Adressübersetzung in einem vollvirtualisierten System

duzieren. Das Anpassen von intern übertragenen Block- und Paketgrößen ist hierbei ebenso wie die effektive Verwendung von Instruktionen, die Interrupts und Exception auslösen, im Blickfeld. Zusätzlich erlauben diese Treiber meist eine Änderung der Hardwarekonfiguration der virtuellen Maschine zur Laufzeit. Neben Änderungen, die auch ein nicht virtualisiertes Betriebssystem unterstützt - zum Beispiel das Anschließen eines USB-Gerätes im laufenden Betrieb, können diese Treiber auch dynamisch Änderungen an der Größe des Hauptspeichers vornehmen und diesen den tatsächlichen Anforderungen anpassen. Diese Technik ermöglicht unter anderem eine Überbuchung bei der Ressourcenzuteilung, da die tatsächliche Größe des Speichers erst dann benötigt wird, wenn das Gastsystem darauf zugreift. In der Hoffnung, dass die Lastspitzen der laufenden virtuellen Maschinen statistisch unabhängig sind, können so teure physische Ressourcen gespart werden. Mittlerweile existieren für alle etablierten Vollvirtualisierer solche Treiberoptimierungen, so dass die reine Vollvirtualisierung im produktiven Umfeld nicht mehr zum Einsatz kommt und durch Native Virtualisierung verdrängt wird.

Kosteneinsparung durch Überbuchung

2.7 Hardware-gestützte Virtualisierung

Korrekturen an der x86-Architektur

Wie aus den bereits vorgestellten Virtualisierungsansätzen ersichtlich ist, lösen alle Implementierungen von Virtualisierern das Problemen der Virtualisierbarkeit der x86-Architektur auf zum Teil sehr unterschiedliche Weise. Ein zusätzlicher und allgemein gültiger Lösungsweg, die x86-Architektur zu virtualisieren, ist es, den verwendeten Befehlssatz so zu verändern, dass er virtualisierbar wird und der Popek/Goldberg-Bedingung genügt. Der eigentliche Befehlssatz darf dabei nur ergänzt, aber nicht verändert werden, da er ansonsten nicht mehr kompatibel zu bestehenden Anwendungen wäre. Sowohl Intel als auch AMD haben diese Ergänzungen an ihren CPUs implementiert. Ihre Realisierungen werden im Folgenden kurz erläutert:

2.7.1 Intel VT-x

Neue Modi: *VMX root* und *VMX nonroot*

Die Virtualisierungsunterstützung im Prozessor wird durch einen Befehlssatz bereitgestellt, den Intel Virtual Maschine Extension (VMX) taufte und unter dem Codenamen Vanderpool entwickelte. Bei VMX existieren zwei Modi: *VMX root* und *VMX nonroot*. Ersterer Modus ist für die Ausführung des Hypervisors gedacht. Alle virtuellen Maschinen laufen im Modus *VMX nonroot*. Das Verhalten des Prozessors im Modus *VMX root* ist im Wesentlichen identisch mit dem herkömmlicher Prozessoren ergänzt mit einigen speziellen Befehlen, die zur Virtualisierung notwendig sind. Der Modus *VMX nonroot* hingegen entspricht einer limitierten Prozessorumgebung, in der das Ausführen bestimmter Befehle nicht möglich ist. Um den VMX-Befehlssatz zu aktivieren, muss der VMM zunächst das Bit 13 im CR4, auch CR4.VMXE genannt, setzen. Anschließend startet das Kommando VMXON die Befehlserweiterung VMX. VMX unterstützt nur den Page-protected Mode zur Speicherverwaltung, die Modi Unpaged-protected Mode sowie der Real Address Mode werden in Kombination mit VMX nicht unterstützt. Enthalten die Register CR0.PE und CR0.PG dennoch entsprechende Werte, wird VMX nicht aktiv. Nach dem Start befindet sich der Prozessor immer im Modus VMX root, so dass ein VMM, der VMX aktiviert hat, die Kontrolle über das gesamte System erlangt. Zur Ausführung von virtuellen Maschinen wird in den Modus *VMX nonroot* geschaltet. Hierfür steht die VMX transition VMentry zur Verfügung. Der Befehl VMexit aktiviert den Modus *VMX root* wieder und gibt die Kontrolle zurück an den Hypervisor (siehe Abbildung 2.14). Auf diese Weise hat der Hypervisor volle Kontrolle über das System, während die virtuellen Maschinen in ihren Aktionen begrenzt sind, obwohl deren Betriebssystem im Ring 0 läuft, der bei den Prozessoren auch Current Privilege Level (CPL) 0 heißt. Damit entfällt die Anpassung des Betriebssystems an andere CPLs. Da es auch kein softwarelesbares Register gibt, das den Modus *VMX nonroot* verraten würde, ist die Virtualisierung für das Betriebssystem transparent.

Kontextwechsel zwischen VMs ähnelt Scheduling in Betriebssystemen

Versucht ein Betriebssystem einen privilegierten Befehl auszuführen, erfolgt seitens des Prozessors ein VMexit, so dass der VMM den Aufruf ordnungsgemäß behandeln kann. Dabei wird eine Art Prozesswechsel durchgeführt, der Ähnlichkeit mit dem des Schedulings hat. Dies funktioniert nur ordentlich, wenn sich der Prozessor beim nächsten Aktivieren eines Prozess im selben Zustand befindet wie vor dem Wechsel. Aus diesem Grund wurde der Virtual Maschine Control State (VMCS) eingeführt. Er besitzt ähnliche Aufgaben wie das Programm Status Wort (PSW) beim Scheduling. VMCS stellt eine Kontrollstruktur bereit, in der zum Beispiel der Status eines Prozessors abgelegt werden kann. Das VMCS ist ein 4KB großer Block im Hauptspeicher des Rechners, der über den VMCS-Pointer des physischen Prozessors referenziert wird. Pro virtuellem Prozessor kann ein VMCS angelegt werden, das mittels VMPTRLD gefolgt von der Adresse des VMCS aktiv wird. VMPTRLD steht für Load Pointer to Virtual Maschine Control Structure und setzt den VMCS-Pointer eines physischen Prozessors. Entsprechend speichert VMPTRST den VMCS-Pointer und VMCLEAR deaktiviert das VMCS. Das VMCS besteht aus drei Bereichen:

- revision identifier, gibt die Version der VMCS-Struktur an.
- abort indicator, speichert einen Abbruch-Wert
- data, enthält den Prozessorstatus und lässt sich weiter unterteilen in:
  - Guest-state area
  - Host-state area
  - VM-execution control fields
  - VM-exit control fields
  - VM-entry control fields
  - VM-exit information fields

Beispielsweise enthält die Guest-state area unter anderem die gesicherten Prozessorregister und VM-exit information fields den Grund der VMX transition. Die genauen Bedeutungen der einzelnen Felder können in der Spezifikation von Vanderpool [INTE 05] nachgelesen werden.
Um einen VMCS zu verwenden, benutzt man das Kommando VMLAUNCH. Er startet eine virtuelle Maschine unter der Kontrolle von VMCS. Um eine Maschine nach einer VMX transition erneut auszuführen, dient das schnellere Kommando VMRESUME. Bei der Verwendung ist darauf zu achten, dass ein VMCS nie von mehreren Prozessoren gleichzeitig verwendet wird. Aus diesem Grund muss bei der Migration von Maschinen auf andere Prozessoren, um zum Beispiel Loadbalancing durchzuführen, der erste Prozessor vor der Migration den VMCS mittels VMCLEAR freigeben. Nach der Migration kann mittels VMLAUNCH und im Folgenden VMRESUME mit demselben VMCS weitergearbeitet werden.
Ein vereinfachter Wechsel zwischen den Modi *VMX root* und *VMX nonroot* mittels der Befehle VMentry und VMexit wird von Abbildung 2.14 dargestellt. Nicht

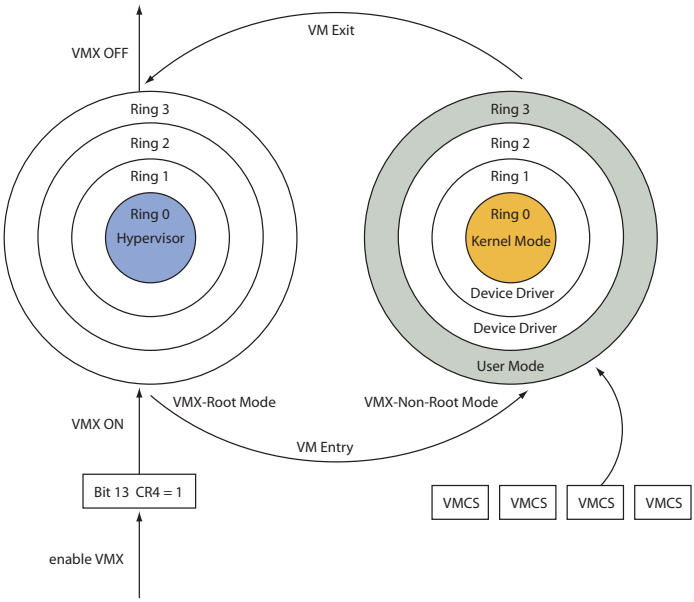


Abbildung 2.14: Vereinfachter Kontextwechsel zwischen *VMX root*und *VMX non-root* bei Intel Vanderpool

dargestellt wird dagegen die Aktivierung der verschiedenen VMCS-Instanzen vor dem Wechsel in den Modus *VMX root* mittels VMentry. Hierzu ist bei bestehendem VMCS die Instruktion VMRESUME, ansonsten das Kommando VMLAUNCH auszuführen.
Durch den Einsatz von VMX können Hypervisor deutlich unkomplizierter realisiert werden. Durch die Hardware basierte Ausführung von Befehlen, die zur Virtualisierung benötigt werden, lässt sich auch die Performanz steigern. So sind in Vergleichsmessungen mit VMX um ein vielfaches weniger Aktionen des VMM nötig als ohne. Die Tests SYSmark Internet und SYSmark Office enthalten eine Reihe für Internet und Office Anwendungen typischen Aktionen. Zu beobachten ist in beiden Tests vor allem eine signifikante Abnahme der Instruktionen im Bereich Interrupt Handling bei der Verwendung von Vanderpool. In den anderen Bereichen fällt der Unterschied nicht so groß aus, es sind jedoch trotzdem positive Veränderungen messbar. Detaillierte Messergebnisse werden in Kapitel 3 betrachtet.

Visionen

VMX ist darauf ausgelegt erweiterungsfähig zu sein. Noch virtualisiert Vander-

pool nur den Prozessor, das langfristige Ziel ist es jedoch klar, die gesamte x86-Plattform Hardware unterstützt zu virtualisieren. Speichercontroller, Netzwerkkarten, Storage-Subsystem, ja sogar Grafikkarten sollen in Zukunft Virtualisierungsunterstützung bieten. Damit könnte in letzter Konsequenz sogar der Einsatz eines Virtual Machine Monitors in Software, der zurzeit immer noch nötig ist, der Vergangenheit angehören. Ein Schritt in diese Richtung ist die kürzlich von Intel eingeführte I/O-Virtualisierung mit dem Namen Intel VT-d.

2.7.2 AMD-V

Unterschiede zu Intel VT-x

AMDs Lösung zur Virtualisierungsunterstützung in Prozessoren mit dem Codenamen Pacifica ist weitgehend identisch mit Vanderpool. AMD geht jedoch einen Schritt weiter als die Konkurrenz und stellt zusätzliche Funktionen bereit. Genauer genommen ist Vanderpool damit nur eine Teilmenge von Pacifica. Die Grundfunktionen von Pacifica, die auch Vanderpool bietet, sind bis auf die Namen der Register und Befehle weitgehend identisch. So nennt AMD seinen Befehlssatz Secure Virtual Maschine (SVM) und aktiviert den „Guest-Mode" mit dem Befehl VMRUN. Beendet wird er mit #VMEXIT. Das dem VMCS entsprechende VMCB, was für Virtual Maschine Control Block steht, existiert ebenfalls. Diese Unterschiede in der Namensgebung ziehen sich durch die gesamte Architektur. Im Weiteren werden daher nur noch die zusätzlichen Funktionen behandelt.

Virtualisierung des Speicher-Controllers

Der wichtigste Unterschied ist die zusätzliche Virtualisierung des Speicher-Controllers mit Pacifica. Dies ist bei AMDs Prozessoren weniger schwierig zu lösen als bei Intel, da die Prozessoren AMD Athlon und AMD Opteron bereits über einen integrierten Speicher-Controller verfügen. Während bei Intel bei jedem Speicherzugriff einer virtuellen Maschine auf eine nicht im Arbeitsspeicher befindliche Seite der VMM aktiv werden muss, um die Speicheradresse zu übersetzen und damit zwangsläufig einen VMexit auslöst, können bei AMDs Pacifica solche Speicherzugriffe von der Hardware abgefangen und übersetzt werden. Damit muss der Guest-Mode nicht verlassen werden und die Performanz erhöht sich.

Direct Memory Access und Trusted Platform Module

Ein zweiter Unterschied ist die Unterstützung von DMA (Direct Memory Access) Zugriffen aus virtuellen Maschinen über den sogenannten Device Exclusion Vector (DEV). Damit können DMA-fähige Geräte ohne Hilfe des Prozessors direkt auf Speicherbereiche der virtuellen Maschine zugreifen. Der DEV verhindert dabei den Zugriff auf den physikalischen Speicher des Gesamtsystems. Ein letzter Unterschied besteht in der Integration der Trusted-Computing-Technologie „Presido" in Pacifica. Daher trägt AMDs Technologie auch den Beinamen Secure Virtual Maschine Architecture. Für die eigentliche Virtualisierung sind diese Sicherheitsfunktionen unnötig, jedoch wird es mit ihrer Hilfe möglich, Software, die TPM (Trusted Platform Module)-Funktionen benötigt, in virtuellen Maschinen zu starten. Mit diesen Funktionen ist AMD seinem Konkurrenten Intel bereits einen kleinen Schritt in die Zukunft voraus.

2.8 OS-Virtualisierung

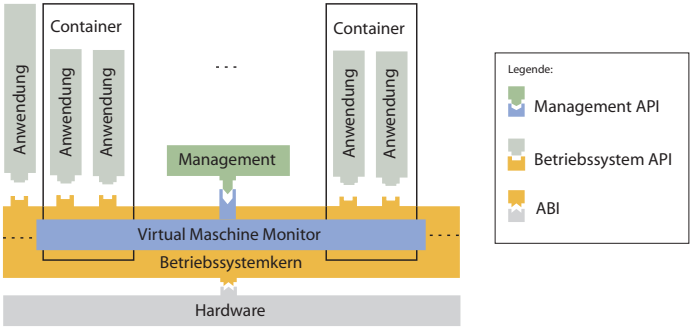


Abbildung 2.15: Architektur eines mit OS-Virtualisierung erzeugten Systems

Erzeugung von Containern durch Partitionierung

Die Techniken der OS-Virtualisierung unterscheiden sich konzeptionell grundlegend von den übrigen bereits vorgestellten Varianten, denn es wird nicht eine komplette virtuelle Maschine erzeugt, sondern lediglich ein bestehendes Betriebssystem in so genannte Container partitioniert, die hier der Einfachheit halber ebenfalls als virtuelle Maschine bezeichnet werden, aber einigen Einschränkungen gegenüber den bisher erläuterten virtuellen Maschinen unterliegen. Eine dieser Einschränkungen beruht darauf, dass sowohl das Gastbetriebssystem als auch alle virtuellen Maschinen mit derselben Kernelinstanz arbeiten. Der Virtual Machine Monitor, der bei diesem Virtualisierungsansatz nahezu vollständig in das Betriebssystem integriert wird, nimmt nur eine Trennung von Adressräumen vor, so dass virtuelle Maschinen auf ähnliche Art voneinander separiert werden können wie in herkömmlichen Betriebssystemen Prozesse. Aufgrund dieser Architektur entsteht zwar verhältnismäßig wenig Overhead, allerdings kann in allen VMs nur dasselbe Betriebssystem ausgeführt werden wie im eigentlichen System. Das geht so weit, dass in der Regel sogar der gleiche Patchlevel in den Systemen eingehalten werden muss. CPU und Hauptspeicher lassen sich damit sehr effizient virtualisieren, andere Peripheriegeräte, wie zum Beispiel Netzwerkkarten, werden in der Regel emuliert. Im Kontext von OS-Virtualisierung kann man auch nicht mehr vom Booten einer VM sprechen, sondern vielmehr vom Aktivieren, da der Kernel ja bereits läuft. Lediglich zusätzliche Verwaltungsinformationen und private Gerate, wie Netzwerkkarten, die durch Emulation bereitgestellt werden, müssen angelegt und initialisiert werden. Einem Container stehen grundsätzlich alle physischen Systemressourcen, wie Arbeitsspeicher und CPUs des Gesamtsystems, zur Verfügung. Eine dedizierte Zuteilung wie bei anderen Virtualisierungsarten ist nicht erforderlich, um aber den Ressourcenverbrauch einzelner Container einschränken zu können, wurden Mechanismen installiert, die das Quota einer Maschine beschränken können.

Unflexible Bindung an eine Betriebssystem Version

2.9 Einordnung von Produkten

Hybride Ansätze

Der folgende Abschnitt gibt eine kurze Einordnung existierender Produkte im Bereich Servervirtualisierung. Produkte aus dem Bereich der Desktopvirtualisierung wurden gezielt nicht aufgenommen, da sie für den Fokus dieser Arbeit nicht interessant sind. Bisweilen existieren Produkte, die nicht zu 100 % einer Virtualisierungstechnik zugeordnet werden können, sondern aus Gründen der Performanz mehrere Techniken parallel implementieren und somit einen hybriden Ansatz verfolgen. Diese wurden entsprechend ihrem Schwerpunkt in die jeweilige Kategorie eingeordnet.

2.9.1 Xen

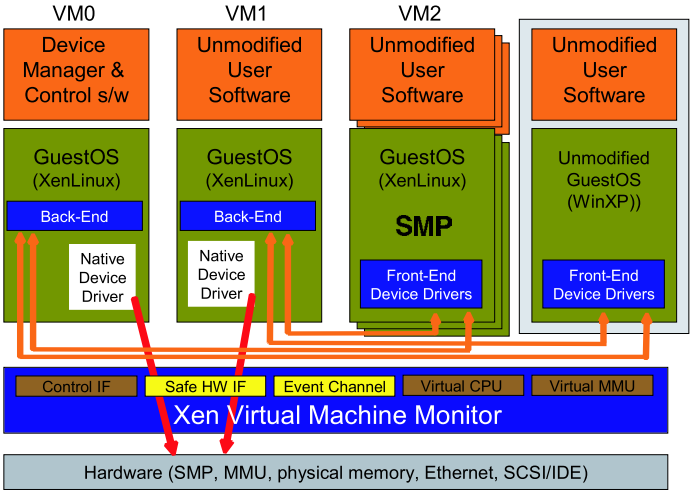


Abbildung 2.16: Die Architektur von Xen [Prat 05]

Paravirtualisierung und Hardware-gestützte Vollvirtualisierung

Xen wurde ursprünglich als Paravirtualisierer entwickelt, unterstützt seit der Version 3 aber auch Hardware-gestützte Virtualisierung, so dass der Betrieb von unmodifizierten Gastbetriebssystemen ebenfalls möglich ist. Wie bereits am Architekturbild 2.16 erkennbar ist, existiert eine privilegierte Maschine, genannt Domain0, der es erlaubt ist, den Hypervisor zu beeinflussen und die übrigen virtuellen Maschinen zu managen. Dieselbe Maschine dient in der Regel auch als Backend für Blockgeräte und Netzanbindung, so dass diese Funktionalität

nicht vom Hypervisor selbst erbracht werden muss. Dies vereinfacht die Wartbarkeit des Hypervisors, da er keine Treiber enthalten muss; diese sind stattdessen in der Domain0 enthalten. Im Paravirtualisierungsmodus läuft in den virtuellen Maschinen ein Linux Kernel, der an die veränderte ABI der x86-Architektur angepasst wurde und Frontend Treiber zum Zugriff auf das Backend enthält. Alle anderen Treiber können nativ verwendet werden, solange der Hypervisor Zugriff auf entsprechende Geräte gewährt. Im Vollvirtualisierungsmodus entfällt die Anpassung der ABI, da die Ausführung von nativen Betriebssystemen möglich ist. Im Gegenzug müssen allerdings Netzwerkkarten und Speichergeräte wie Festplatten und CD/DVD-Laufwerke sowie ein BIOS emuliert werden, damit herkömmlich Treiber diese ansprechen können. Zurzeit kommen für diesen Zweck Komponenten des Emulators Bochs/Plex86 zu Einsatz. Die eigentliche Emulation dieser Geräte erfolgt analog zum Paravirtualisierungsmodus durch die Domain0. Xen wird in zwei Varianten auf dem Markt angeboten: Eine frei verfügbare OpenSource Variante sowie eine durch Citrix unterstützte, kommerzielle Variante, die neben dem eigentlichen Xen auch Managementwerkzeuge enthält. Damit sind unter anderem Live-Migration und Lastausgleichsmechanismen realisierbar.

2.9.2 Hyper-V

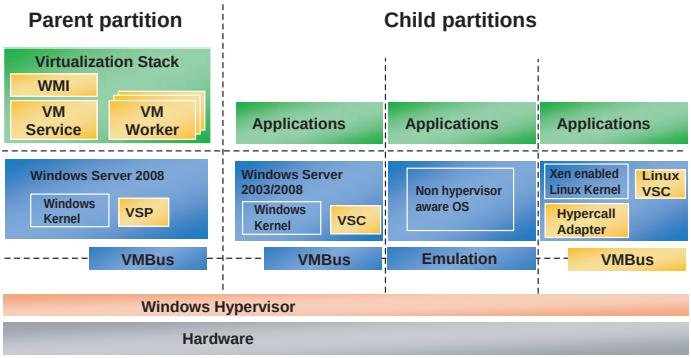


Abbildung 2.17: Die Architektur von Hyper-V [ARC 07]

Architekturelle Ähnlichkeit zu Xen

Hyper-V ist von seiner Architektur gesehen genau wie Xen ein Paravirtualisierer, der jedoch Gebrauch von Hardware-gestützten Virtualisierungsansätzen machen kann. Entwickelt wurde Hyper-V von Microsoft als Komponente für den Windows Server 2008. Bei der Entwicklung halfen jedoch die Entwickler von Xen, so dass die Ähnlichkeiten der beiden Architekturmodelle nicht verwundern. Bei Verwendung von Hyper-V können sogar unmodifizierte XenLinux Kernel mittels

eines integrierten Hypercall Adapters ausgeführt werden (siehe Abbildung 2.17). Die Kommunikation der virtuellen Maschinen bezüglich ihrer I/O-Zugriffe erfolgt analog zu Xen über den sogenannten SMBus. Die der Domain0 entsprechende Instanz nennt Microsoft „Parent Partition", während die übrigen Maschinen als „Child Partitions" bezeichnet werden. Da sich das Produkt zurzeit noch im Beta Stadium befindet, liegen leider noch keine weiteren Details vor.

2.9.3 VMware ESX Server / Virtual Infrastructure

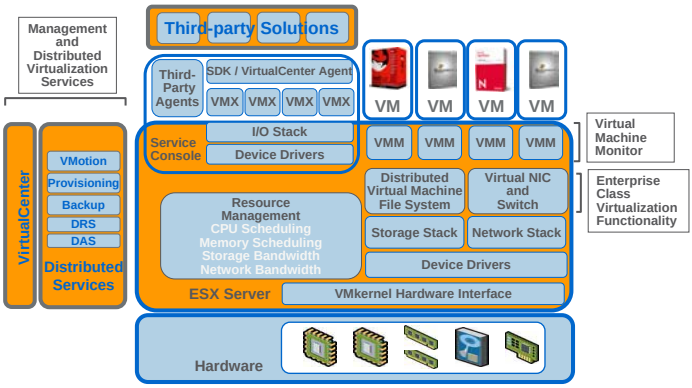


Abbildung 2.18: Die Architektur von VMware ESX / Virtual Infrastructure [Lo 05]

Typ 1 Vollvirtualisierung und reduzierter Hardware Support

Der ESX Server der Firma VMware stellt derzeit den Marktführer im Bereich Servervirtualisierung da. Der ESX Server ist ein Vollvirtualisierer vom Typ 1 (siehe Abbildung 2.18). Im Gegensatz zu Xen enthält bereits der Hypervisor Treiber zum Zugriff auf die Hardware. Aus diesem Grund ist die Liste der unterstützten Geräte und Server deutlich kürzer als verglichen mit Xen, das seine Geräte über die Treiber eines Linux Kernels anspricht und damit fast alle verfügbaren Geräte unterstützt. Diesen Nachteil in der Architektur gleicht VMware allerdings durch die umfangreichste Management Lösung mehr als aus. Kein anderes Produkt unterstützt Hochverfügbarkeitslösungen, Lastausgleich, Stromsparmechanismen sowie Backup & Recovery Funktionen in diesem Ausmaß. VMware vereint aus Gründen der Performanz mehrere Virtualisierungstechniken miteinander. Intern macht der ESX Server Gebrauch von Binär-Übersetzung, Trap and Emulate Techniken sowie optionaler Hardware Unterstützung durch Intel VT-x und AMD-V.

2.9.4 Virtuozzo / OpenVZ

Betriebssystem-virtualisierung: Hohe Effizienz bei eingeschränkter Flexibilität

OpenVZ ist eine Erweiterung des Linux Kernels, die es erlaubt, mehrere separierte OS-Instanzen zur selben Zeit auf demselben System zu betreiben. OpenVZ wird von einer freien Entwicklergemeinschaft unter der OpenSource Lizenz entwickelt. Das kommerzielle Virtuozzo baut auf diesem Projekt auf und stellt zusätzlich eine Managementoberfläche bereit. Des Weiteren wurde Virtuozzo mittlerweile auf Windows portiert und kommt vor allem im Bereich des Webhosting und VServer Marktes zum Einsatz, da der Ressourcenverbrauch einer Maschine sehr gering ist und die zwingende Homogenität der einzelnen Container keinen Nachteil darstellt. Auf entsprechenden Servern lassen sich zum Teil über hundert virtuelle Instanzen zur gleichen Zeit betreiben, was für derartige Rechenzentren einen gravierenden Kostenvorteil im Hinblick auf Hardwarekosten, Stellfläche, Kühlung und Personal impliziert. Virtuozzo unterstützt wie auch Xen und VMware Live-Migration von Containern, sofern Ziel- und Endsystem auf identischen Betriebssystemen laufen. Aufgrund der starren Einschränkungen in der Wahl des Betriebssystems und der vergleichsweise schwachen konzeptionellen Trennung der einzelnen Instanzen kommen sowohl Virtuozzo und OpenVZ in Unternehmen eher selten zum Einsatz. Neben Virtuozzo und OpenVZ existieren noch weitere Produkte, die auf der Betriebssystemvirtualisierung aufbauen. Erwähnenswert sind hier etwa FreeBSD, Jails oder Solaris Containers, die im Folgenden jedoch nicht weiter betrachtet werden.

2.10 Zusammenfassung

Die vorgestellten Konzepte zur Virtualisierung von Maschinen unterscheiden sich in ihrem Ansatz grundsätzlich. Nichtsdestotrotz haben sich bereits hybride Virtualisierungsansätze gebildet, die nicht mehr exakt eine Technik zur Virtualisierung verwenden, sondern aus Gründen der Effizienz für unterschiedliche Teilaufgaben unterschiedliche Techniken verwenden. Diese Tatsache wird im Abschnitt 3.2 behandelt. Ein weiterer Unterschied der unterschiedlichen Virtualisierungsansätze ist Flexibilität, die sich meist indirekt proportional zur Effizienz eines Ansatzes verhält. So kann bei der OS-Virtualisierung, bei der alle virtuellen Instanzen auf exakt einen Kernel zugreifen auch nur eine Sorte an Gastbetriebssystemen ausgeführt werden, dafür ist der anfallende Mehraufwand für die Virtualisierung entsprechend klein. Ebenso verhält es sich mit der Paravirtualisierung, die auf portierte Betriebssystem Kernel angewiesen ist und in Kauf nimmt, dass die Virtualisierung durch diese Maßnahme nicht mehr transparent für Betriebssysteme und Applikationen ist.

Trotz der Unterschiede in der reinen Virtualisierungstechnik bieten beinahe alle kommerziellen Anbieter von Virtualisierungslösungen funktional annähernd identische Funktionen an. Zurzeit ist insbesondere im Bereich der Managementfunktionalität VMware der Marktführer, der mit seinem Ressourcen Management System

die Richtung für andere Hersteller festlegt. Insbesondere im Managementbereich sind jedoch noch weitreichende Entwicklungen notwendig, um Virtualisierung in das Management großer Infrastrukturen zu integrieren, da eine einfache Betrachtung von virtuellen Maschinen nicht ausreichend ist, um zum Beispiel Netztopologien zu erfassen, die durch virtuelle Bridges und Router instanziiert werden.

3 Leistungsverhalten im Kontext von Virtualisierung

Inhalt

Dieses Kapitel definiert zunächst einen klassischen Leistungsbegriff angelehnt an die Arbeiten von [Tane 02, HePa 94] und diskutiert im Anschluss daran unter Verwendung der im vorstehenden Kapitel dargelegten Grundlagen der Virtualisierung, wie dieser im Kontext der Virtualisierung verwendet werden kann und welche Unterschiede zum nicht-virtualisierten Fall bestehen. Ist erst einmal ein Leistungsbegriff im Kontext der Virtualisierung definiert, so können darauf aufbauend Einflussfaktoren auf das Leistungsverhalten von Virtualisierung auf der Basis von Virtualisierungstechniken untersucht und gemessen werden. Auf Basis

dieser Analysen und Messungen können anschließend Vergleiche bestehender Virtualisierungstechniken betreffend deren Leistungsverhalten und Optimierungen durchgeführt werden.

3.1 Leistungsbegriffe

Leistung der Komponenten vs. Leistung des Gesamtsystems

In vielen Bereichen unserer Umwelt hat sich in der Vergangenheit ein Leistungsbegriff entwickelt. Beispiele für solche Bereiche sind Autos, Mitarbeiter oder Aktiengesellschaften. Allen Leistungsbegriffen ist gemein, dass sie unter Leistung eine zu erbringende Arbeit pro Zeiteinheit verstehen. Bereits der Begriff Arbeit ist jedoch schon wesentlich schwächer definiert. Im Falle des Autos könnte man darunter zum Beispiel die Anzahl der erbrachten Umdrehungen des Motors oder aber die Anzahl der erbrachten Umdrehungen der Reifen verstehen. Mit ersterer Definition erhält man die Komponentenleistung des Motors, mit zweiterer die Systemleistung des Autos bestehend aus dem Produkt der Komponenten Motor und Getriebe. Letztendlich ist es eine Frage des Einsatzzweckes, welche Definition besser geeignet ist. Darüber hinaus existiert zusätzlich zu den unterschiedlichen Leistungsbegriffen der Begriff des Wirkungsgrades, welcher einheitlich als Quotient aus erbrachter und eingesetzter Leistung definiert ist. Im Beispiel des Autos ist der Wirkungsgrad durch den Spritverbrauch gegeben. Damit ist der Begriff des Wirkungsgrades nicht nur direkt abhängig vom gewählten Leistungsbegriff, sondern ebenso von variablen Einflussfaktoren, wie der Fahrweise des Fahrers.

Die eben angerissenen Definitionen werden in den folgenden Abschnitten zunächst für klassische, sprich physische Computersysteme, und anschließend für virtuelle Maschinen diskutiert. Das Ergebnis ist eine Art Leistungsfunktion, anhand der sowohl eine Einschätzung der Leistung von Virtualisierungstechnologien und Produkten erfolgen kann als auch die Grundlage für empirische Leistungsmessungen darstellt.

3.1.1 Klassischer Leistungsbegriff

In der Informatik existiert seit der Erfindung der ersten elektrisch betriebenen Rechenmaschinen der Bedarf die Leistung verschiedener Systeme zu messen und zu vergleichen. Während Messungen bei den ersten Batch-Maschinen verhältnismäßig einfach und intuitiv durchführbar waren, sind heutige Systeme um Größenordnungen komplexer und ihre Leistung auch nicht mehr intuitiv und eindeutig vergleichbar, da unterschiedliche Vorstellungen von Leistung existieren. Die verschiedenen Leistungsbegriffe werden im Folgenden diskutiert.

3.1.1.1 Begriffsbildung

Physikalischer Leistungsbegriff

Wie bereits erwähnt, ist der Begriff Leistung in der Informatik an die Definition des physikalischen Leistungsbegriffes angelehnt, nämlich verrichtete Arbeit pro Zeiteinheit.

$$Leistung = \frac{Arbeit}{Zeit} \tag{3.1}$$

Während die Leistung einer einzelnen Komponente wie zum Beispiel eines Speichermoduls dadurch intuitiv definierbar ist, ist die Leistung eines Systems wesentlich schwieriger zu fassen. Die Leistung des Speichermoduls hängt offensichtlich hauptsächlich von den Parametern Busbreite und Taktfrequenz ab, während die Systemleistung abhängig von der Leistung aller verbauten Komponenten ist. Da in keinem System alle Teilkomponenten immer gleichzeitig aktiv sind und im Programmfluss meist wechselweise Gebrauch von unterschiedlichen Komponenten gemacht wird und auf diese Weise ein Kommunikations-Overhead entsteht, wird die Leistung des Gesamtsystems im Allgemeinen immer kleiner sein als die Summe der Leistungen aller Komponenten.

$$Leistung_{System} \leq \sum_{i=1}^{Anzahl} Leistung_{Komponente_i} \tag{3.2}$$

Leistung wird durch Anwendungen definiert

Die Leistung eines Systems hängt daher nicht nur von den einzelnen Komponentenleistungen ab, sondern auch stark von den jeweiligen Anwendungen, die auf dem System laufen, und der Definition des Arbeitsbegriffes. Daher ist es ein gängiger Ansatz, Arbeit in Form von Benchmarks plattformunabhängig zu standardisieren. Damit wird der Begriff Arbeit zur Konstante, während die Zeit für die Abarbeitung der standardisierten Aufgabe gemessen werden kann. Die Leistung als fehlende dritte Größe kann somit theoretisch abhängig von einem Arbeitsbegriff errechnet werden. Für einen definierten Benchmark ist diese Berechnung aber unnötig, da bereits die gemessene Zeit ein valides Leistungsmaß darstellt. Der Vergleich zweier Leistungen, die auf demselben System mittels unterschiedlicher Benchmarks und damit unterschiedlichen Arbeitsbegriffen gemessen wurden, ist mit diesem Ansatz jedoch nicht trivial möglich, da der Arbeitsbegriff damit implizit zum Tupel mutiert und die beiden Arbeitsbegriffe somit nicht länger skalar vergleichbar bleiben. Hier unterscheiden sich die verwendeten Leistungs- und Arbeitsbegriffe auch von den entsprechenden Termini der Physik, da im Falle von Rechensystemen mehrere Arten von Arbeit existieren, die zueinander nicht kompatibel sind und sich auch in ihrer Einheit von den klassischen Einheiten unterscheiden.

Messung über Laufzeit

Die Beurteilung der Leistung eines Systems erfolgt heute meist über die benötigte Zeit, die für die Ausführung einer definierten Aufgabe benötigt wird. Dabei unterscheidet man die Begriffe „Ausführungszeit", „Antwortzeit" und „Durchsatz". Unter der Ausführungszeit versteht man die Zeit, die ein Prozess während seiner gesamten Abarbeitungszeit aktiv war. Die Antwortzeit ist diejenige Zeit, die Nutzer auf das Ergebnis ihrer Berechnung warten. Die Abarbeitungszeit entspricht der Antwortzeit eines Prozesses abzüglich seiner Wartezeit. Der Begriff Durchsatz ist hauptsächlich für Betreiber von großen Batch-Systemen interessant, die die Anzahl der abgearbeiteten Jobs pro Zeiteinheit maximieren wollen. So kann es sein, dass ein I/O-intensives Programm eine sehr kurze Ausführungszeit, aber eine sehr lange Antwortzeit hat, da es die meiste Zeit auf I/O-Daten wartet. Berechnet man mehrere Programme gleichzeitig unter der Verwendung von Scheduling-Mechanismen, so kann solch ein System trotzdem einen hohen Durchsatz erzielen, da sowohl CPU als auch I/O-System gleichzeitig durch verschiedene Prozesse ausgelastet werden können anstatt nur wechselweise. Je nachdem welche der drei Kriterien optimiert werden sollen, implementiert man unterschiedliche Klassen von Schedulern für Stapelverarbeitung oder interaktive Systeme. Interaktive Systeme versuchen eine möglichst geringe Antwortzeit zu erreichen, während Stapelverarbeitungssysteme den Durchsatz optimieren. Daneben existiert eine weitere Klasse an Schedulern, die versucht Echtzeitanwendungen zu unterstützen. Hierbei kommt es weniger auf die Minimierung von Antwortzeiten oder auf die Erhöhung des Durchsatzes an, sondern vielmehr auf die exakte Einhaltung von Timingvorschriften. Ansonsten würden beispielsweise Videos zu schnell oder zu langsam abgespielt oder Bild und Tonspuren würden unterschiedlich schnell wiedergegeben und würden auseinander laufen.

Objektive Leistung ist kein Skalar

Moderne Betriebssysteme für den Endanwender stehen vor dem Problem, dass sie all diese Szenarien gleichermaßen gut unterstützen sollen. Da dies offensichtlich zu einem Trade-off führt, können diese Systeme nie die Leistung von auf eine Klasse spezialisierten Systemen erreichen. Die Leistung eines Systems kann aus diesem Grunde ebenso wenig wie die Arbeit als Skalar angegeben werden, sondern muss als Tupel betrachtet werden. Die Einträge dieses Tupels sind die Einflussfaktoren der Komponentenleistungen, beispielsweise CPU-Takt, Cache-Größe, Bus-Takt, Bus-Breite, Speicher-Takt, Netzbandbreite, Netz-Latenzzeiten und so weiter. Weitere Einflussfaktoren sind beschleunigende Erweiterungen wie zum Beispiel Pipelining und Sprungvorhersagemechanismen, die Art des Prozessormodels (RISC oder CISC) und der Scheduling-Algorithmus selbst. Die anteilige Gewichtung der einzelnen Komponenten an der skalaren Systemleistung wird implizit durch einen Benchmark festgelegt.

Metriken

Der Leistungsvergleich von Systemen über ihre Tupel ist zwar korrekt, aber sehr aufwendig, weshalb man an skalaren Kenngrößen zu Vergleichszwecken interessiert ist. Diese lassen sich für eine exakt definierte Aufgabe, welche man meist in Form eines Benchmarks realisiert, über ihre Antwortzeit realisieren. Mathematisch gesehen stellt die Aufgabe beziehungsweise der Benchmark eine einer Metrik ähnliche Abbildung dar, die den Leistungsvektor in einen Skalar überführen.

Man muss sich jedoch darüber im Klaren sein, dass der Wechsel auf eine andere Metrik auf demselben System ein stark abweichendes Leistungsmaß ergeben kann, da eventuell unterschiedliche Komponenten des Systems unterschiedlich stark gewichtet werden oder einander beeinflussen. Da die Leistungsmessung mittels Benchmarks auf Applikationsebene erfolgt, werden anwendungsspezifisch genau die Komponenten bzw. deren Zusammenspiel gemessen, die für die Applikation relevant sind. Dabei können auch einzelne Komponenten des Gesamtsystems brach liegen, wenn sie zur Abarbeitung des Benchmarks keinen Beitrag leisten. Dies ist ein weiterer Grund, warum die Leistung eines Systems bezogen auf eine Anwendung kleiner sein wird als die entsprechende theoretische Systemleistung.

Leistungsverhalten von Infrastrukturen

Noch einen Schritt weiter geht man, wenn man das Leistungsverhalten einer kompletten Infrastruktur untersuchen möchte. Neben den einzelnen Systemleistungen beeinflussen nun zusätzlich Netzkoppelkomponenten, die Art der Vermaschung, die Intensität der Kommunikation zwischen den Systemen und im Falle von Hochleistungssystemen Parallelisierbarkeit von Programmen das Ergebnis. Zurzeit sind lediglich Betreiber von Hochleistungssystemen an Leistungsmessungen für Infrastrukturen interessiert, da moderne Großrechner de facto Netze darstellen. Im Zuge von Virtualisierung möchte man jedoch nicht nur Systeme, sondern komplette Infrastrukturen virtuell bereitstellen, so dass dieser Gesichtspunkt verstärkt in den Vordergrund gerät. Da Endsysteme in einem gewissen Abstand der Entwicklung im Hochleistungsrechnen folgen werden, ist es nur eine Frage der Zeit, bis entsprechende Infrastrukturprobleme Einzug in Rechnerarchitekturen für Endsysteme halten. Die ersten Schritte hierzu sind mit Multicore-Architekturen und deren Anbindung an den zentralen Hauptspeicher bereits Realität. Auch die Leistung von Infrastrukturen ist in der Regel stark abhängig von der oder den Applikationen, die darin laufen, und kann zurzeit mangels verteilter Benchmarks nicht fundiert gemessen werden.

Flaschenhälse

Unabhängig davon, ob die Leistung von Systemen oder Infrastrukturen analysiert werden soll, wird im es Allgemeinen immer der Fall sein, dass die Gesamtleistung von einer einzelnen Komponente limitiert wird und den Flaschenhals bildet. Im Falle der bereits diskutierten x86-Architektur liegt dieser Flaschenhals in der Regel in der Speicheranbindung. Wo der Engpass in einem gegebenen System zu suchen ist, ist allerdings stark softwareabhängig. Abhängig von der zu lösenden Aufgabe werden gewisse Komponenten öfter als andere bei der Abarbeitung des Prozesses benutzt und stehen unter unterschiedlich hoher Last. Da die vorhandenen Komponenten letztendlich alle über denselben Systembus kommunizieren, ergibt sich eine weitere Abhängigkeit der Komponenten untereinander. All diese Einflussfaktoren sind zwar theoretisch nachvollziehbar, eine Gesamtleistung kann aufgrund der Komplexität der Abhängigkeiten aller Einflussfaktoren nicht mehr berechnet werden, zumal die theoretischen Leistungsangaben der Komponenten bereits geschönt sind nur unter sehr optimalen Bedingungen erreichbar sind. Ein Beispiel hierfür sind DDR2 SDRAM Speichermodule. Im Gegensatz zu DDR SDRAM ist bei gleicher Taktfrequenz der Speicherriegel der Systembus mit der doppelten Taktfrequenz getaktet. Um den Systembus auszulasten werden bei einem Speicher-

zugriff nicht nur zwei Bits, sondern automatisch vier Bits pro Zelle übertragen. Im Falle von DDR3 SDRAM sind es sogar acht Bits pro Zyklus. Diese Daten werden übertragen unabhängig davon, ob sie von der CPU jemals benötigt werden, denn angefordert wurden nur die ersten beiden Bits. In der zum Teil berechtigten Annahme, dass benachbarte Speicherzellen ebenfalls in Kürze benötigt werden, steigert man in dieser Situation die Leistung des Speichermoduls durch Prefetch-Techniken auf bis zu 400%. Der Speicher ist deswegen aber nicht wie von den Herstellern fälschlicherweise angegeben viermal so schnell. Die Leistung eines komplexen Systems kann aus den genannten Gründen nur noch empirisch bestimmt werden. Im folgenden Abschnitt wird die Messbarkeit von Leistung daher detaillierter diskutiert.

3.1.1.2 Messbarkeit

Historische und moderne Messverfahren

Bereits in den 70er Jahren wurden Messungen der Performanz an Computersystemen durchgeführt. Das Ergebnis wurde in MIPS (Million Instructions Per Second) oder FLOPS (Floating Point Operations Per Second) angegeben. Alleine die Einheit der Ergebnisse zeigt, dass diese Messungen lediglich CPU-basiert waren und wenig über die reale Rechenleistung des Systems aussagen. Anfang der 80er Jahre kamen daher die sogenannten Whetstone und Dhrystone Benchmarks auf, die bereits einen Mix aus Gleitkommazahl-Operationen, Integer-Arithmetik und Zugriffe auf Array-Elemente durchführten und auf diese Weise auch die Anbindung des Hauptspeichers in die Berechnung einbezogen. Seit 1989 existieren anwendungsbasierte Benchmarks, die von der SPEC [spe 08] (Standard Performance Evaluation Corporation) entwickelt wurden, mittlerweile existieren aber auch viele Anwendungsbenchmarks von Drittherstellen. Dabei wird ein für eine Anwendung typisches Nutzungsprofil durchgeführter Operationen erstellt, das anschließend durch eine Softwarekomponente auf dem zu testenden System simuliert wird. Das Ergebnis wird in Punkten gemessen und kann direkt mit Messergebnissen anderer Systeme verglichen werden. Zurzeit existieren unter anderem SPEC Benchmarks für:

- CPU
- Graphics/Workstations
- MPI/OMP
- Java Client/Server
- Mail Servers
- Network File System
- Power
- SIP
- Web Servers

Anwendungsorientierte Benchmarks sind unflexibel

Damit lassen sich sowohl Leistungsmessungen spezieller Komponenten als auch weit verbreiteter Anwendungen, insbesondere Server-Anwendungen durchführen. Anwendungen, für die kein Benchmark existiert oder die sich im konkreten Fall in ihrem Verhalten nicht durch einen standardisierten Benchmark beschreiben lassen, weil zum Beispiel das simulierte Nutzerverhalten Unterschiede aufweist, lassen sich durch Benchmarks nicht bewerten. Wenn man allein einen Ausschnitt einer Liste der am Leibniz-Rechenzentrum betriebenen Dienste betrachtet, so wird man feststellen, dass mit den verfügbaren SPEC Benchmarks nicht einmal die Hälfte aller Dienste evaluiert werden kann.

- DNS-Server
- DHCP-Server
- File-Server
- Print-Server
- Mail-Server
- Web-Server
- Datenbank-Server
- Terminal-Server
- Streaming-Server (Multimedia)
- Verzeichnisdienste
- Backup-Server
- Monitoringdienste
- Softwareverteilungsdienste
- Firewall
- Intrusion Detection Systeme

Benchmarks für Virtualisierung

Darüber hinaus stellt sich das Problem, dass keiner der verfügbaren Benchmarks auf die Eigenheiten von virtuellen Maschinen oder Virtualisierung im Allgemeinen eingeht. In dieser Arbeit wird daher folgender Ansatz gewählt, um allgemeine Aussagen zur Leistungsfähigkeit der Virtualisierungsansätze zu erhalten:
Die Messungen werden mit Benchmarks durchgeführt, die jeweils exakt eine virtualisierte Komponente des Systems einzeln analysieren. Exemplarisch werden diese Messungen an den Komponenten CPU, Hauptspeicher, Hintergrundspeicher und Netzanbindung, welche in allen Virtualisierern vorhanden sind gemessen.

Klassifizierung von Anwendungen

Über eine Matrix bestehend aus Anwendungen und von ihnen beanspruchte Komponenten, die jeder Administrator selbst auf Basis von Erfahrungswerten aufstellen kann, können Anwendungen kategorisiert werden. Für die in 3.1.1.2 genannte Liste an Anwendungen, könnte das exemplarisch wie in Tabelle 3.1 aussehen. Die Klassifizierung kann dabei entweder binär oder gewichtet erfolgen, je nach gewünschtem Detaillierungsgrad.

	CPU	RAM	Disk	Netz
DNS-Server	X			X
DHCP-Server				X
File-Server			X	X
Print-Server	X	X		X
Mail-Server		X	X	X
Web-Server				X
Datenbank-Server	X	X	X	
Terminal-Server	X	X		X
Streaming-Server (Multimedia)		X	X	X
Verzeichnisdienste	X	X	X	
Backup-Server			X	X
Monitoringdienste				X
Softwareverteilungsdienste				X
Firewall	X			X
Intrusion Detection Systeme	X			X

Tabelle 3.1: Klassifizierung von Anwendungstypen

Wirkungsgrad der Virtualisierung von Anwendungen

Verwendet eine Anwendung eine oder gar mehrere Komponenten, deren Wirkungsgrad bedingt durch die Virtualisierungsschicht eher schlecht ist, so sollte die entsprechende Maschine im produktiven Einsatz tendenziell nicht virtualisiert betrieben werden. Für den gewichteten Fall lassen sich Schwellwerte definieren, die durch die Summe aller Produkte aus Wirkungsgrad der Komponente und ihrem Gewichtungsfaktor für eine Anwendung nicht überschritten werden darf. Natürlich existieren Ausnahmen von dieser Regel, etwa wenn ältere Anwendungen nur noch aus Kompatibilitätsgründen betrieben und eher selten verwendet werden, so dass Performanz und Effizienz nur eine untergeordnete Rolle spielen. Durch die losgelöste Messung der genannten Einzelkomponenten lässt sich also zunächst ein komponentenspezifischer Wirkungsgrad einer Virtualisierungslösung ermitteln. Durch die Matrix-Abbildung auf Anwendungen lässt sich eine Aussage über einen anwendungsspezifischen Wirkungsgrad abschätzen. Da der Wirkungsgrad des Virtualisierers in der Praxis eventuell nicht erreicht wird, da andere Faktoren bei der parallelen Ausführung von virtuellen Maschinen einen Engpass bilden können, müssen im Anschluss daran Messungen durchgeführt werden, wie sich die einzelnen Wirkungsgrade gegenseitig beeinflussen. Hierzu misst man zu jeder der vier Komponenten jeweils einen der Wirkungsgrade der drei restlichen Komponenten parallel. Für den Fall von nebenläufigen virtuellen Maschinen sind auf die Kombinationen von identischen Komponenten möglich. Die durchzuführenden Messungen erläutert Tabelle 3.2.

Fällt bei einer dieser Kombinationen von Messungen ein Wirkungsgrad gegenüber der Einzelmessung stark ab, so sollten Anwendungen, die beide Komponenten

	CPU	RAM	Disk	Netz
CPU	(✓)	✘	✘	✘
RAM	✓	(✓)	✘	✘
Disk	✓	✓	(✓)	✘
Netz	✓	✓	✓	(✓)

Tabelle 3.2: Parallel zu messende Wirkungsgrade

benutzen, nicht virtualisiert werden, und Anwendungen, die jeweils eine der Komponenten verwenden, auf verschiedenen physischen Systemen ausgeführt werden, da die Ursache damit klar in der Hardware-Schicht zu suchen ist.

3.1.2 Leistungsbegriff im Kontext Virtualisierung

Grundsätzlich behält der klassische Leistungsbegriff im virtuellen Kontext seine volle Gültigkeit. Allerdings müssen ein paar zusätzliche Randbedingungen beachtet werden, um weiterhin sinnvolle Leistungsmessungen vornehmen zu können. Diese beruhen im Wesentlichen auf der Architektur der betrachteten Virtualisierungsansätze und werden nachstehend einzeln behandelt.

3.1.2.1 Unterschiede zum klassischen Fall

Zusätzliche Scheduler Instanz

Ein Unterschied im virtuellen Fall gegenüber dem klassischen ist die Existenz zweier, von einander unabhängig agierender Scheduler. Ein Scheduler arbeitet wie gewohnt im Betriebssystem, während in der Regel ein zweiter durch die Virtualisierungsschicht eingeführt wird und für das Scheduling zwischen den virtuellen Instanzen beziehungsweise dem VMM zuständig ist. Eine Ausnahme von diesem Konzept stellt die OS-Virtualisierung dar, da diese nur auf Kernel-Ebene ansetzt. Die beiden Scheduler arbeiten autonom und sind nicht synchronisiert. Im schlimmsten Fall kann es daher geschehen, dass ein Prozess in einer virtuellen Maschine dispatched wird, der Zeitslot der virtuellen Maschine aber gerade abgelaufen ist, so dass der Prozess einen vollen Rechenzyklus bezogen auf das Scheduling des VMM warten muss, bis er zur Abarbeitung kommt. Dadurch kann sich zum einen bei gleichem Durchsatz die Antwortzeit erhöhen, was das System für einen Anwender träge erscheinen lässt, als auch zu ernsthaften Problemen bei Echtzeit-Anwendungen führen, da diese ihre Timing-Vorgaben nicht mehr einhalten können. Die Existenz zweier nicht synchronisierter Scheduler ist in der Hauptsache ein Problem der Koordination, da die durchschnittliche effektive Rechenleistung der virtuellen Maschine identisch zu einem physischen System, das dieselbe Aufgabe erfüllt, sein mag. Lediglich die Verteilung der Leistung variiert und kann zu einer erhöhten Antwortzeit führen, wenn zum Zeitpunkt der

Ausführung mehrere virtuelle Maschinen aktiv sind. Dies ist ein generelles Problem der Host-Virtualisierung, sobald die virtuelle Maschine eine eigene Kernel Instanz erzeugt. Lösbar ist diese Problematik lediglich durch den Einsatz von Techniken der Paravirtualisierung, die einen Austausch von Informationen der beiden Scheduler erlauben.

Flaschenhälse in der Hardware

Ein weiterer Unterschied besteht darin, dass im virtuellen Fall meist mehrere virtuelle Maschinen auf demselben physischen System zur selben Zeit aktiv sind. Durch das Vorhandensein mehrerer Prozessoren und Rechenkerne können in der Tat mehrere Instanzen gleichzeitig Berechnungen ausführen, jedoch sind nicht alle Systemkomponenten mehrfach vorhanden und müssen daher zwischen den virtuellen Instanzen aufgeteilt werden. Insbesondere der Systembus ist nur einmal vorhanden und im Gegensatz zum klassischen Szenario nicht leistungsfähiger als der Bus eines einzelnen physischen Rechners. Der von Neumansche Flaschenhals der I/O-Anbindung wird durch Virtualisierung daher zunehmend zum Problem.

Virtualisierungsaufwand

Der wohl am meisten kritisierte Punkt an Virtualisierung ist der zusätzlich erforderliche Aufwand zur Erzeugung und Trennung virtueller Instanzen. Speicheradressen der virtuellen Maschinen müssen vom Hypervisor abgefangen und übersetzt werden, Prozessorinstruktionen auf ihre Unbedenklichkeit in virtuellen Maschinen überprüft und gegebenenfalls modifiziert werden und einzelne Hardware-Komponenten müssen gar komplett emuliert werden, um sie einer virtuellen Maschine überhaupt bereitstellen zu Können. All diese Operationen benötigen Rechen- und Speicherkapazität, die im klassischen Fall nicht notwendig wäre und die Leistung einer virtuellen Maschine im Vergleich zu einer physischen deutlich reduzieren können. Ein weiterer Einflussfaktor auf die Leistung einer virtuellen Maschine ist daher durch den Virtual Maschine Monitor gegeben. Da sich die Leistung des darunterliegenden physischen Systems jedoch nicht geändert hat, spricht man in diesem Zusammenhang nicht von Leistung, sondern vom Wirkungsgrad eines Virtualisierers. Welche einzelnen Faktoren tatsächlich Einfluss auf den Wirkungsgrad von Virtualisierung haben, wird im Abschnitt 3.2 genauer untersucht. Da die Auswirkung abhängig von der Art der Virtualisierung ist, wird die Analyse separat für die wichtigsten Virtualisierungsansätze durchgeführt.

3.1.2.2 Messbarkeit

Absolute vs. relative Metriken

Im Gegensatz zum Leistungsbegriff, der relativ unabhängig von Virtualisierung definiert werden kann, müssen bei der Leistungsmessung in virtuellen Systemen einige Randbedingungen beachtet werden, um die Messungen nicht zu verfälschen. Zum einen muss man sich entscheiden, ob die Metrik, die man zur Messung verwenden will, eine absolute Metrik wie zum Beispiel MIPS sein soll oder aber relativ, das heißt bezogen auf das physische Hostsystem. Absolute Metriken sind nur dann sinnvoll einsetzbar, wenn genau eine virtuelle Maschine auf dem System aktiv ist, denn zusätzliche virtuelle Instanzen würden das Ergebnis des Benchmarks abhängig von der zusätzlich induzierten Last eventuell stark beeinflussen.

Interessanter als absolute Metriken sind relative Metriken, die das Ergebnis eines Benchmarks, der sowohl auf einer virtuellen Maschine als auch auf dem darunter liegenden, physischen System durchgeführt wurde, in ein Verhältnis setzt. Dieses Vorgehen ist aufwendiger als absolutes Messen, da pro relativer Messung zwei absolute Messungen notwendig sind. Das Ergebnis einer relativen Messung ist für den Zweck dieser Arbeit aber deutlich besser geeignet, da sich aus den Verhältnissen der Mehraufwand für die Virtualisierung an sich als auch der Einfluss parallel laufender virtueller Maschinen ablesen lässt. Da durch relative Metriken ein Maß für die Effizienz von Virtualisierung und virtuellen Infrastrukturen existiert, können diese auch als Indikator verwendet werden, um die Effizienz, die in dieser Arbeit als Wirkungsgrad bezeichnet wird, zu erhöhen.

Wirkungsgrad

Der Wirkungsgrad eines Systems ist in der Physik eine skalare Größe, die die Effizienz eines Vorgangs darstellt. Berechnet wird sie aus dem Verhältnis geleisteter Arbeit zu zugeführter Energie. Bezogen auf die Effizienz von Virtualisierung bedeutet dies: Erbrachte Leistung einer virtuellen Maschine zur Leistung des darunterliegenden Systems bezogen auf dieselbe Arbeit.

$$Wirkungsgrad_A = \frac{Leistung_{A,VM}}{Leistung_{A,physischesSystem}}, \tag{3.3}$$

bezogen auf dieselbe Anwendung A.

Wirkungsgrad virtueller Maschinen

Während die Metriken zur klassischen und virtuellen Leistungsmessung absolute Metriken darstellen, ist der Wirkungsgrad der Virtualisierung eine relative Metrik. Normalerweise sind auf einem System viele virtuelle Instanzen aktiv. Jede von diesen basiert auf individuellen Konfigurationsparametern der Virtualisierungsschicht, enthält in der Regel unterschiedliche Software und verrichtet unterschiedliche Aufgaben. Damit kann jeder einzelnen Instanz ein Wirkungsgrad zugeordnet werden, nicht aber dem Gesamtsystem. Den Wirkungsgrad eines Systems erhält man erst durch gleichzeitiges Ausführen der Benchmarks. Dazu müssen alle Benchmarks auf dem physischen System gleichzeitig ausgeführt werden, um den absoluten Messwert bezogen auf das System zu ermitteln. In einem zweiten Schritt wiederholt man die Messungen, indem man parallel jeden der Benchmarks in einer eigenen virtuellen Maschine ausführt. Die Summe der gemessenen virtuellen Leistungen im Verhältnis zur Summe der physisch gemessenen Benchmark-Ergebnisse ergibt den Wirkungsgrad der Virtualisierung des Systems bezogen auf einen Benchmark.

$$Wirkungsgrad_{A_1..A_N} = \frac{\sum_{i=1}^{N} Leistung_{A_i,VM}}{\sum_{i=1}^{N} Leistung_{A_i,physischesSystem}}, \tag{3.4}$$

bezogen auf die parallel aktiven Anwendungen $A_1..A_N$.

Zeitmessung in virtuellen Maschinen ist ungenau

Grundsätzlich können alle Benchmarks, die in physischen Systemen ausführbar sind, auch in virtuellen Maschinen ausgeführt werden. Es muss lediglich beachtet werden, dass Zeitmessungen in virtuellen Maschinen verglichen mit physischen Systemen eventuell ungenauere oder semantisch unterschiedliche Werte liefern kann [Stil 07]. So enthalten beispielsweise die Zähler für Real-, User- und System-Zeit im Vergleich zum Unix-Standard unter Xen semantisch unterschiedliche Werte. Die User-Zeit wird zur relativen Laufzeit einer Domain, und die Real- und System-Zeit beinhalten die Wartezeiten aller aktiven Domains. Andere Ansätze zur Zeitmessung verwenden den Time-Stamp-Counter (TSC), über den jeder CPU Kern separat verfügt, zumindest bei neueren Prozessoren ab der Intel Pentium Reihe. Dieser wird allerdings nur alle vier, teilweise aber sogar nur alle 25 CPU-Zyklen aufgefrischt und von Xen zusätzlich bisweilen mit dem virtualisierten Interrupt-Controller synchronisiert, so dass auch dieser Wert starken Schwankungen unterliegt. Die einzig genaue Zeitbestimmung unter Xen basiert daher auf dem Ansatz, den Xen-Wallclock-Timer zu befragen. Die ist zurzeit allerdings nur mit Techniken der Paravirtualisierung möglich, da das virtualisierte System zum Lesen der Werte auf Shared Memory zugreifen muss anstatt auf die üblichen Zeitregister. Um dies zu erreichen, muss der Kernel verändert werden.

Messung über den Hypervisor

Ein weiterer Ansatz, die Leistung eines Systems zu messen, ist es, die Monitoring Schnittstelle des Hypervisors zu nutzen. Diese liefert meist sowohl einen komponentenbezogenen Anteil einer virtuellen Maschine in Bezug auf das physische System als auch absolute Werte bezogen auf eine Komponente. Im Falle einer CPU könnte der relative Werte zum Beispiel 30% lauten, und der absolute CPU Anteil fünf Sekunden betragen. Diese Werte sind zwar nicht mit Benchmark-Ergebnissen vergleichbar, können aber wohl einen Hinweis darauf geben, welches System die meisten Ressourcen benötigt und wo im Falle von Engpässen der Flaschenhals zu suchen ist. Beinahe alle gängigen Hypervisor bieten eine solche Monitoring-Schnittstelle, da die Daten zum Zwecke des Scheduling intern ohnehin benötigt werden. Viele der kommerziellen Lösungen bieten darüber hinaus auch noch Monitoring-Plattformen, die diese Information graphisch bereitstellen. Gängige Monitoring-Parameter des Monitoring virtueller Maschinen sind CPU, Disk-I/O und Netz-I/O.

3.2 Einflussfaktoren auf das Leistungsverhalten

Vor den eigentlichen Messungen der Leistungsfähigkeit von Virtualisierungslösungen müssen fundierte Überlegungen stehen, welche Parameter das Leistungsverhalten von virtuellen Infrastrukturen beeinflussen können. Wie bereits im Kapitel 2 dargelegt, basieren alle bestehenden Virtualisierungslösungen auf einer Architektur bestehend aus drei unterschiedlichen Komponenten.

- Hardware
- VMM / Hypervisor
- Gastbetriebssystem

Identifizierung von Einflussfaktoren

Will man die Menge potentieller Einflussfaktoren identifizieren, welche Virtualisierung in ihrer Leistungsfähigkeit beeinflussen, so muss man die Parameter untersuchen, welche die Leistungsfähigkeit der genannten Komponenten beeinflussen, und wie diese miteinander interagieren. Dieser Abschnitt beschäftigt sich daher mit der Identifizierung von Einflussfaktoren auf die Leistung virtueller Systeme, indem Hard- und Software-Komponenten sowie mögliche Konfigurationsparameter diesbezüglich analysiert werden. Diese Analyse muss für mehrere Applikationen und Applikations-Typen, welche die Art des Benchmarks verkörpern, erfolgen.

3.2.1 Hardware

Komplexes Zusammenspiel vieler Komponenten

Aktuelle Hardwarekomponenten haben einen wesentlichen Einfluss auf die Leistungsfähigkeit eines Systems. Um die Leistungsfähigkeit eines physischen Systems zu steigern, wurden von den Hardware-Herstellern bereits eine Vielzahl von Optimierungsansätzen integriert, die das Zusammenspiel der Einzelkomponenten zwar beschleunigen, aber die Komplexität drastisch erhöhen. Ein Beispiel hierfür ist unter anderem das Pipelining in Prozessoren und die bereits erwähnte Technik des Prefetching bei DDR-Speichern. Bei derart komplexen Architekturen kann daher beinahe jede kleine unbedachte Änderung an einer Komponente die Performanz des gesamten Systems wesentlich beeinflussen. Die Einführung von Virtualisierung schafft einen zweiten Scheduler, der beim Dispatchen ein nicht eingeplantes Löschen der Caches notwendig macht, was negative Folgen auf die virtuelle Maschine haben wird, da alle Daten erneut aus dem langsamen Hauptspeicher gelesen werden müssen. Aus diesem Grunde wird dieser Abschnitt alle für Virtualisierung essentiell notwendigen Hardware-Komponenten identifizieren und deren Einfluss auf das Leistungsverhalten theoretisch analysieren und bewerten.

Existierende Architekturen

Im Wesentlichen existieren zwei unterschiedliche Architekturen für x86-basierte Server: Eine davon ist die Direct Connect Architektur von AMD, bei der der Memory Controller in den CPUs integriert ist und jede CPU über direkt angebundenen Speicher verfügt, der jedoch für alle anderen Prozessoren zugreifbar ist, wenn auch

mit leichter Verzögerung, da dies nicht direkt möglich ist, sondern nur über die extra hierfür implementierte Hypertransport-Schnittstelle. Im Gegensatz hierzu läuft alle Kommunikation bei der Intel-Architektur über den etwas langsamen Front Side Bus, so dass die Kommunikation der einzelnen Kerne untereinander sowie die Anbindung von Memory Controller und Hauptspeicher deutlich langsamer ist und zudem um denselben Bus konkurriert. Neuere Varianten der Intel-Architektur mit dem Feature Intel Virtualization Technology for Directed I/O (VT-d), basieren daher auf einer zu AMD ähnlichen Variante. Einen groben Vergleich der beiden Architekturen gibt Abbildung 3.1.

Anbindung von Caches

Ein weiterer Unterschied ist die Anbindung der Caches an einzelne Kerne des Prozessors. Während AMD bei seinen Quad-Cores exklusive Level 1 und 2 Caches pro Kern und gemeinsame Level 3 Caches implementiert, haben Intels Quad-Cores nur einen dedizierten Level 1 Cache, den Level 2 Cache teilen sich bereits zwei Kerne. Gleichzeitig aktive virtuelle Maschinen können sich daher bei Intel-Prozessoren gegenseitig den Inhalt der Caches leichter überschreiben, was bei erneutem Zugriff ein Neuladen der Daten aus dem langsameren Hauptspeicher erzwingt.

Unterschiede im Kommunikationsmodell

Sowohl Intel als auch AMD verwenden trotz unterschiedlicher Architektur im Wesentlichen dieselben Einzelkomponenten, so dass Änderungen an einzelnen Hardwarekomponenten im Schnitt ähnliche Ergebnisse bei Benchmarks hervorbringen werden. Da sich allerdings das Kommunikationsmodell der beiden Architekturen erheblich unterscheidet, wird ein einzelnes Ergebnis bei der Direct Connect Architektur erheblich davon abhängen, an welchen Sockel ein Prozess gebunden ist. Im Gegensatz dazu sind Prozesse bei der Front Side Bus Architektur in höherem Maße von nebenläufigen Prozessen abhängig, da sie auf demselben Bus operieren, der sich als Engpass herausstellen könnte.

In Summe ergeben sich daher für die Virtualisierung folgende Möglichkeiten, basierend auf Hardware leistungsrelevante Änderungen vorzunehmen:

- Systemarchitektur
- Prozessor
  - Taktfrequenz
  - Anzahl Kerne
  - Anzahl Sockel
  - Virtualisierungsunterstützung
- RAM
  - Größe
  - Anzahl Module
  - Taktfrequenz

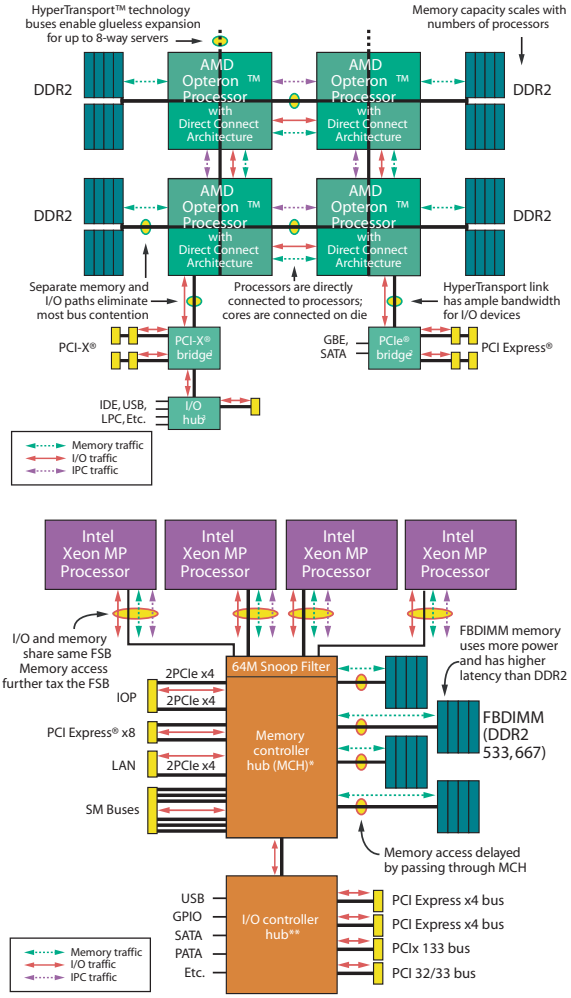


Abbildung 3.1: Gegenüberstellung der Systemarchitekturen von AMD und Intel basierten Servern [AMD 08].

- PCI-Geräte
  - Netzwerkkarten
  - Host Based Adapter (HBA)
- Hintergrundspeicher

3.2.2 Software

In die Kategorie der Software-Komponenten fallen sowohl VMM- und Hypervisor-Techniken als auch Faktoren, die von den eingesetzten Gastbetriebssystemen abhängig sind. Diese werden im Folgenden detailliert erläutert.

3.2.2.1 VMM / Hypervisor

Aufgaben des Hypervisors

Der Virtual Machine Monitor spielt eine zentrale Rolle bei der Virtualisierung. Je nach eingesetzter Virtualisierungstechnik können bestimmte Teilaufgaben effizienter als andere durchgeführt werden. Aus diesem Grunde bedienen sich beinahe alle implementierten Hypervisor eines Mischkonzeptes an Techniken, um die maximal mögliche Performanz zu erreichen. Aus diesem Grund werden hier nicht vollständige Hypervisor-Implementierungen analysiert, sondern die von ihnen eingesetzten Techniken bezogen auf eine Teilaufgabe. Die wichtigsten zu behandelnden Teilaspekte sind:

- Scheduling
- Adressübersetzung
- Privilegierte Instruktionen
- Virtuelle Netzkomponenten
- Hintergrundspeicher

Bekannte Virtualisierungstechniken, die diese Punkte realisieren, sind die in Kapitel 2 bereits vorgestellten Konzepte:

- Vollvirtualisierung
- Native Virtualisierung
- Paravirtualisierung
- OS-Virtualisierung
- Emulation

Jede der oben stehenden Virtualisierungsarten macht von fremden Techniken Gebrauch, um die erwähnten Teilaufgaben zu realisieren. Virtualisierer, die strikt einen Ansatz implementieren existieren in der Praxis nicht. Einen ersten Überblick gibt dazu Tabelle 3.3. Nachfolgend wird die Leistungsfähigkeit der einzelnen Techniken detailliert erläutert.

Schedul-ingkonzepte

3.2.2.1.1 Scheduling Das Schedulingkonzept von Vollvirtualisierung, Paravirtualisierung und Emulation [Math 07, Doro 07], [Wiki 08, Stichwort: OpenVZ] basiert auf der Zuordnung von virtuellen CPUs der Gastbetriebssysteme auf physische CPUs oder Kerne. Sobald eine virtuelle CPU auf eine physische Recheneinheit geschalten ist, wird direkt auf dieser gerechnet, ohne weitere Übersetzungsschritte. Das dazu notwendige Schedulingverfahren wird daher bei allen drei genannten Virtualisierungsarten in zwei Stufen ausgeführt. Als erstes wird eine virtuelle Instanz ausgewählt, die rechnen darf. Dies erledigt der Scheduler des Virtual Maschine Monitor. Im zweiten Schritt entscheidet ein Scheduler in der virtuellen Maschine, welcher Prozess rechnen darf. Diese beiden Schritte sind nicht mit einander synchronisiert, so dass sie meist nicht direkt aufeinander folgen. Das Schedulingkonzept der OS-Virtualisierung arbeitet ebenfalls in zwei Schritten. Zunächst wird ein Container ausgewählt, anschließend ein Prozess aus dem gewählten Container. Der Unterschied zu den anderen Virtualisierungsarten ist lediglich, dass keine virtuellen CPUs benötigt werden, da direkt Prozesse dispatched werden können. Die zugrunde liegende Architektur ist also bei allen Virtualisierern die identische, lediglich die Art des Schedulers im Hypervisor variiert von Implementierung zu Implementierung. Zum Teil sind die Scheduler-Implementierungen sogar zur Laufzeit dynamisch änderbar. Insbesondere für Xen wurden vier verschiedene Scheduler implementiert, die nach unterschiedlichen Kriterien optimiert wurden. Verfügbare Scheduler für Xen sind unter anderem ein Echtzeit-Scheduler sowie Credit-basierte Scheduler.

Scheduler auf unterschiedlichen Hosts sind nicht synchronisiert

Ein weiteres Problem, das durch das Scheduling von virtuellen Maschinen entstehen kann, ist, dass zwei virtuelle Maschinen, die häufig interagieren wie zum Beispiel ein Web- und ein Datenbankserver, zeitlich versetzt ausgeführt werden und ständig aufeinander warten müssen, was zu deutlichen Leistungseinbußen führen kann. Ein synchrones Scheduling ausgewählter Maschinen wäre daher sinnvoll, wird aber zurzeit noch von keinem Hersteller unterstützt.

Shodow Page Tables

3.2.2.1.2 Adressübersetzung Das Konzept der Shadow Page Tables sieht vor, das Gastbetriebssystem seine Page Tables selbst verwalten zu lassen. Dies funktioniert solange performant, bis ein Page Fault auftritt und ein schreibender Zugriff auf die lokalen Paging Tabellen des Gastes nötig wird. Dieser kann nur vom Virtual Machine Monitor ausgeführt werden, da zeitgleich die globalen Paging Tabellen aktualisiert werden müssen, die auch die MMU verwendet. Für diese Aufgabe muss ein teurer Kontext-Switch zwischen Gastbetriebssystem und Virtual Maschine Monitor und zurück vorgenommen werden. Bei der Verwendung von

	Voll-Virt.	**Native-Virt.**	**Para-Virt.**	**OS-Virt.**	**Emulation**
Scheduling	asynchron	asynchron	asynchron	asynchron	asynchron
Adressübersetzung	Shadow Page Tables	Shadow Page Tables	Hypercall	Separierung	VMM
Privilegierte Instr.	Binärübersetzung	Binärübersetzung	Hypercall	-	Binärübersetzung
Virt. Netzkomponenten	Emulation	Pass Through	Pass Through	Pass Through	Emulation
Hintergrundspeicher	Emulation	Pass Through	Pass Through	Pass Through	Emulation

Tabelle 3.3: Aufstellung eingesetzter Techniken zur Realisierung von zur Virtualisierung notwendigen Teilaufgaben.

Shadow Page Tables ist der Fall, dass sich eine Seite bereits im Speicher befindet, optimal gelöst, allerdings auf Kosten des Falles, dass beim Zugriff auf eine Seite ein Page Fault entsteht und langwierig abgefangen und emuliert werden muss. Das Verfahren dürfte aus diesem Grund genau dann zu Problemen führen, wenn in der Praxis viele Page Faults zu erwarten sind. Dies ist unter anderem bei Datenbank-Systemen der Fall.

Paravirtualisierung

Effizienter arbeitet in diesem Fall die Paravirtualisierung, da die Übersetzung von Adressen in jedem Fall vollständig vom Hypervisor selbst vorgenommen wird. Eine Übersetzung von virtuellen Adressen in physische Adressen der virtuellen Maschine durch das Gastbetriebssystem ist nicht notwendig. Der dafür notwendige Kontext-Switch ist verhältnismäßig billig, da der Adressraum des Hypervisors am oberen Ende in jedem Adressraum einer virtuellen Maschine eingeblendet wird. Das Verfahren ist insofern effizienter als das Verwalten von Shadow Page Tables, als dass zwei Übersetzungsschritte zu einem zusammengefasst werden können und unabhängig von auftretenden Page Faults konstante Leistung erbringt. Die notwendigen Kontextwechsel dürften allerdings dazu führen, dass die Übersetzung bei geringer Anzahl von Page Faults geringfügig langsamer ist als bei der Verwendung von Shadow Page Tables.

Isolation von Adressräumen

Die Adress-Übersetzung innerhalb von OS-Virtualisierung unterscheidet sich nur unwesentlich von der in herkömmlichen Betriebssystemen, da im Wesentlichen nur ein Betriebssystemkern ausgeführt wird und die zu verschiedenen Containern gehörenden Prozesse lediglich etwas stärker von einander isoliert werden. OS-Virtualisierung erzeugt mit dieser Technik generell nur einen sehr kleinen Mehraufwand. Die Adress-Übersetzung arbeitet daher mehr oder weniger in ihrer gewohnten Performanz.

Emulation

Die schlechtesten Ergebnisse bezogen auf die Adress-Übersetzung sind von reinen Emulatoren zu erwarten, da sie aufgrund der unterschiedlichen Architekturen von Gast und physischem System zur schrittweisen Übersetzung der Adressen gezwungen werden. Jede Übersetzung einer Adresse ist daher zwangsweise mit einem Kontextwechsel verbunden und kostet wertvolle Zeit. Dieses einfache, aber ineffektive Verfahren kommt bei Produkten zur Servervirtualisierung daher nicht zum Einsatz.

Emulation vs. Paravirtualisierung

3.2.2.1.3 Privilegierte Instruktionen Der technisch größte Unterschied der verschiedenen Virtualisierungsansätze, der zugleich auch die größte Herausforderung darstellt, ist das Abfangen von privilegierten Prozessorinstruktionen. Im Wesentlichen existieren hierfür zwei grundlegend verschiedene Ansätze: Emulation verbotener Instruktionen durch eine Folge unkritischer Instruktionen oder Vermeidung verbotener Instruktionen durch Anpassung des Kernels im Gastbetriebssystem an die modifizierte ABI durch die Einführung von Hypercalls. Da bei der Modifikation der ABI, welche bei der Paravirtualisierung eingesetzt wird, zur Laufzeit nahezu kein zusätzlicher Aufwand entsteht, wenn man von

verhältnismäßig billigen Kontext-Switchen zwischen Gast und Hypervisor sowie zurück absieht, handelt es sich in Summe um eine sehr performante Technik.

Binärübersetzung

Im Gegensatz hierzu entsteht bei der Binärübersetzung, wie sie bei der Vollvirtualisierung zum Einsatz kommt, vor der Ausführung eines Code-Blocks zunächst ein Übersetzungsaufwand. Da die Übersetzung eines ausgeführten Programmteils während einer Prozessdauer gespeichert wird, fällt anschließend kein Mehraufwand mehr an. Lediglich die Befehlsfolge, die eine Instruktion emuliert, sollte geringfügig langsamer ausgeführt werden, da sie in der Regel in mehreren Prozessorzyklen abgearbeitet werden muss. Initial wird diese Technik daher definitiv langsamer sein als die paravirtuelle Variante der Hypercalls, bereits übersetzte Code Blöcke werden dagegen annähernd dieselbe Ausführgeschwindigkeit erreichen.

Emulation

Obwohl auch reine Emulatoren Binärübersetzung anwenden, müssen diese wie auch schon bei der Adressübersetzung Befehl für Befehl von einer Architektur in eine andere übersetzen. Selbst bei einer Speicherung des übersetzten Codes ist der initiale Aufwand so groß, dass diese Technik aus Performanz-Gründen eigentlich nur zu Entwicklungszwecken, nicht aber für den Serverbetrieb eingesetzt werden kann.

Container

Im Gegensatz zu den bereits erwähnten Virtualisierungsansätzen benötigt die OS-Virtualisierung kein Abfangen privilegierter Instruktionen, da nur eine einzige Kernel-Instanz am aktiv ist. Diese besitzt die nötigen Berechtigungen, um alle Instruktionen ausführen zu können. Der Kernel benötigt daher lediglich zusätzliche Mechanismen, die ein Zugreifen eines Containers auf nicht zu ihm gehörende Bereiche des Dateisystems unterbinden. Dies lässt sich verhältnismäßig einfach realisieren, unter Linux eignet sich hierzu etwa das Programm `chroot`.

Koppelkomponenten vs. Netzwerkkarten

3.2.2.1.4 Virtuelle Netzkomponenten Bei der Bereitstellung von virtuellen Netzkomponenten muss man grundsätzlich zwei unterschiedliche Arten an virtueller Hardware differenzieren. Zum einen Netzkoppelkomponenten, die in der Regel durch Bridges realisiert werden, und Netzwerkkarten für die virtuelle Maschine. Bei der OS-Virtualisierung und Paravirtualisierung kommen angepasste Netzwerkkarten-Treiber zum Einsatz. Diese sind dahin optimiert, dass sie keinen vollwertigen Treiber, wie zum Beispiel bei der Vollvirtualisierung oder Emulation, darstellen und auf emulierte Hardware angewiesen sind, sondern nur eine einfache Schnittstelle im Gastbetriebssystem bereitstellen, über die Pakete für das Netz entgegengenommen werden. Diese Pakete werden direkt an die entsprechende virtuelle Bridge übergeben. Gleiches gilt für den Rückweg. Nur für den Fall, dass das Paket das virtuelle Netz verlässt, wird ein vollwertiger Treiber benötigt. Dieser kann je nach Architektur in einer privilegierten virtuellen Maschine, wie etwa in Xen oder Hyper-V, oder aber im Hypervisor, der dadurch zum Microkernel wird, integriert werden. Ein Beispiel für einen Microkernel ist VMwares ESX Server. Um auch bei der Vollvirtualisierung eine höhere Leistung der Netzan-

bindung zu erreichen, entstand das Konzept optimierte Netzwerkkartentreiber für die Vollvirtualisierung zu entwickeln. Da dies de facto aber einer Anpassung des Betriebssystems gleicht und damit Konzepte der Paravirtualisierung verwendet, trägt dieser hybride Virtualisierungsansatz den Namen Native Virtualisierung. Da in der Praxis die reine Vollvirtualisierung und Emulation keine Rolle spielen, ist die geringe Leistung der virtuellen Netzanbindungen bei diesen Techniken kaum relevant. Eine Ausnahme stellt Xen in der vollvirtualisierten Variante mit Hardware-Unterstützung dar. In der aktuellen Version 3 werden Netzwerkkarten vollständig emuliert, was zu einem größeren Leistungsverlust führt.

Blockbasiert vs. dateibasiert

3.2.2.1.5 Hintergrundspeicher Ähnlich zur Bereitstellung virtueller Netzkomponenten verhält es sich bei der Bereitstellung von Festspeicher. Hier existieren ebenfalls für den Einsatz in virtuellen Maschinen optimierte Treiber. Interessanter ist in diesem Zusammenhang vielmehr, welche Art der Speicheranbindung gewählt wird, da Dateien der virtuellen Maschinen sowohl datei- als auch blockbasiert über diverse Medien angebunden sein können. Typische Realisierungen sind etwa lokaler Speicher auf SCSI-Festplatten oder entfernter Speicher angebunden über Fibre Channel oder Ethernet. Da lokaler Speicher meist mangels Unterstützung von Live-Migration ausscheidet, hat man die Wahl aus einer Kombination von Fibre Channel oder Ethernet und datei- oder blockbasiert. Da die nominelle Datentransferrate von Fibre Channel höher liegt als bei entsprechenden Ethernet Varianten, wird auch der Durchsatz entsprechend höher liegen, sofern nicht die angeschlossenen Systeme den Flaschenhals bilden.

3.2.2.2 Betriebssysteme

Speicherbedarf moderner Betriebssysteme

Während der Ressourcenbedarf und Umfang von Betriebssystemen auf physischen Servern nahezu keine Rolle spielt, da diese in der Regel überproportional mit Ressourcen ausgestattet sind, spielt dieser Aspekt bei der Virtualisierung plötzlich wieder eine größere Rolle, da die Ressourcen eines Systems, das zur Virtualisierung eingesetzt wird, beschränkt sind. Insbesondere der Bedarf an Hauptspeicher, der bei einigen Betriebssystemen in den letzten Jahren stark gewachsen ist, könnte zur Kostenfalle bei der Virtualisierung werden, da server-zertifizierte Speicherbausteine mit großen Kapazitäten nach wie vor sehr teuer sind. Die Wahl sollte daher auf schlanke Betriebssysteme fallen, die nur die benötigte Software enthalten. Hier eignen sich insbesondere modular aufgebaute Systeme wie zum Beispiel einige Linux-Varianten. Auch Microsoft geht hier einen Schritt in die richtige Richtung, indem es der Server 2008 erlaubt, lediglich eine „Core-Installation" ohne grafische Oberfläche zu installieren und diese um weitere Dienste zu erweitern.

Treiber für virtuelle Hardware

Der Einsatz von für Virtualisierung optimierten Treibern innerhalb der Gastbetriebssysteme ist ein weiterer zu beachtender Aspekt, da diese den Durchsatz auf virtuellen Geräten wie Netzwerkkarten und Festplatten erheblich steigern können und oft auch weitere Funktionalitäten bereitstellen. Eine dieser zusätzlichen

Möglichkeiten, die von beinahe allen Para- und Native-Virtualisierern unterstützt wird, ist das dynamische Vergrößern und Verkleinern des Arbeitsspeichers innerhalb der virtuellen Maschinen, so dass virtuellen Maschinen mit kurzfristig erhöhtem Speicherbedarf dynamisch Speicher zugewiesen und bei sinkendem Bedarf auch wieder entfernt werden kann. Darüber hinaus erlauben diese Treiber eine Kommunikation des Hypervisors in das Gastbetriebssystem, wodurch sich zum Beispiel Kopier- und Einfügefunktionen zwischen verschiedenen virtuellen Maschinen realisieren lassen. Dieses Interface ließe sich eventuell erweitern, um zum Beispiel die Scheduler zwischen Hypervisor und Scheduler zu synchronisieren.

3.2.3 Konfiguration

Virtuelle Hardware

Die wesentlichen, konfigurierbaren Parameter einer virtuellen Maschine sind Hauptspeicher und Prozessor. Während die Größe des benötigten Hauptspeichers hauptsächlich von der auszuführenden Anwendung abhängig ist, kann bei der Zuteilung der Prozessoren sowohl die Anzahl der virtuellen CPUs als auch deren Taktfrequenz festgelegt werden. Die zugrunde liegende Rechenleistung ergibt sich daher aus dem Produkt der beiden Werte. Allerdings dürfte der Wirkungsgrad bei einer geringeren Anzahl an virtuellen Prozessoren, aber bei gleicher nomineller Leistung, höher sein, weil in diesem Fall die Zyklen im Scheduler des Hypervisors länger werden können, da weniger Bewerber für eine physische Recheneinheit konkurrieren. Damit reduziert sich der Overhead des Scheduling im Hypervisor.

Binden virtueller Instanzen an physische Ressourcen

Eine weitere Konfigurationsmöglichkeit, die die virtuellen Prozessoren betrifft, ist es, diese fest an dedizierte Prozessoren oder Kerne zu binden. Je nach eingesetzter physischer Architektur haben unterschiedliche Prozessoren dedizierte Teile des Arbeitsspeichers zugewiesen, für die sie eine performante Anbindung besitzen. Zugriffe auf andere Speicherbereiche müssen über einen oder mehrere separierte Speichercontroller in anderen Sockeln abgewickelt werden und sind daher weniger performant. Daher kann es sinnvoller sein, von der vorherrschenden Greedy-Strategie bei der Ressourcenzuteilung abzuweichen und eine virtuelle Maschine an eine höher ausgelastete CPU zu binden, die dafür aber die bessere Speicheranbindung besitzt.

Auch die Wahl eines für den Einsatzzweck passenden Scheduling-Algorithmus sollte geachtet werden. Die entsprechenden Möglichkeiten wurden bereits in Abschnitt 3.2.2.1.1 erläutert. Ebenso erläutert wurden die verschiedenen Varianten der Festspeicher Anbindung, siehe hierzu Abschnitt 3.2.2.1.5.

Exponentiell wachsende Konfigurationsmöglichkeiten

Während die Konfigurationsmöglichkeiten innerhalb eines Virtualisierers noch überschaubar sind, sind sie in virtuellen Infrastrukturen wesentlich größer, da die Anzahl der möglichen Verteilungen von m virtuellen Maschinen auf n Hosts exponentiell von der Anzahl der virtuellen Maschinen abhängt. Somit ergeben sich n^m mögliche Verteilungen von virtuellen Maschinen auf Hosts, die mit

allen lokalen Konfigurationsparametern kombiniert werden können. Letztendlich muss global eine dieser Verteilungen gefunden werden, bei der die physischen Ressourcen möglichst optimal ausgelastet werden können.

3.3 Kategorisierung von zu analysierenden Einflussfaktoren

In den vorstehenden Abschnitten wurde bereits erläutert, welche grundsätzlichen Kriterien Einfluss auf den Wirkungsgrad von Virtualisierung haben können. Die Abbildung 3.2 fasst diese noch einmal graphisch zusammen. Zudem kann bereits eine grobe Einteilung der Einflussfaktoren in statisch gesetzte und dynamisch änderbare Einflussfaktoren vorgenommen werden.

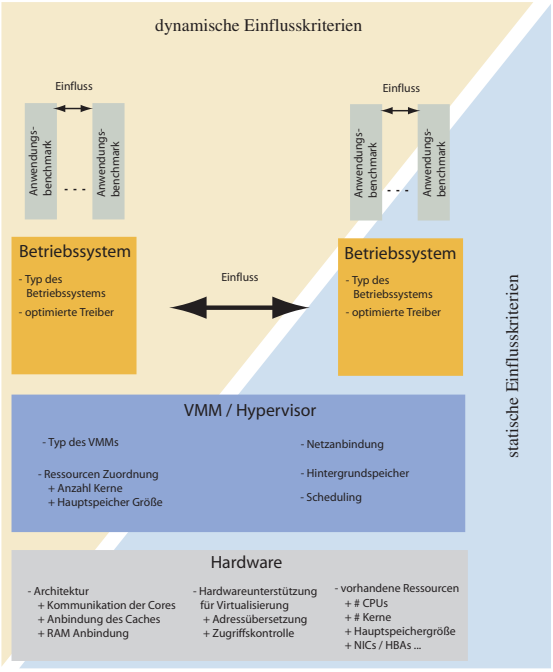


Abbildung 3.2: Einflussfaktoren auf das Leistungsverhalten von Virtualisierung

3.3.1 Statische Einflussfaktoren

Zu den statisch gesetzten Einflussfaktoren gehören alle Einflussfaktoren, die zum Zeitpunkt der Instanziierung der virtuellen Infrastruktur festgelegt wurden und zur Laufzeit nicht oder nur schwer änderbar sind. Typischer Weise gehören hierzu die hierarchisch tiefer liegenden Schichten des Virtualisierungsmodells. Statische Einflussfaktoren auf diesen Ebenen sind daher unter anderem:

- vorhandene physische Ressourcen
  - Anzahl CPUs
  - Anzahl Kerne
  - Anzahl und Größe der Caches
  - Hauptspeichergröße
  - NICs / HBAs
  - ...
- Netzanbindung des physischen Systems
  - Technik
  - Protokolle
- Hintergrundspeicher des physischen Systems
  - Block-/Dateibasiert
  - Dateisysteme
- Virtualisierungsunterstützung in Hardware für
  - Kontextwechsel
  - Adressübersetzung
- Architektur des physischen Systems
  - Bus Architektur, wie z.B. Intel Front Side Bus Architektur
  - Vermaschte Architektur, wie z.B. AMD Direct Connect Architektur
- Art des Virtualisierungsansatzes für eine Komponente
  - Vollvirtualisierung
  - Paravirtualisierung
  - Emulation
  - OS-Virtualisierung

- Scheduling Algorithmen
  - interaktiver Scheduler
  - Batch Scheduler
  - Echtzeit Scheduler
- Hintergrundspeicher der virtuellen Maschine
  - Images mit fester Größe
  - Images mit wachsender Größe
- Typ der Gastbetriebssysteme
  - Linux
  - Windows
  - Solaris
  - FreeBSD
  - ...
- Applikationen

3.3.2 Dynamische Einflussfaktoren

Im Gegensatz zu den statischen Einflussfaktoren sind dynamische Einflussfaktoren Einflussgrößen, die zur Laufzeit eines Systems automatisiert beeinflusst werden können. Somit sind sie Kandidaten, die von einem Resource Scheduling System nach einem vorliegenden Regelwerk verwaltet werden könnten. Zu ihnen gehören unter anderem:

- Zuordnung von virtuellen auf physische Ressourcen
  - Anzahl der virtuellen CPUs bei gleichbleibender Anzahl CPU-Zyklen
  - statische / dynamische Zuordnung von vCPUs zu physischen Kernen
- Anzahl parallel laufender virtueller Instanzen pro physischer Maschine
- Art der nebenläufigen virtuellen Maschinen
- Last der einzelnen virtuellen Maschinen
  - CPU Last
  - Disk I/O Last
  - Netz I/O Last
  - Hauptspeicher

- Kommunikationswege virtueller Maschinen
  - virtuell zu virtuell (gleiches physisches System)
  - virtuell zu virtuell (unterschiedliche physische Systeme)
  - virtuell zu physisch

3.3.3 Auswahl von zu analysierenden Einflussfaktoren

Gruppenbildung von assoziierten Parametern

Prinzipiell können alle in den vorangehenden Kapiteln genannten Faktoren Einfluss auf die Leistungsfähigkeit virtueller Infrastrukturen haben, wobei die Liste der aufgelisteten Parameter keinen Anspruch auf Vollständigkeit hat. Einige der Parameter werden die Leistungsfähigkeit unter Umständen nur im Zusammenspiel mit weiteren Faktoren beeinflussen. Letztendlich müssten daher sämtliche Permutationen aller Einflussfaktoren gemessen werden, um empirisch mit Sicherheit sagen zu können, welche Auswirkungen welche Parameter haben. Da dies mehrere tausend Messungen implizieren würde und schon deshalb nicht praktikabel ist, da noch nicht einmal eine zu 100% vollständige Liste an Einflussgrößen bekannt ist, muss man versuchen diese Anzahl auf ein durchführbares Maß zu beschränken. Ein möglicher Ansatz dies zu erreichen ist es, auf Basis der theoretischen Analyse der einzelnen Virtualisierungsansätze und deren bekannter Funktionsweise logische Gruppen zu analysierender Einflussfaktoren zu bilden. Durch diese Gruppenbildung verringert sich die Komplexität der Messungen erheblich. Weiterhin kann die Komplexität dadurch reduziert werden, dass Parameter, die nicht primär virtualisierungsspezifisch sind, wie etwa das Dateisystem des zugrunde liegenden Hintergrundspeichers, separat gemessen werden und nicht in Kombination mit allen anderen Parametern. Anstatt Anwendungsbenchmarks auszuführen und zu messen, welche sich nicht einfach auf reale Anwendungen übertragen lassen, da selten exakt dieselbe Anwendung im Benchmark und im produktiven Betrieb eingesetzt werden sowie zusätzlich in einem Anwendungsbenchmark nur das durchschnittliche Nutzverhalten simuliert werden kann, bietet es sich an die Performanz einzelner Hardware-Komponenten einer virtuellen Maschine zu messen. Ein Rückschluss auf die Performanz von Anwendungen ist mit Hilfe des Verfahrens in Abschnitt 3.1.1.2 möglich. Zwingend notwendige Komponenten einer virtuellen Maschine sind CPU, RAM und Hintergrundspeicher, welcher bei allen existierenden Hypervisorn aus einer Kombination aus Speicher-Controller und Speichermedium - typischerweise eine Festplatte - besteht. Hinzu kommt eine Netzkarte, ohne die ein Server in der Regel nutzlos ist. Die im PC-Bereich oft gerne gemessene Grafikkarte ist bei typischen Serveranwendungen irrelevant. Hinzu kommt, dass die existierenden Virtualisierungslösungen keine High-End Grafikkarten emulieren, da eine Grafikkarte in der Regel nur notwendig ist, um die virtuelle Maschine zu konfigurieren und administrieren. Dies ändert sich jedoch schlagartig, wenn CAD-Programme oder 3D-Spiele auf virtuellen Maschinen ausgeführt werden sollen. Hierzu bieten serverbasierte Virtualisierungslösungen

Messung des Wirkungsgrades einzelner virtueller Komponenten

derzeit noch keine Lösungen an, jedoch könnte man mit der beschriebenen Systematik auch diese Komponente messen.

Systematische Messreihen

Eine in sich abgeschlossene Menge an Messungen bezogen auf den Wirkungsgrad von virtualisierten Komponenten erhält man, wenn man zu jeder der vier zu messenden Komponenten (CPU, Hauptspeicher, Hintergrundspeicher und Netzanbindung) basierend auf einer Standardkonfiguration direkt assoziierte Parameter variiert und eventuell deren Wertebereich auf ein überschaubares Maß reduziert. Die hierfür nötigen Messungen müssen für jede der möglichen Virtualisierungstechniken sowie einem physischen Referenzsystem durchgeführt werden, dass ein Wirkungsgrad ermittelt werden kann. Um die Einflüsse paralleler Instanzen von virtuellen Maschinen auf die Effizienz von Virtualisierung zu analysieren müssen Tests analog zu Tabelle 3.2 mit wachsender Anzahl Instanzen durchgeführt werden. Die Konfiguration der virtuellen Maschinen sowie der Hardware und des Virtualisierers werden hierfür konstant gehalten, damit die Anzahl der Messungen überschaubar bleiben kann.

3.3.4 Durchzuführende Messungen

Berücksichtigte Parameter

Basierend auf den Vorschlägen in Abschnitt 3.3.3 wird in diesem Teilkapitel eine Auswahl der durchzuführenden Messungen getroffen. Die Auswahl der zu analysierenden Parameter ist spezifisch für jede zu untersuchende Komponente, allerdings existieren einige Faktoren, die so eng mit der Thematik Virtualisierung verknüpft sind, dass ihr Einfluss auf den Wirkungsgrad aller Komponenten interessant erscheint. Hierzu gehören:

- Virtualisierungstechnik
- Virtualisierungsunterstützung in Hardware
- Paravirtuelle Treiber
- Gastbetriebssystem

Je nach zu analysierender Komponente existieren weitere Einflussfaktoren, die zusätzlich zu den allgemeinen Faktoren gemessen werden müssen.

Nicht berücksichtigte Parameter

Parameter, die sich auf die Test-Hardware beziehen, werden im Folgenden nicht berücksichtigt, da dies immense Kosten für zusätzliche Testsysteme verursachen würde. Der Einfluss von Parametern, wie etwa Architektur des physischen Systembusses und Cache-Größen kann daher in dieser Arbeit nicht quantitativ bestimmt werden, ihr Einfluss wurden aber bereits qualitativ erläutert, so dass dies die Qualität der Messreihe nicht entscheidend beeinflussen wird. Weiterhin wird darauf verzichtet, Festplattenabbilder mit wachsender Größe zu untersuchen, da dies nur von wenigen Virtualisierungslösungen unterstützt wird und eine Vergleichbarkeit der Messungen daher nicht gewährleistet ist. Ähnlich verhält es sich

mit austauschbaren Schedulern. Diese Funktionalität wird bisher nur von Xen unterstützt, aber nicht produktiv eingesetzt, da die existierenden Scheduler mehr den Charakter einer prototypischen Implementierung zu Demonstrationszwecken besitzen.

3.3.4.1 CPU

Messbare Parameter virtueller Prozessoren

Direkten Einfluss auf den Wirkungsgrad der Virtualisierung von Prozessoren hat neben denn allgemeinen Parametern auch die Anzahl der virtuellen Prozessoren in einer virtuellen Maschine, da dies selbst bei identischer nomineller Rechenleistung Auswirkung auf das Scheduling des Virtualisierers besitzt. Interessante Parameter für die Komponente CPU sind daher:

- Virtualisierungstechnik
- Virtualisierungsunterstützung in Hardware
- Paravirtuelle Treiber
- Gastbetriebssystem
- Anzahl vCPUs

3.3.4.2 RAM

Zu messende Parameter beim Zugriff auf Arbeitsspeicher

Kein existierender Virtualisierer unterstützt zum heutigen Zeitpunkt Änderungen an der Konfiguration des Arbeitsspeichers einer virtuellen Maschine, abgesehen von dessen Größe. Ein interessanter Aspekt bei der Virtualisierung von Arbeitsspeicher ist jedoch die Technik der Adressübersetzung. Diese ist abgesehen von den allgemeinen Parametern auch von der Art des Zugriffes auf eine Speicherseite abhängig. So spielt es eine Rolle, ob der Zugriff lesend oder schreibend ausgeführt wird und ob Speicher sequenziell oder zufällig angefordert wird. Weiterhin können Kontextwechsel den Inhalt des Hardware TLBs invalidieren, so dass auch Scheduling-Aspekte untersucht werden müssen. Dies ist jedoch ein Aspekt, der nicht im Rahmen der Messung einzelner Komponenten durchgeführt werden kann, da hierfür nebenläufige virtuelle Maschinen benötigt werden. Interessante Parameter für die Komponente RAM sind daher:

- Virtualisierungstechnik
- Virtualisierungsunterstützung in Hardware
- Paravirtuelle Treiber
- Gastbetriebssystem
- Lesender / Schreibender Zugriff
- Sequentieller / Zufälliger Zugriff

3.3.4.3 Netz

Zu messende Parameter beim Zugriff auf virtuelle Netze

Interessante Parameter für die Komponente Netz sind neben den allgemeinen Parametern vor allem die Lokation und Art der kommunizierenden Maschinen, da virtuelle Maschinen im Gegensatz zu physischen Maschinen nur in definierten Zeitfenstern aktiv sind und Daten senden bzw. empfangen können. Daher sind folgende Einflussfaktoren interessant:

- Virtualisierungstechnik
- Virtualisierungsunterstützung in Hardware
- Paravirtuelle Treiber
- Gastbetriebssystem
- Quelle-Ziel
  - VM zu VM; identischer Host
  - VM zu VM; unterschiedlicher Host
  - VM zu PM
  - PM zu VM

3.3.4.4 Disk

Zu messende Parameter beim Zugriff auf virtuelle Festplatten

Die Einflussfaktoren auf den Wirkungsgrad von Hintergrundspeicher sind annähernd identisch mit denen für RAM. Zusätzlich existieren bei Hintergrundspeichern jedoch zwei Dateisysteme (eines auf dem Hintergrundspeicher, ein zweites im Abbild der virtuellen Maschine), die Einfluss auf die Effizienz im Zugriff auf Daten besitzen. Ferner existieren unterschiedliche Verfahren zum Zugriff auf den Speicher, die entweder Datei- oder Blockbasiert arbeiten. In dieser Messreihe wird ausschließlich der Zugriff via iSCSI verwendet, FibreChannel und NFS werden nicht betrachtet. Diese Festlegung stellt einen Kompromiss aus Leistung und Kosten dar. Auf Grund der höheren Übertragungsleistung von FibreChannel kann mit dieser Technologie voraussichtlich eine höhere Leistung erzielt werden, jedoch kostet eine dementsprechende Infrastruktur überproportional viel mehr und stand für Tests leider nicht zur Verfügung. NFS als dritte Variante arbeitet dateibasiert, und hat allein aus diesem Grunde bereits einen geringeren Wirkungsgrad, da es auf einer höheren Schicht arbeitet als FibreChannel oder iSCSI. Eine Variante, die jedoch in diesem Kontext berücksichtigt werden soll, ist die Untersuchung des Einflusses von Host Based Adapters in Hardware im Vergleich zu reinen Software iSCSI Initiators, da die Auslagerung dieser Funktionalität an dedizierte Hardware eventuell wertvolle CPU Ressourcen sparen kann. Folgende Parameter werden daher im Kontext der Komponente Disk untersucht:

- Virtualisierungstechnik
- Virtualisierungsunterstützung in Hardware inkl. iSCSI HBA
- Paravirtuelle Treiber
- Gastbetriebssystem
- Dateisysteme
- Lesender / Schreibender Zugriff
- Sequentieller / Zufälliger Zugriff

Parameter nebenläufiger virtueller Maschinen

Für die Messung paralleler Einflussfaktoren gemäß Tabelle 3.2 werden Konfigurationen, die in den Einzelmessungen einen hohen Wirkungsgrad erreicht haben, parallel gemessen. Folgende Einflussfaktoren werden hierbei zusätzlich interessant:

- Bindung einzelner vCPUs an dedizierte Kerne
- Anzahl der aktiven Instanzen
- Typen der aktiven Instanzen

3.3.5 Übertragbarkeit der Messergebnisse

Abstraktion von konkreten Produkten

Alle im folgenden Kapitel untersuchten Host-Virtualisierungslösungen liegen bereits in mehreren Major-Versions vor oder basieren auf Produkten, die bereits länger auf dem Markt erhältlich sind, so dass nicht zu erwarten ist, dass die einzelnen Produkte gravierende Implementierungsschwachstellen besitzen. Gleichzeitig sind alle Produkte Hybride der theoretischen Virtualisierungsmodelle, so dass abweichende Messungen einzelner Produkte von gleichartigen Implementierungskonzepten (siehe Tabelle 3.3) einen Hinweis auf eine Implementierungsschwäche geben können, sollte dennoch eine solche vorliegen. Die durchgeführten Messungen einer Gruppe von Konfigurationen haben damit einen Anspruch auf Allgemeingültigkeit; sie beschreiben damit nicht nur das Verhalten eines spezifischen Produktes in einer spezifischen Version, sondern auch die Leistungsfähigkeit eines Virtualisierungskonzeptes bezogen auf eine virtualisierte Komponente. Im Falle einer signifikanten Änderung an der Implementierung eines Virtualisierers ist die Messung dieses speziellen Produktes eventuell nicht mehr gültig. Die Methodik der Messung bleibt allerdings nach wie vor valide, so dass mit relativ geringem Aufwand neue, vergleichbare Messungen vorgenommen werden können.

4 Messungen und Ergebnisse

Inhalt

Gliederung in Einzel- und Parallel-Messungen

Die im folgenden Kapitel durchgeführten Messungen sind wie im Abschnitt 3.3.3 erläutert in zwei Teile gegliedert. Der erste Teil beschäftigt sich mit der Messung des Wirkungsgrades der Virtualisierung von einzelnen Komponenten, während der zweite Teil sich mit der Messung von parallel laufenden Instanzen von virtuellen Maschinen beschäftigt. Im ersten Fall genügt es hierzu eine einzige virtuelle Maschine pro Virtualisierungslösung zu instanziieren und mit dieser entsprechende Messungen durchzuführen. Lediglich Einflussfaktoren, die von mehreren virtuellen Maschinen abhängig sind, wie zum Beispiel die Leistungsfähigkeit der physischen Systemarchitektur oder Kommunikationsdurchsatz zwischen mehreren virtuellen Maschinen, benötigen mehrere virtuelle Instanzen und eventuell auch unterschiedliche Applikationen. Der Abschnitt 4.2 beschäftigt sich daher mit der Messung des Wirkungsgrades einzelner virtualisierter Komponenten, exemplarisch am Beispiel RAM, Disk, CPU und Netz. Anschließend werden im Abschnitt 4.3 Messungen durchgeführt, die das Zusammenspiel mehrerer virtueller Maschinen untersuchen. Ziel der Messungen sind hier unter anderem die Grenzen von hardwareseitigen Caches und Bussen sowie eventuelle Schwächen im Hypervisor selbst zu finden. Zunächst jedoch beschreibt das folgende Kapitel 4.1 die allgemein gültigen Randbedingungen der durchgeführten Messungen.

4.1 Randbedingungen

In den folgenden abschnitten wird die Zusammensetzung des Testsystems, der Virtualisierungslösungen und deren Konfiguration sowie die durchzuführenden Benchmarks beschrieben.

4.1.1 Eingesetzte Hardware

Alle Messungen wurden auf baugleichen Systemen ausgeführt, die die nachfolgend aufgelisteten Komponenten umfassen.

CPU AMD Athlon64 X2 4800+
Sockel: AM2 Anzahl Kerne: 2
Typ: Brisbane
Revision: G2
L1-Cache: je Kern 64 + 64 KB (Daten + Instruktionen)
L2-Cache: je Kern 512 KB mit Prozessortakt
Befehlssätze: MMX, Extended 3DNow!, SSE, SSE2, SSE3, NX-Bit, AMD64, Cool'n'Quiet, AMD-V
Fertigungstechnik: 65 nm

Mainboard ASUS M2N-SLI Deluxe
Chipsatz: nForce570 SLI mit 1.000 MHz FSB und HyperTransport 2.0
Netz: 2x 1000 Mbit/s LAN Marvell Yukon 88E8056
SATA Controler 1: nVidia Corporation MCP55 SATA Controller
SATA Controler 2: JMicron JMB363 SATA 3.0 Gb/s

Hauptspeicher 4096 MB DDR2-667 Samsung Cl5
Anzahl Module: 4 je 1024 MB
Dual Channel: ja

Festplatte Western Digital WD5000AAKS
Interface: SATA2
Rotationsgeschwindigkeit: 7200 RPM
Mittlere Zugriffszeit 8,9 ms
Cache: 16 MB
Kapazität: 500 GB

Netz Intel PRO/ 1000 GT Desktop Adapter
Interface: PCI

4.1.2 Untersuchte Virtualisierungslösungen

Analyse von Virtualisierungstechniken durch Analyse exemplarischer Produkte

Für jede existierende Virtualisierungstechnik (Vollvirtualisierung, Paravirtualisierung, OS-Virtualisierung, Emulation) muss ihm Rahmen der in diesem Kapitel durchgeführten Messreihen ein Vertreter untersucht werden, um einen vollständigen Vergleich anstellen zu können. Die Liste der für den Einsatz in Rechenzentren geeigneten und verbreiteten Virtualisierungslösungen umfasst derzeit die folgenden vier Produkte in den zum Testzeitpunkt aktuellsten Versionen: VMware ESXi 3.5, Xen 3.2, Parallels Virtuozzo 3.5 und Microsoft Hyper-V. Bei den Produkten VMware ESXi, Microsoft Hyper-V und Xen im HVM Modus handelt es sich primär um Vollvirtualisierer, während Xen in der ursprünglichen Variante ein Paravirtualisierer ist. Virtuozzo setzt primär auf das Konzept der OS-Virtualisierung, reine Emulatoren existieren im Serverbereich hingegen gar nicht. Da ein Vergleich der Produkte komponentenbezogen durchgeführt werden muss, genügt eine generelle Kategorisierung der Produkte als ganzes nicht. Stattdessen wird eine Kategorisierung nach eingesetzten Virtualisierungstechniken auf Komponentenebene benötigt, welche in Tabelle 4.1 gegeben ist. Alle Messergebnisse von Komponenten, welche in einer Tabellenspalte identische Einträge enthalten, sollten zumindest ähnliche Ergebnisse bei identischen Messungen liefern.

	CPU	RAM	Disk	Netz
VMware ESXi 3.5	Vollvirt.	Vollvirt.	Emuliert	Emuliert
VMware ESXi 3.5 + VMware Tools	Vollvirt.	Vollvirt.	Paravirt.	Paravirt.
Xen 3.2 Para	Paravirt.	Paravirt.	Paravirt.	Paravirt.
Xen 3.2 HVM	Vollvirt. [1]	Vollvirt. [1]	Emuliert	Emuliert
Xen 3.2 HVM + XenTools	Vollvirt. [1]	Vollvirt. [1]	Emuliert	Emuliert
Virtuozzo 3.5	-	Separiert	Separiert	Paravirt.
Microsoft Hyper-V	Vollvirt. [1]	Vollvirt. [1]	Emuliert	Emuliert
Microsoft Hyper-V + Integration Services	Vollvirt. [1]	Vollvirt. [1]	Paravirt.	Paravirt.

Tabelle 4.1: Hybride Virtualisierungsansätze existierender Produkte

4.1.3 Konfiguration

	Windows Server 2003 SP2 32/64-Bit	Ubuntu Server 9.04 32/64-Bit
VMware ESXi 3.5		
CPU	1 Kern (automatisch gebunden)	1 Kern (automatisch gebunden)
RAM	1024 MB	1024 MB
NIC	VMware Accelerated AMD PCNet	VMware Accelerated AMD PCNet
Controller	LSI Logic PCI-X Ultra 320	LSI Logic PCI-X Ultra 320
Disk	VMware Virtual SCSI Disk	VMware Virtual SCSI Disk
VMware ESXi 3.5 + VMware Tools		
CPU	1 Kern (automatisch gebunden)	1 Kern (automatisch gebunden)
RAM	1024 MB	1024 MB
NIC	VMware Accelerated AMD PCNet	VMware Accelerated AMD PCNet
Controller	LSI Logic PCI-X Ultra 320	LSI Logic PCI-X Ultra 320
Disk	VMware Virtual SCSI Disk	VMware Virtual SCSI Disk
Xen 3.2 Para		
CPU	1 Kern (automatisch gebunden)	1 Kern (automatisch gebunden)
RAM	1024 MB	1024 MB
NIC	Paravirtualisiert	Paravirtualisiert
Controller	Paravirtualisiert	Paravirtualisiert
Disk	Paravirtualisiert	Paravirtualisiert

	Windows Server 2003 SP2 32/64-Bit	Ubuntu Server 9.04 32/64-Bit
Xen 3.2 HVM		
CPU	1 Kern (automatisch gebunden)	1 Kern (automatisch gebunden)
RAM	1024 MB	1024 MB
NIC	Realtek RTL8139 Family PCI	Realtek RTL8139 Family PCI
Controller	Standard IDE Controller	Standard IDE Controller
Disk	Standard IDE Disk	Standard IDE Disk
Xen 3.2 HVM + XenTools		
CPU	1 Kern (automatisch gebunden)	-
RAM	1024 MB	-
NIC	Realtek RTL8139 Family PCI	-
Controller	Standard IDE Controller	-
Disk	Standard IDE Disk	-
Virtuozzo 3.5		
CPU	1 Kern (automatisch gebunden)	-
RAM	1024 MB	-
NIC	Paravirtualisiert	-
Controller	wie Host	-
Disk	wie Host	-
OpenVZ		
CPU	-	1 Kern (automatisch gebunden)
RAM	-	1024 MB
NIC	-	Paravirtualisiert
Controller	-	wie Host
Disk	-	wie Host
Microsoft Hyper-V + Integration Services		
CPU	1 Kern (automatisch gebunden)	1 Kern (automatisch gebunden)
RAM	1024 MB	1024 MB
NIC	MS Virtual Machine Bus Network	MS Virtual Machine Bus Network
Controller	Intel 82371 AB/EB Bus Master	Intel 82371 AB/EB Bus Master
Disk	MS Vitual HD	MS Virtual HD

Tabelle 4.2: Konfiguration der eingesetzten virtuellen Maschinen

4.1.4 Benchmarks

Kriterien für die Auswahl von Benchmarks

Synthetische Benchmarks zur Messung von CPU, Arbeitsspeicher, Hintergrundspeicher oder Netzanbindung existieren in rauen Mengen. Die Anforderungen, die die Messungen in dieser Arbeit stellen, erfüllen jedoch nur sehr wenige Werkzeuge. Zum einen müssen die Benchmarks lauffähig auf den zwei Plattformen Linux und Windows sein, zum anderen müssen sowohl 32-Bit als auch 64-Bit Architekturen

unterstützt werden, um garantieren zu können, dass alle Maschinen den exakt identischen Benchmark ausführen können und damit Vergleichbarkeit gewährleistet ist. Ein weiterer Punkt ist die Verfügbarkeit des Benchmarks im Quellcode, da er auf die Verwendbarkeit in virtuellen Maschinen untersucht und gegebenenfalls angepasst werden muss. Der Grund für die eventuell notwendigen Modifikationen ist, dass Zeitmessung in virtuellen Maschinen mit Ungenauigkeiten verbunden sein kann. Es besteht daher die Gefahr, dass eventuell Messfehler das Ergebnis eines Virtualisierers nach oben oder unten verfälschen und damit zu falschen Schlussfolgerungen führen können, wenn Instruktionen zur Zeitmessung im Code des Benchmarks nicht ersetzt werden. Für Messung der reinen Rechenleistung bietet sich der Benchmark Linpack an, der aufgrund seines Einsatzes zur Ermittlung der TOP 500 Liste der Supercomputer im HPC Bereich weite Verbreitung gefunden hat. Für die Bereiche Netz und Hintergrundspeicher eignet sich das Programm Iometer. Es erstellt neben dem reinen Durchsatz auch Protokoll über die Anzahl getätigter I/O-Aktionen. Beide Werte sind sowohl als Durchschnittswerte als auch als Peak-Werte verfügbar. Parametrisieren lässt sich Iometer mit Lese/Schreibe Verhältnissen sowie einem Verhältnis zwischen sequentiellen und zufälligen Zugriffsmustern. Des Weiteren lassen sich verschiedene Blockgrößen definieren mit denen der Test durchgeführt werden soll. Im Bereich Arbeitsspeicher wird auf das Programm RAMspeed zurückgegriffen, welches ebenfalls sequentielle und nichtsequentielle Zugriffe auf den Arbeitsspeicher und auf Caches erlaubt. Beide Zugriffsarten werden auch von RAMspeed sowohl lesend als auch schreibend getestet.

4.2 Messung des Wirkungsgrades von Hardwarekomponenten

In den folgenden Abschnitten wird der Wirkungsgrad für die Komponenten CPU (Abschnitt 4.2.1), RAM (Abschnitt 4.2.2), Netz (Abschnitt 4.2.3) und Disk (Abschnitt 4.2.4) ermittel. Hierzu werden pro zu ermittelndem Wirkungsgrad einer Komponente Benchmarks ausgewählt, Messungen durchgeführt und ausgewertet sowie der Wirkungsgrad bestimmt.

4.2.1 Messungen an der Komponente CPU

Beschreibung des Messverfahrens

Zur Messung der Leistung von CPUs existiert eine Vielzahl an Programmen und Tools. Die bekanntesten unter ihnen sind unter anderem die Benchmarks SpecInt und SpecFloat der Standard Performance Evaluation Corporation, sowie Linpack, der zur Ermittlung der TOP 500 Liste herangezogen wird. Für die Messungen in dieser Arbeit kommt ausschließlich Linpack zum Einsatz, da dieser im Gegensatz zu den beiden anderen Benchmarks kostenlos sowie im Source Code verfügbar ist. Linpack ist von seinem Ursprung gesehen eine numerische Programmbibliothek

zum Lösen von linearen Gleichungssystemen und belastet beinahe ausschließlich die CPU eines Rechners. Diese Tatsache ist der Grund dafür, dass Linpack in die Kritik geraten ist kein ausgeglichenes Maß für alle Architekturen moderner Hochleistungssysteme zu sein, da er keine Aussagen über Interconnects und gemeinsamen Speicher eines verteilten Hochleistungssystems zulässt. Für Messungen an der Komponente CPU, wie sie in dieser Arbeit benötigt werden, ist Linpack jedoch perfekt geeignet, da dieses Verhalten exakt den Anforderungen entspricht.

Zeitmessung

Ein Problem, das entsteht, wenn Linpack auf virtuellen Maschinen ausgeführt wird, ist, dass Linpack zur Ermittlung der FLOP-Rate zu mehreren Zeitpunkten die Systemzeit der virtuellen Maschine liest. Die Systemzeit der virtuellen Maschine ist jedoch in der Regel nicht exakt mit der realen Zeit der physischen Maschine synchronisiert, so dass Messfehler auftreten können. Aus diesem Grund muss der Benchmark modifiziert werden, so dass alle Virtualisierer den identischen Messbedingungen unterliegen. Am einfachsten realisierbar ist dabei eine Messung der Zeit über das Netz mittels ICMP, wie es in [VMwa 06] vorgeschlagen wird. Mit dieser Methode kann allerdings nur die Messung der Dauer des gesamten Benchmarks erfolgen, nicht die von vielen Unterprogrammaufrufen. Des Weiteren wird bei dieser Methode keine Zeit an die Benchmark-Umgebung übertragen, so dass dieser keine Werte zur Berechnung des Leistungsindex zur Verfügung stehen. Eine Möglichkeit dieses Problem zu lösen besteht darin, den eigentlichen Benchmark um einen TimeClient zu erweitern, der bei Bedarf die aktuelle Zeit von einem entsprechenden TimeServer über eine stehende TCP-Verbindung anfordern kann. Die mit diesem Verfahren erhaltenen Messwerte werden bedingt durch den zusätzlichen Aufwand bei der Kommunikation mit dem TimeServer geringfügig schlechter ausfallen als die des nativen Linpack Benchmarks. Dies stellt jedoch nur im direkten Vergleich mit entsprechenden Messungen ein Problem dar, für die Bestimmung des Wirkungsgrades ist diese Tatsache nicht relevant, da sowohl die Messungen des virtuellen als auch des physischen Systems auf den identischen Messverfahren basieren. Eine modifizierte Variante des Linpack Benchmarks in C liegt dieser Arbeit im Anhang A.1 bei.

Variable Einflussgrößen

Die folgenden Einflussfaktoren bezogen auf die Standardkonfiguration werden im Laufe der CPU basierten Messungen unabhängig voneinander variiert, um ihren Einfluss auf den Wirkungsgrad der untersuchten Virtualisierer zu messen. Aufgrund technischer Abhängigkeiten sind nicht alle Permutationen der angegebenen Parameter implementierbar. So kann man bei Xen im Paravirtualisierungsmodus beispielsweise keine zusätzlichen Treiber für Paravirtualisierung installieren, Hyper-V benötigt zwangsweise Virtualisierungsunterstützung durch den Prozessor und OpenVZ / Virtuozzo können nur virtuelle Gastinstanzen erzeugen, die vom selben Typ sind wie das Host- Betriebssystem. Aus diesem Grund scheinen einige Messergebnisse in den folgenden Diagrammen zu fehlen. Die leeren Felder wurden dennoch in den Diagrammen belassen, um die realisierbaren und sinnvollen Konstellationen aufzuzeigen. Diese Tatsache bezieht sich nicht nur auf die folgenden CPU basierten Messungen, sondern auf alle Messungen, die sich mit der

Analyse einer einzelnen Komponente auseinandersetzen. Im Einzelnen werden alle Messungen auf den genannten Virtualisierungslösungen, jeweils mit und ohne zusätzlich installierten Treibern, auf unterschiedlichen Gastbetriebssystemen sowie mit aktivierter und ausgeschalteter Prozessorunterstützung durchgeführt.

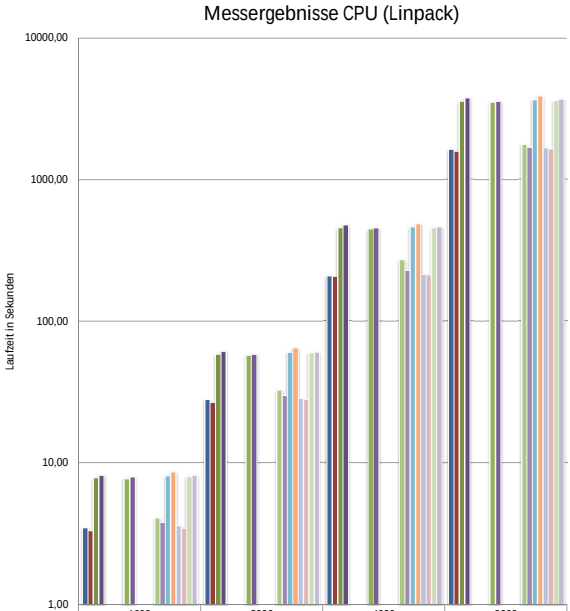
1. Virtualisierungslösung
   a) Nativ (keine Virtualisierung)
   b) Xen (Paravirtualisiert)
   c) Xen (Vollvirtualisiert)
   d) OpenVZ (OS-Virtualisierung)
   e) Virtuozzo (OS-Virtualisierung)
   f) Hyper-V (Vollvirtualisiert)
   g) VMware ESXi (Vollvirtualisiert)
2. Paravirtualisierende Treiber
   a) keine paravirtualisierenden Treiber
   b) win-pv Treiber (Xen)
   c) Integration Tools (Hyper-V)
   d) VMware Tools (ESXi)
3. Gast
   a) Windows Server 2003 32 Bit
   b) Windows Server 2003 64 Bit
   c) Ubuntu Linux 32 Bit
   d) Ubuntu Linux 64 Bit
4. Prozessorunterstützung
   a) AMD-V aktiviert
   b) AMD-V deaktiviert

4.2.1.1 Messergebnisse

Messreihen

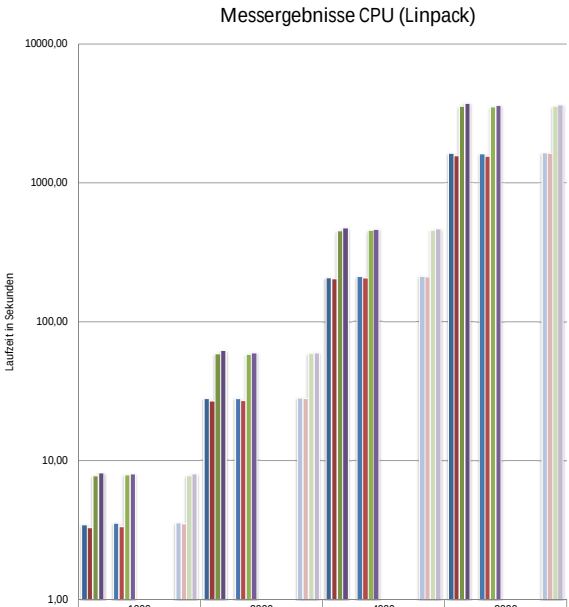
Zur Messung des Wirkungsgrades der CPU wurden vier Gruppen gebildet, die sich aus den möglichen Belegungen der Parameter AMD-V aktiviert und deaktiviert sowie Treiber zur Paravirtualisierung installiert / nicht installiert zusammensetzen. In einer dieser Gruppen wurden jeweils alle möglichen Permutationen der verbleibenden Parameter Virtualisierungslösung und Gastbetriebssystem gemessen.

Die Messergebnisse der Dauer der einzelnen Linpackläufe werden durch die Abbildungen 4.1, 4.2, 4.3 und 4.4 dargestellt. Die Ergebnisse der nativen Messungen wurden in alle Diagramme übernommen, um das Abschneiden eines einzelnen Test grafisch zu vereinfachen. Zu beachten sind die exponentielle X-Achse sowie die logarithmische Y-Achse der abgebildeten Diagramme!



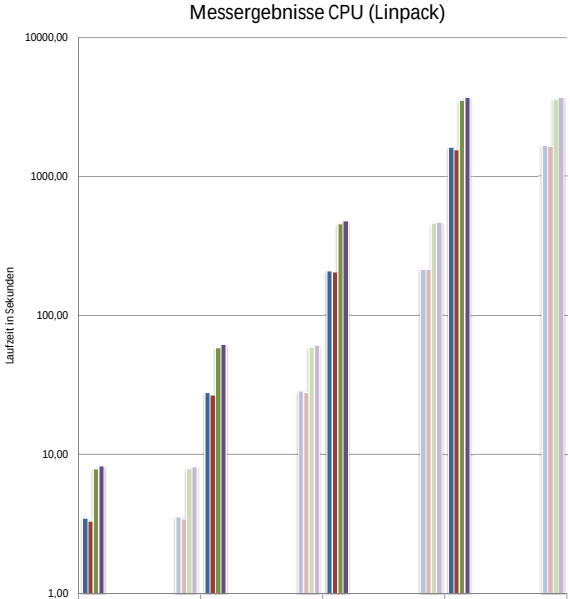
	1000	2000	4000	8000
Ubuntu 9.04 32 Bit - Nativ	3,49	27,88	209,14	1620,58
Ubuntu 9.04 64 Bit - Nativ	3,31	26,79	205,38	1566,83
Win2K3 32 Bit - Nativ	7,84	58,16	452,45	3547,73
Win2K3 64 Bit - Nativ	8,23	61,20	476,94	3745,25
Ubuntu 32 Bit - Xen 64 Bit Para				
Ubuntu 64 Bit - Xen 64 Bit Para				
Ubuntu 32 Bit - Xen 64 Bit Voll				
Ubuntu 64 Bit - Xen 64 Bit Voll				
Win2K3 32 Bit - Xen 64 Bit Voll	7,75	57,69	447,88	3515,47
Win2K3 64 Bit - Xen 64 Bit Voll	7,97	58,86	456,84	3589,75
Ubuntu 32 Bit - OpenVZ 32 Bit				
Ubuntu 64 Bit - OpenVZ 64 Bit				
Win2K3 32 Bit - Virtuozzo 32 Bit				
Win2K3 64 Bit - Virtuozzo 64 Bit				
Ubuntu 32 Bit - HyperV 64	4,09	32,53	271,44	1764,07
Ubuntu 64 Bit - HyperV 64	3,80	29,82	229,05	1680,38
Win2K3 32 Bit - HyperV 64	8,10	59,98	465,65	3652,66
Win2K3 64 Bit - HyperV 64	8,64	64,37	487,36	3874,46
Ubuntu 32 Bit - ESXi 3.5 32 Bit	3,59	28,25	214,43	1662,37
Ubuntu 64 Bit - ESXi 3.5 32 Bit	3,47	27,89	213,28	1643,35
Win2K3 32 Bit - ESXi 3.5 32 Bit	7,98	59,56	457,64	3577,77
Win2K3 64 Bit - ESXi 3.5 32 Bit	8,13	60,11	466,83	3664,55

Abbildung 4.1: Messwerte Linpack (AMD-V, paravirt. Treiber)



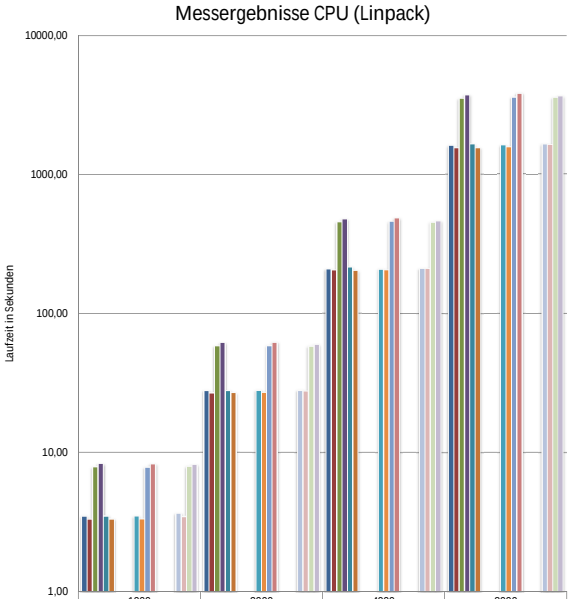
	1000	2000	4000	8000
Ubuntu 9.04 32 Bit - Nativ	3,49	27,88	209,14	1620,58
Ubuntu 9.04 64 Bit - Nativ	3,31	26,79	205,38	1566,83
Win2K3 32 Bit - Nativ	7,84	58,16	452,45	3547,73
Win2K3 64 Bit - Nativ	8,23	61,20	476,94	3745,25
Ubuntu 32 Bit - Xen 64 Bit Para				
Ubuntu 64 Bit - Xen 64 Bit Para				
Ubuntu 32 Bit - Xen 64 Bit Voll	3,54	28,04	211,02	1630,53
Ubuntu 64 Bit - Xen 64 Bit Voll	3,35	26,83	206,62	1571,27
Win2K3 32 Bit - Xen 64 Bit Voll	7,83	57,75	449,78	3538,40
Win2K3 64 Bit - Xen 64 Bit Voll	7,99	59,20	460,11	3625,05
Ubuntu 32 Bit - OpenVZ 32 Bit				
Ubuntu 64 Bit - OpenVZ 64 Bit				
Win2K3 32 Bit - Virtuozzo 32 Bit				
Win2K3 64 Bit - Virtuozzo 64 Bit				
Ubuntu 32 Bit - HyperV 64				
Ubuntu 64 Bit - HyperV 64				
Win2K3 32 Bit - HyperV 64				
Win2K3 64 Bit - HyperV 64				
Ubuntu 32 Bit - ESXi 3.5 32 Bit	3,59	28,33	212,22	1650,41
Ubuntu 64 Bit - ESXi 3.5 32 Bit	3,54	28,15	210,24	1635,32
Win2K3 32 Bit - ESXi 3.5 32 Bit	7,84	59,05	454,92	3573,53
Win2K3 64 Bit - ESXi 3.5 32 Bit	8,09	59,84	465,42	3660,40

Abbildung 4.2: Messwerte Linpack (AMD-V, keine paravirt. Treiber)



	1000	2000	4000	8000
Ubuntu 9.04 32 Bit - Nativ	3,49	27,88	209,14	1620,58
Ubuntu 9.04 64 Bit - Nativ	3,31	26,79	205,38	1566,83
Win2K3 32 Bit - Nativ	7,84	58,16	452,45	3547,73
Win2K3 64 Bit - Nativ	8,23	61,20	476,94	3745,25
Ubuntu 32 Bit - Xen 64 Bit Para				
Ubuntu 64 Bit - Xen 64 Bit Para				
Ubuntu 32 Bit - Xen 64 Bit Voll				
Ubuntu 64 Bit - Xen 64 Bit Voll				
Win2K3 32 Bit - Xen 64 Bit Voll				
Win2K3 64 Bit - Xen 64 Bit Voll				
Ubuntu 32 Bit - OpenVZ 32 Bit				
Ubuntu 64 Bit - OpenVZ 64 Bit				
Win2K3 32 Bit - Virtuozzo 32 Bit				
Win2K3 64 Bit - Virtuozzo 64 Bit				
Ubuntu 32 Bit - HyperV 64				
Ubuntu 64 Bit - HyperV 64				
Win2K3 32 Bit - HyperV 64				
Win2K3 64 Bit - HyperV 64				
Ubuntu 32 Bit - ESXi 3.5 32 Bit	3,58	28,48	214,47	1661,35
Ubuntu 64 Bit - ESXi 3.5 32 Bit	3,45	27,90	213,39	1644,03
Win2K3 32 Bit - ESXi 3.5 32 Bit	7,89	58,58	455,73	3586,50
Win2K3 64 Bit - ESXi 3.5 32 Bit	8,14	60,39	467,33	3668,78

Abbildung 4.3: Messwerte Linpack (kein AMD-V, paravirt. Treiber)



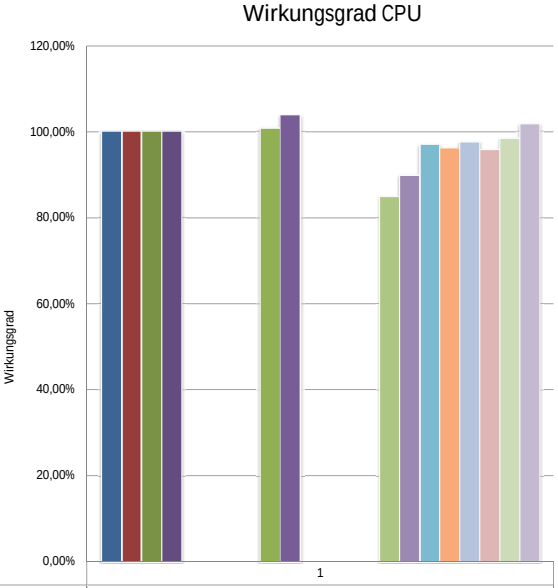
	1000	2000	4000	8000
Ubuntu 9.04 32 Bit - Nativ	3,49	27,88	209,14	1620,58
Ubuntu 9.04 64 Bit - Nativ	3,31	26,79	205,38	1566,83
Win2K3 32 Bit - Nativ	7,84	58,16	452,45	3547,73
Win2K3 64 Bit - Nativ	8,23	61,20	476,94	3745,25
Ubuntu 32 Bit - Xen 64 Bit Para	3,50	27,88	214,64	1656,88
Ubuntu 64 Bit - Xen 64 Bit Para	3,33	26,88	202,69	1562,27
Ubuntu 32 Bit - Xen 64 Bit Voll				
Ubuntu 64 Bit - Xen 64 Bit Voll				
Win2K3 32 Bit - Xen 64 Bit Voll				
Win2K3 64 Bit - Xen 64 Bit Voll				
Ubuntu 32 Bit - OpenVZ 32 Bit	3,51	28,21	209,61	1624,24
Ubuntu 64 Bit - OpenVZ 64 Bit	3,33	27,01	206,54	1573,25
Win2K3 32 Bit - Virtuozzo 32 Bit	7,85	58,93	458,68	3591,00
Win2K3 64 Bit - Virtuozzo 64 Bit	8,31	62,32	486,30	3831,10
Ubuntu 32 Bit - HyperV 64				
Ubuntu 64 Bit - HyperV 64				
Win2K3 32 Bit - HyperV 64				
Win2K3 64 Bit - HyperV 64				
Ubuntu 32 Bit - ESXi 3.5 32 Bit	3,65	28,23	211,68	1649,67
Ubuntu 64 Bit - ESXi 3.5 32 Bit	3,46	27,81	211,72	1634,29
Win2K3 32 Bit - ESXi 3.5 32 Bit	7,91	58,46	453,98	3569,85
Win2K3 64 Bit - ESXi 3.5 32 Bit	8,13	59,89	465,05	3652,25

Abbildung 4.4: Messwerte Linpack (kein AMD-V, keine paravirt. Treiber)

4.2.1.2 Ermittlung des Wirkungsgrades

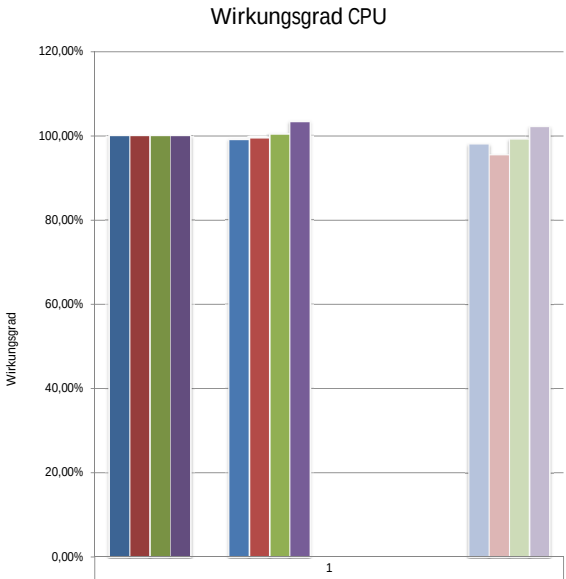
Wirkungsgrad der Virtualisierung von CPUs

Zur Ermittlung des Wirkungsgrades der gemessenen Ergebnisse wurde jeweils der Quotient aus der durchschnittlichen Dauer des Benchmarks im nativen Fall und Dauer des Benchmarks im virtuellen Fall berechnet. Damit ist der Wirkungsgrad nicht abhängig von einer konkreten Matrizen- beziehungsweise Problemgröße, so dass das Ergebnis nicht von unterschiedlich großen Frames der Seitenverwaltung abhängig ist. Der berechnete Wirkungsgrad für virtuelle CPUs ist in den Abbildungen 4.5, 4.6, 4.7 und 4.8 dargestellt.



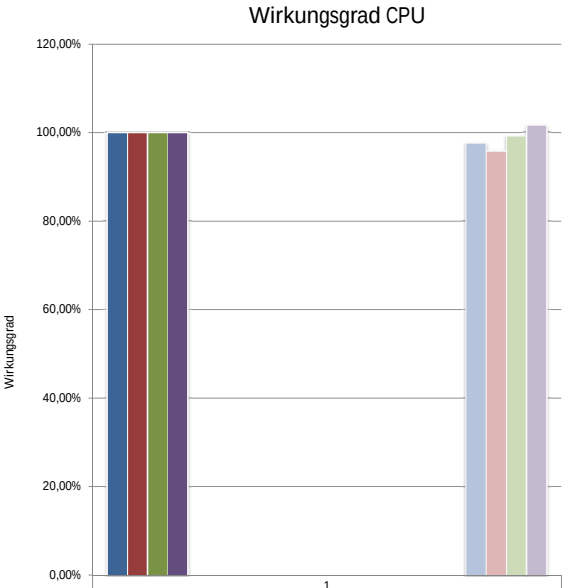
Ubuntu 9.04 32 Bit - Nativ	100,00%
Ubuntu 9.04 64 Bit - Nativ	100,00%
Win2K3 32 Bit - Nativ	100,00%
Win2K3 64 Bit - Nativ	100,00%
Ubuntu 32 Bit - Xen 64 Bit Para	
Ubuntu 64 Bit - Xen 64 Bit Para	
Ubuntu 32 Bit - Xen 64 Bit Voll	
Ubuntu 64 Bit - Xen 64 Bit Voll	
Win2K3 32 Bit - Xen 64 Bit Voll	100,98%
Win2K3 64 Bit - Xen 64 Bit Voll	103,99%
Ubuntu 32 Bit - OpenVZ 32 Bit	
Ubuntu 64 Bit - OpenVZ 64 Bit	
Win2K3 32 Bit - Virtuozzo 32 Bit	
Win2K3 64 Bit - Virtuozzo 64 Bit	
Ubuntu 32 Bit - HyperV 64	84,98%
Ubuntu 64 Bit - HyperV 64	89,96%
Win2K3 32 Bit - HyperV 64	97,03%
Win2K3 64 Bit - HyperV 64	96,21%
Ubuntu 32 Bit - ESXi 3.5 32 Bit	97,71%
Ubuntu 64 Bit - ESXi 3.5 32 Bit	95,81%
Win2K3 32 Bit - ESXi 3.5 32 Bit	98,48%
Win2K3 64 Bit - ESXi 3.5 32 Bit	101,85%

Abbildung 4.5: Wirkungsgrad CPU (AMD-V, paravirt. Treiber)



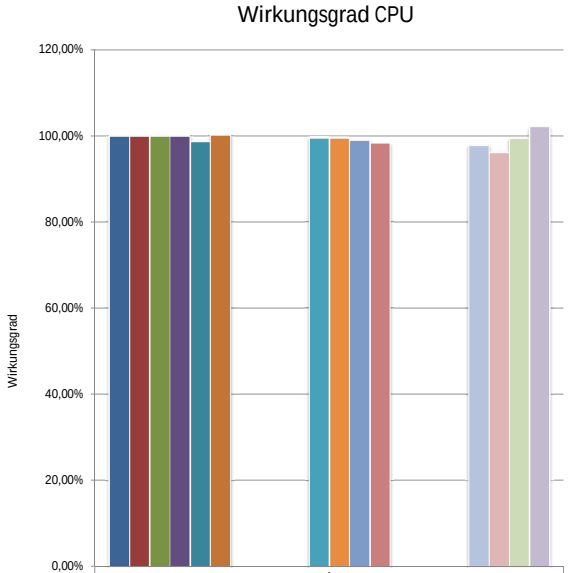
Ubuntu 9.04 32 Bit - Nativ	100,00%
Ubuntu 9.04 64 Bit - Nativ	100,00%
Win2K3 32 Bit - Nativ	100,00%
Win2K3 64 Bit - Nativ	100,00%
Ubuntu 32 Bit - Xen 64 Bit Para	
Ubuntu 64 Bit - Xen 64 Bit Para	
Ubuntu 32 Bit - Xen 64 Bit Voll	99,11%
Ubuntu 64 Bit - Xen 64 Bit Voll	99,49%
Win2K3 32 Bit - Xen 64 Bit Voll	100,42%
Win2K3 64 Bit - Xen 64 Bit Voll	103,34%
Ubuntu 32 Bit - OpenVZ 32 Bit	
Ubuntu 64 Bit - OpenVZ 64 Bit	
Win2K3 32 Bit - Virtuozzo 32 Bit	
Win2K3 64 Bit - Virtuozzo 64 Bit	
Ubuntu 32 Bit - HyperV 64	
Ubuntu 64 Bit - HyperV 64	
Win2K3 32 Bit - HyperV 64	
Win2K3 64 Bit - HyperV 64	
Ubuntu 32 Bit - ESXi 3.5 32 Bit	98,09%
Ubuntu 64 Bit - ESXi 3.5 32 Bit	95,55%
Win2K3 32 Bit - ESXi 3.5 32 Bit	99,31%
Win2K3 64 Bit - ESXi 3.5 32 Bit	102,20%

Abbildung 4.6: Wirkungsgrad CPU (AMD-V, keine paravirt. Treiber)



	1
Ubuntu 9.04 32 Bit - Nativ	100,00%
Ubuntu 9.04 64 Bit - Nativ	100,00%
Win2K3 32 Bit - Nativ	100,00%
Win2K3 64 Bit - Nativ	100,00%
Ubuntu 32 Bit - Xen 64 Bit Para	
Ubuntu 64 Bit - Xen 64 Bit Para	
Ubuntu 32 Bit - Xen 64 Bit Voll	
Ubuntu 64 Bit - Xen 64 Bit Voll	
Win2K3 32 Bit - Xen 64 Bit Voll	
Win2K3 64 Bit - Xen 64 Bit Voll	
Ubuntu 32 Bit - OpenVZ 32 Bit	
Ubuntu 64 Bit - OpenVZ 64 Bit	
Win2K3 32 Bit - Virtuozzo 32 Bit	
Win2K3 64 Bit - Virtuozzo 64 Bit	
Ubuntu 32 Bit - HyperV 64	
Ubuntu 64 Bit - HyperV 64	
Win2K3 32 Bit - HyperV 64	
Win2K3 64 Bit - HyperV 64	
Ubuntu 32 Bit - ESXi 3.5 32 Bit	97,64%
Ubuntu 64 Bit - ESXi 3.5 32 Bit	95,89%
Win2K3 32 Bit - ESXi 3.5 32 Bit	99,21%
Win2K3 64 Bit - ESXi 3.5 32 Bit	101,66%

Abbildung 4.7: Wirkungsgrad CPU (kein AMD-V, paravirt. Treiber)



Ubuntu 9.04 32 Bit - Nativ	100,00%
Ubuntu 9.04 64 Bit - Nativ	100,00%
Win2K3 32 Bit - Nativ	100,00%
Win2K3 64 Bit - Nativ	100,00%
Ubuntu 32 Bit - Xen 64 Bit Para	98,76%
Ubuntu 64 Bit - Xen 64 Bit Para	100,22%
Ubuntu 32 Bit - Xen 64 Bit Voll	
Ubuntu 64 Bit - Xen 64 Bit Voll	
Win2K3 32 Bit - Xen 64 Bit Voll	
Win2K3 64 Bit - Xen 64 Bit Voll	
Ubuntu 32 Bit - OpenVZ 32 Bit	99,45%
Ubuntu 64 Bit - OpenVZ 64 Bit	99,42%
Win2K3 32 Bit - Virtuozzo 32 Bit	99,00%
Win2K3 64 Bit - Virtuozzo 64 Bit	98,27%
Ubuntu 32 Bit - HyperV 64	
Ubuntu 64 Bit - HyperV 64	
Win2K3 32 Bit - HyperV 64	
Win2K3 64 Bit - HyperV 64	
Ubuntu 32 Bit - ESXi 3.5 32 Bit	97,85%
Ubuntu 64 Bit - ESXi 3.5 32 Bit	96,23%
Win2K3 32 Bit - ESXi 3.5 32 Bit	99,41%
Win2K3 64 Bit - ESXi 3.5 32 Bit	102,13%

Abbildung 4.8: Wirkungsgrad CPU (kein AMD-V, keine paravirt. Treiber)

4.2.1.3 Auswertung

Lineare Skalierbarkeit

Betrachtet man die Diagramme der Messergebnisse, so fällt zunächst ein linearer Anstieg der Dauer der Linpack-Läufe mit der Größe der zu berechnenden Matrix auf. Unabhängig von der Problemgröße ist die Effizienz der Virtualisierung in allen Problemstellungen für eine gegebene Virtualisierungslösung daher nahezu dieselbe. Durch Virtualisierung eingeführte zusätzliche Adressübersetzungen und deren unterschiedliche Größen der Speicherseiten haben nur geringeren Einfluss auf den Wirkungsgrad der CPU-Virtualisierung, können jedoch Schwankungen von wenigen Prozent verursachen und somit Wirkungsgrade größer 100% erklären. In diesen Fällen ist die Speicherverwaltung der Virtualisierer durch geeignetere Seitengrößen der Seitenverwaltung von nativen Betriebssystemen überlegen. Dies ist vermutlich jedoch problemspezifisch für den Linpack-Benchmark. Eine allgemeine Gültigkeit dieser Tatsache kann dadurch nicht abgeleitet werden, da sehr wahrscheinlich Probleme existieren, für die das Verhalten invers beobachtet werden kann. Weiterhin fällt auf, dass Messungen unter Windows beinahe um den Faktor zwei länger dauern als vergleichbare Messungen unter Linux. Dies ist jedoch keine Tatsache, deren Ursache in der Virtualisierung zu suchen ist, da das Verhalten ebenso bei nativen Systemen beobachtet werden kann. Den Wirkungsgrad der Messreihen unter Windows beeinflusst dieses Verhalten nicht, da der Wirkungsgrad lediglich einen Quotienten aus nativen und virtuellen Messergebnissen darstellt und somit beide Messungen unter identischen Randbedingungen durchgeführt wurden. Der Einfluss von Treibern zur Paravirtualisierung ist nicht messbar, da die Treibersammlungen zwar Treiber zum optimierten Zugriff auf I/O-Geräte besitzen, aber keine Komponente, die für eine Steigerung der reinen Computingleistung verantwortlich sein könnte. Da der Anteil der I/O-Aktionen und Kontextwechsel bei dem durchgeführten Linpack-Benchmark absichtlich minimal gehalten wurden, können daher weder künstlich erzwungene Paravirtualisierungstechniken noch Prozessorunterstützung in Form von AMD-V die Messergebnisse gravierend beeinflussen. Die Frage nach der besseren Architektur kann ebenfalls nicht eindeutig beantwortet werden. Während unter Linux 64-Bit Systeme stets minimal besser abschnitten als 32-Bit Systeme, stellt sich diese Tatsache unter Windows exakt invers dar. Das Phänomen ist ebenso wie die beobachtete längere Laufzeit des Benchmarks unter Windows unabhängig von der Art der Virtualisierung. Die Ursachen für dieses Verhalten sind innerhalb der untersuchten Betriebssysteme zu suchen. Das Verhalten selbst kann bei einem Austausch der Benchmark-Anwendung verschwinden oder in das Gegenteil umschlagen. Eine Analyse der exakten Ursache würde den Rahmen dieser Arbeit bei weitem sprengen, jedoch keine Hilfe zum Optimieren virtueller Infrastrukturen darstellen. In Summe schlagen sich alle Virtualisierungstechniken, egal ob sie auf SBE oder Paravirtualisierung oder Separierungsansätzen beruhen, sehr gut. Selbst der schlechteste, einzeln gemessene Wirkungsgrad liegt bei über 95%, während der beste bei ca. 103% liegt. Schätzt man den Effekt des Paging der Virtualisierer sowohl nach oben als nach unten auf ca. 3%, was dem Ausbrechen mancher Mess-

Windows rechnet länger

Treiber für Paravirtualisierung besitzen keinen Einfluss

Architekturen

Fazit

ergebnisse über 100% entsprechen würde, so würde dies einen mageren Vorsprung von 2% für Paravirtualisierung und OS-Virtualisierung bedeuten, der maximal im Hochleistungsrechnen interessant sein könnte. Eine Empfehlung eines Produktes oder einer Technologie aufgrund ihres Wirkungsgrades der CPU erscheint nicht sinnvoll. Lediglich Microsofts Hyper-V erreicht als einziges Produkt nur einen Wirkungsgrad von 85% und scheint daher momentan nicht geeignet für produktive Virtualisierung. Einen Rückschluss auf einen eventuell schlechteren technologischen Ansatz von Hyper-V lässt sich daraus nicht ableiten, da Hyper-V in großen Teilen sehr ähnlich zu Xen ist, welches einen sehr guten Wirkungsgrad erreichen konnte. Bei den Messergebnissen von Hyper-V dürfte es sich um Implementierungsschwächen handeln, die eventuell schon in der nächsten Version des Produktes behoben sein könnten.

4.2.2 Messungen an der Komponente RAM

Beschreibung des Messverfahrens

Als Werkzeug für den Arbeitsspeicher-Benchmark wurde RAMspeed [Rhet 09] ausgewählt. RAMspeed scheint das einzig existierende Programm zum Testen des Durchsatzes von Arbeitsspeicher zu sein, das sowohl unter Windows und Linux als auch auf beiden zu testenden Plattformen lauffähig ist. Ein anderes Programm, das diese Eigenschaften erfüllt, wurde auch nach intensiven Recherchen nicht gefunden. RAMspeed selbst besteht aus mehreren einzelnen Benchmarks, die den Durchsatz des Arbeitsspeichers und vorhandener Caches wahlweise lesend oder schreibend, sequentiell oder nicht sequentiell testen. Die Tests INTmark und FLOATmark untersuchen das Leistungsverhalten beim Lesen und Schreiben von sequentiellen Datenblöcken mit den Standardblockgrößen 1 Kb, 2 Kb, 4 Kb, 8 Kb, 16 Kb, 32 Kb, 64 Kb, 128 Kb, 256 Kb, 512 Kb, 1024 Kb, 2048 Kb, 4096 Kb, 8192 Kb, 16384 Kb und 32768 Kb. Die Tests INTmem und FLOATmem lesen beziehungsweise schreiben im Gegenzug dazu nichtsequentielle Daten in den Modi Copy, Scale, Add und Triad. Bei Copy handelt es sich um ein Kopieren von Daten von einer Speicherzelle in eine andere, Scale multipliziert die gelesenen Zelle mit einem konstantem Faktor, ehe sie zurück in den Arbeitsspeicher geschrieben wird, Add liest zwei Speicherzellen aus und schreibt deren Summe zurück und Triad skaliert die erste Zelle mit einem konstantem Faktor, addiert den Inhalt einer zweiten Zelle und schreibt das Ergebnis zurück in den Arbeitsspeicher. Diese einfachen Berechnungen sind neben dem reinen Lesen und Schreiben von Speicherzellen nötig, da manche Architekturen nur performanten Speicherzugriff realisieren können, wenn Lese- und Schreibanforderungen direkt aufeinander folgen und das Pipelining der CPU greift. Die Durchführung von Benchmarks mit einer entsprechenden Berechnungskomponente kann daher Einblicke in das Verhalten des Pipelining in virtuellen Maschinen geben. RAMspeed deckt mit seiner Funktionalität alle aufgestellten Anforderungen an einen Speichertest zur Messung des Wirkungsgrades von virtuellen Maschinen ab. Der Benchmark ist im Source Code erhältlich, enthält allerdings an den relevanten Stellen nur Assembler-Code, um eine Optimierung des Benchmarks durch einen Compiler zu verhindern. Eine

Analyse des Quellcodes ist daher zwar möglich, aber aufwändig. Leider existiert für jede unterstützte Architektur eine eigene Version des Benchmarks, die zwar untereinander funktional identisch sind, sich jedoch nicht wirklich vergleichen lassen, da sie Gebrauch von architekturspezifischen Befehlssätzen machen. Insbesondere die Länge bestimmter Datentypen und darauf angepasste Befehlssätze verhindern einen direkten Vergleich zwischen den nachfolgend untersuchten x86 und x64 Messungen. Für 64 Bit Windows-Systeme existiert keine angepasste Version des Benchmarks, daher wird für die entsprechenden Messungen die 32 Bit Version verwendet.

Keine Floating Point Messungen

Für die konkret durchgeführten Tests wurde auf die ursprünglich geplanten Messungen der Floating Point Arithmetik verzichtet, da erste Tests zeigten, dass diese sehr stark von der zugrundeliegenden Architektur abhängig sind. Da die Intuition des RAMspeed-Benchmarks in diesem Abschnitt die Untersuchung des Hauptspeichers und nicht die von Prozessorarchitekturen ist, wurden die Messungen nur mit Integer-Typen durchgeführt.

Zeitmessung

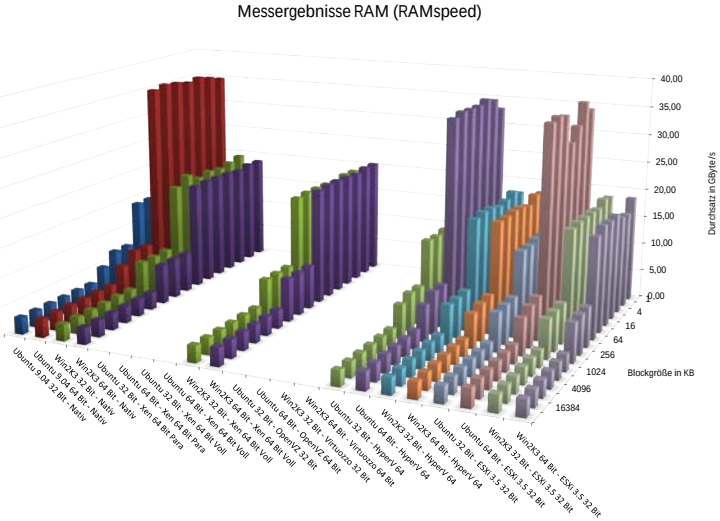
Den erzielten Durchsatz einer Messreihe ermittelt RAMspeed mittels dem Quotienten aus übertragenem Datenvolumen und Zeitintervall. Das Zeitintervall wird aus der Differenz zweier Systemaufrufe von *gettimeofday()* ermittelt. Die Zeit dieses Counters ist für Zeitintervalle von mindestens 10ms Dauer auch in virtuellen Maschinen hinreichend genau, als dass eine Anpassung des Benchmarks notwendig wäre [Stil 07, Drum 08]. Eine Anpassung des Benchmarks ist demnach nicht notwendig, wenn die einzelnen Testintervalle lange genug dauern, was mit Sekunden und Minutenbereichen hinreichend erfüllt ist.

Variable Einflussgrößen

1. Virtualisierungslösung
    a) Nativ (keine Virtualisierung)
    b) Xen (Paravirtualisiert)
    c) Xen (Vollvirtualisiert)
    d) OpenVZ (OS-Virtualisierung)
    e) Virtuozzo (OS-Virtualisierung)
    f) Hyper-V (Vollvirtualisiert)
    g) VMware ESXi (Vollvirtualisiert)
2. Paravirtualisierende Treiber
    a) keine paravirtualisierenden Treiber
    b) win-pv Treiber (Xen)
    c) Integration Tools (Hyper-V)
    d) VMware Tools (ESXi)

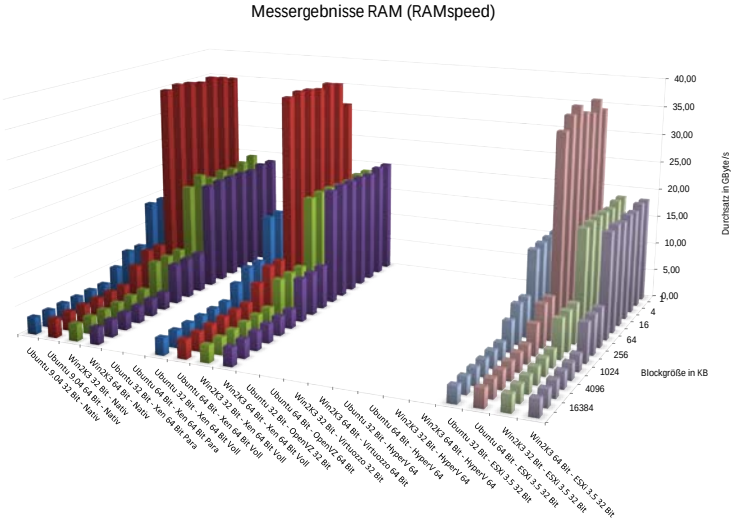
3. Gast
    a) Windows Server 2003 32 Bit
    b) Windows Server 2003 64 Bit
    c) Ubuntu Linux 32 Bit
    d) Ubuntu Linux 64 Bit
4. Prozessorunterstützung
    a) AMD-V aktiviert
    b) AMD-V deaktiviert
5. Aktion
    a) Lesen
    b) Schreiben
6. Typ
    a) Sequentiell
    b) Zufällig

4.2.2.1 Messergebnisse



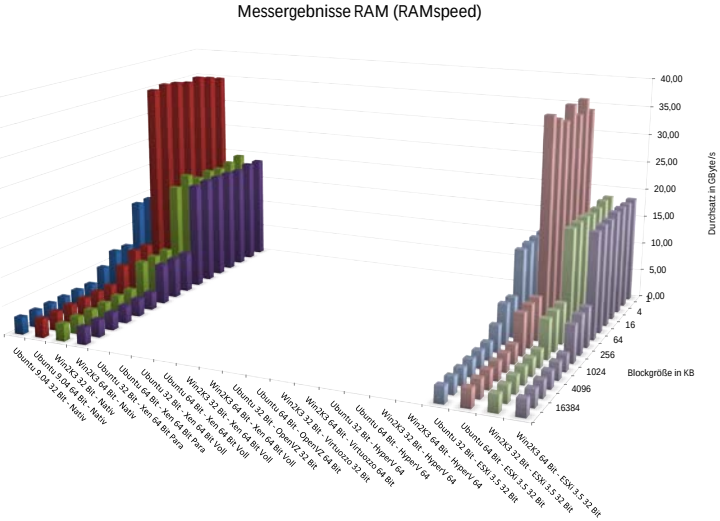
	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
Ubuntu 9.04 32 Bit - Nativ	14,10	14,26	14,04	14,20	14,29	14,33	14,28	7,12	7,12	5,12	3,07	3,08	3,03	3,02	3,01	3,01
Ubuntu 9.04 64 Bit - Nativ	34,86	35,60	36,24	35,86	36,38	36,64	36,20	7,65	7,71	5,91	3,34	3,34	3,32	3,32	3,31	3,31
Win2K3 32 Bit - Nativ	19,42	18,89	18,44	18,77	18,28	19,65	18,20	7,23	7,13	7,13	3,10	3,13	3,06	3,05	3,04	3,04
Win2K3 64 Bit - Nativ	18,72	18,89	18,44	18,77	18,89	18,98	18,85	7,13	7,13	7,12	3,11	3,11	3,05	3,04	3,03	3,03
Ubuntu 32 Bit - Xen 64 Bit Para																
Ubuntu 64 Bit - Xen 64 Bit Para																
Ubuntu 32 Bit - Xen 64 Bit Voll																
Ubuntu 64 Bit - Xen 64 Bit Voll																
Win2K3 32 Bit - Xen 64 Bit Voll	18,68	18,85	18,44	18,72	18,85	18,94	18,77	7,10	7,10	7,09	3,11	3,11	3,04	3,03	3,02	3,02
Win2K3 64 Bit - Xen 64 Bit Voll	20,41	20,56	20,16	20,46	20,61	20,66	20,51	7,76	7,76	7,74	3,40	3,39	3,33	3,31	3,31	3,30
Ubuntu 32 Bit - OpenVZ 32 Bit																
Ubuntu 64 Bit - OpenVZ 64 Bit																
Win2K3 32 Bit - Virtuozzo 32 Bit																
Win2K3 64 Bit - Virtuozzo 64 Bit																
Ubuntu 32 Bit - HyperV 64	13,03	13,59	13,29	13,32	13,75	13,56	13,80	6,63	6,76	5,92	2,88	2,87	2,81	2,82	2,82	2,82
Ubuntu 64 Bit - HyperV 64	33,12	35,33	36,10	35,60	35,71	35,93	35,44	7,50	7,54	6,39	3,24	3,23	3,21	3,22	3,20	3,23
Win2K3 32 Bit - HyperV 64	17,70	18,77	18,12	17,89	18,56	18,81	18,48	7,06	6,94	7,01	3,04	3,02	2,99	2,98	2,96	2,96
Win2K3 64 Bit - HyperV 64	18,12	18,64	17,44	18,16	18,48	18,36	18,44	6,75	6,54	6,08	2,91	3,03	2,98	2,95	2,96	2,95
Ubuntu 32 Bit - ESXi 3.5 32 Bit	13,68	13,97	13,78	13,92	14,01	13,86	13,98	6,98	6,98	6,86	2,90	2,88	2,98	2,96	2,95	2,96
Ubuntu 64 Bit - ESXi 3.5 32 Bit	34,23	36,17	32,62	30,38	35,80	36,42	36,02	7,66	7,29	6,46	3,27	3,28	3,28	3,27	3,27	3,28
Win2K3 32 Bit - ESXi 3.5 32 Bit	18,56	19,07	18,24	18,56	18,68	18,81	18,56	7,05	6,97	6,92	2,98	3,02	2,99	2,91	2,97	2,98
Win2K3 64 Bit - ESXi 3.5 32 Bit	18,72	16,71	17,12	18,52	18,40	18,68	17,81	7,05	6,98	6,89	2,97	3,02	2,97	2,97	2,97	2,95

Abbildung 4.9: Messwerte RAMspeed sequentielles Lesen (AMD-V, paravirt. Treiber)



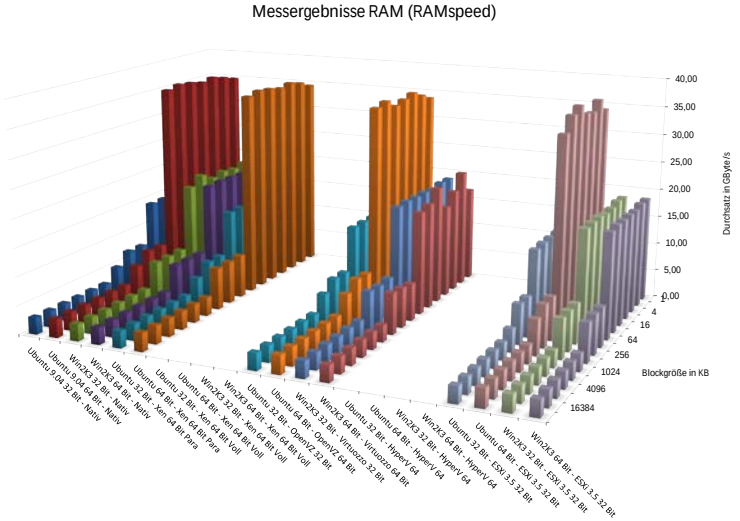
	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
Ubuntu 9.04 32 Bit - Nativ	14,10	14,26	14,04	14,20	14,29	14,33	14,28	7,12	7,12	5,12	3,07	3,08	3,03	3,02	3,01	3,01
Ubuntu 9.04 64 Bit - Nativ	34,86	35,60	36,24	35,86	36,38	36,64	36,20	7,65	7,71	5,91	3,34	3,34	3,32	3,32	3,31	3,31
Win2K3 32 Bit - Nativ	19,42	18,89	18,44	18,77	18,28	19,65	18,20	7,23	7,13	7,13	3,10	3,13	3,06	3,05	3,04	3,04
Win2K3 64 Bit - Nativ	18,72	18,89	18,44	18,77	18,89	18,98	18,85	7,13	7,13	7,12	3,11	3,11	3,05	3,04	3,03	3,03
Ubuntu 32 Bit - Xen 64 Bit Para																
Ubuntu 64 Bit - Xen 64 Bit Para																
Ubuntu 32 Bit - Xen 64 Bit Voll	13,82	14,25	14,04	14,19	14,27	14,31	14,27	7,11	7,11	5,26	3,08	3,06	3,02	3,01	3,01	3,01
Ubuntu 64 Bit - Xen 64 Bit Voll	31,48	36,03	36,68	36,28	36,81	37,08	36,63	7,78	7,79	5,81	3,38	3,38	3,36	3,34	3,34	3,33
Win2K3 32 Bit - Xen 64 Bit Voll	18,68	18,85	18,44	18,56	18,85	18,94	18,68	7,10	7,09	7,03	3,11	3,11	3,04	3,03	3,01	3,02
Win2K3 64 Bit - Xen 64 Bit Voll	20,31	20,51	20,07	20,36	20,46	20,61	20,41	7,73	7,71	7,71	3,38	3,39	3,32	3,29	3,28	3,27
Ubuntu 32 Bit - OpenVZ 32 Bit																
Ubuntu 64 Bit - OpenVZ 64 Bit																
Win2K3 32 Bit - Virtuozzo 32 Bit																
Win2K3 64 Bit - Virtuozzo 64 Bit																
Ubuntu 32 Bit - HyperV 64																
Ubuntu 64 Bit - HyperV 64																
Win2K3 32 Bit - HyperV 64																
Win2K3 64 Bit - HyperV 64																
Ubuntu 32 Bit - ESXi 3.5 32 Bit	13,92	14,21	14,01	14,14	14,25	14,26	14,21	7,09	7,09	5,43	3,04	3,01	2,97	2,97	2,97	2,97
Ubuntu 64 Bit - ESXi 3.5 32 Bit	34,23	36,60	34,89	35,70	37,51	36,62	34,62	7,63	7,64	5,52	3,28	3,27	3,24	3,25	3,25	3,27
Win2K3 32 Bit - ESXi 3.5 32 Bit	18,56	18,72	18,32	18,60	18,81	18,85	18,72	7,06	7,05	7,00	3,05	3,03	2,99	2,99	2,99	2,98
Win2K3 64 Bit - ESXi 3.5 32 Bit	18,40	18,68	18,16	18,60	18,24	18,81	18,60	6,97	7,01	6,93	2,95	2,90	2,96	2,97	2,98	2,98

Abbildung 4.10: Messwerte RAMspeed sequentielles Lesen
(AMD-V, keine paravirt. Treiber)



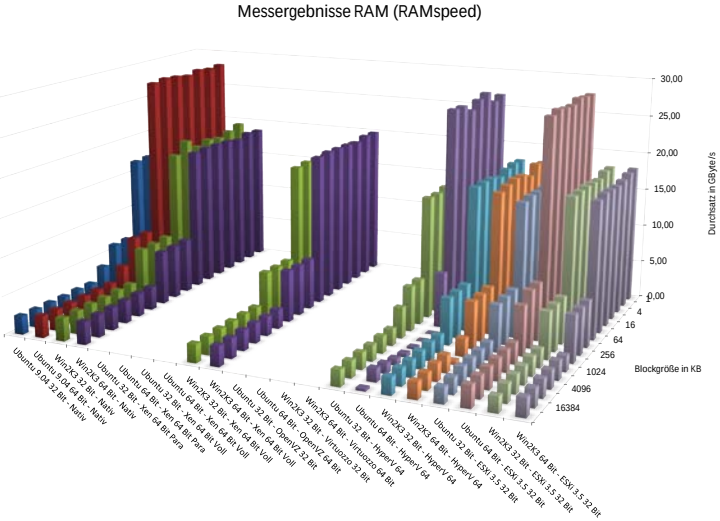
	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
Ubuntu 9.04 32 Bit - Nativ	14,10	14,26	14,04	14,20	14,29	14,33	14,28	7,12	7,12	5,12	3,07	3,08	3,03	3,02	3,01	3,01
Ubuntu 9.04 64 Bit - Nativ	34,86	35,60	36,24	35,86	36,38	36,64	36,20	7,65	7,71	5,91	3,34	3,34	3,32	3,32	3,31	3,31
Win2K3 32 Bit - Nativ	19,42	18,89	18,44	18,77	18,28	19,65	18,20	7,23	7,13	7,13	3,10	3,13	3,06	3,05	3,04	3,04
Win2K3 64 Bit - Nativ	18,72	18,89	18,44	18,77	18,89	18,98	18,85	7,13	7,13	7,12	3,11	3,11	3,05	3,04	3,03	3,03
Ubuntu 32 Bit - Xen 64 Bit Para																
Ubuntu 64 Bit - Xen 64 Bit Para																
Ubuntu 32 Bit - Xen 64 Bit Voll																
Ubuntu 64 Bit - Xen 64 Bit Voll																
Win2K3 32 Bit - Xen 64 Bit Voll																
Win2K3 64 Bit - Xen 64 Bit Voll																
Ubuntu 32 Bit - OpenVZ 32 Bit																
Ubuntu 64 Bit - OpenVZ 64 Bit																
Win2K3 32 Bit - Virtuozzo 32 Bit																
Win2K3 64 Bit - Virtuozzo 64 Bit																
Ubuntu 32 Bit - HyperV 64																
Ubuntu 64 Bit - HyperV 64																
Win2K3 32 Bit - HyperV 64																
Win2K3 64 Bit - HyperV 64																
Ubuntu 32 Bit - ESXi 3.5 32 Bit	13,90	14,20	13,97	14,13	14,21	14,24	14,19	7,01	7,07	4,69	3,01	2,99	2,97	2,94	2,95	2,93
Ubuntu 64 Bit - ESXi 3.5 32 Bit	34,12	36,69	34,59	37,04	34,85	35,99	37,27	7,53	7,56	7,32	3,28	3,16	3,23	3,26	3,26	3,25
Win2K3 32 Bit - ESXi 3.5 32 Bit	18,56	18,77	18,36	18,24	18,77	18,77	18,68	7,07	7,03	6,90	3,04	3,02	2,99	3,00	2,96	2,99
Win2K3 64 Bit - ESXi 3.5 32 Bit	18,56	18,64	18,89	19,02	18,64	18,72	18,56	7,04	7,00	6,51	2,98	3,02	2,97	2,97	2,97	2,97

Abbildung 4.11: Messwerte RAMspeed sequentielles Lesen
(kein AMD-V, paravirt. Treiber)



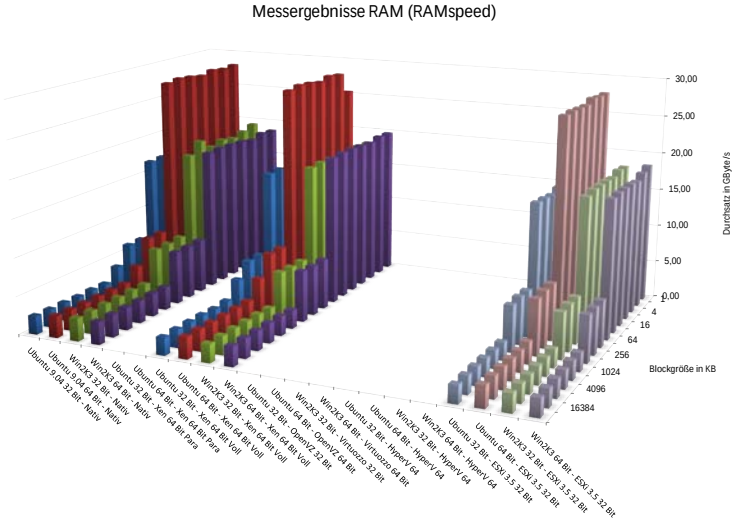
	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
Ubuntu 9.04 32 Bit - Nativ	14,10	14,26	14,04	14,20	14,29	14,33	14,28	7,12	7,12	5,12	3,07	3,08	3,03	3,02	3,01	3,01
Ubuntu 9.04 64 Bit - Nativ	34,86	35,60	36,24	35,86	36,38	36,64	36,20	7,65	7,71	5,91	3,34	3,34	3,32	3,32	3,31	3,31
Win2K3 32 Bit - Nativ	19,42	18,89	18,44	18,77	18,28	19,65	18,20	7,23	7,13	7,13	3,10	3,13	3,06	3,05	3,04	3,04
Win2K3 64 Bit - Nativ	18,72	18,89	18,44	18,77	18,89	18,98	18,85	7,13	7,13	7,12	3,11	3,11	3,05	3,04	3,03	3,03
Ubuntu 32 Bit - Xen 64 Bit Para	14,01	14,26	14,04	14,20	14,28	14,32	14,27	7,11	7,12	5,21	3,09	3,08	3,04	3,03	3,03	3,02
Ubuntu 64 Bit - Xen 64 Bit Para	34,56	35,60	36,24	35,86	36,38	36,64	36,19	7,71	7,61	7,52	3,33	3,33	3,31	3,30	3,30	3,30
Ubuntu 32 Bit - Xen 64 Bit Voll																
Ubuntu 64 Bit - Xen 64 Bit Voll																
Win2K3 32 Bit - Xen 64 Bit Voll																
Win2K3 64 Bit - Xen 64 Bit Voll																
Ubuntu 32 Bit - OpenVZ 32 Bit	14,10	14,26	14,04	14,20	14,28	14,33	14,28	7,12	7,12	5,54	3,09	3,08	3,03	3,01	3,01	3,01
Ubuntu 64 Bit - OpenVZ 64 Bit	33,89	35,00	36,08	35,55	35,02	36,48	35,97	7,41	7,61	6,07	3,34	3,33	3,31	3,30	3,32	3,32
Win2K3 32 Bit - Virtuozzo 32 Bit	18,72	18,85	18,48	18,77	18,77	18,94	18,81	7,11	6,89	7,10	3,09	3,09	3,01	3,03	3,03	2,99
Win2K3 64 Bit - Virtuozzo 64 Bit	16,95	21,02	18,44	16,42	20,76	18,77	18,24	6,65	7,06	7,37	2,97	3,05	3,02	2,97	2,97	2,95
Ubuntu 32 Bit - HyperV 64																
Ubuntu 64 Bit - HyperV 64																
Win2K3 32 Bit - HyperV 64																
Win2K3 64 Bit - HyperV 64																
Ubuntu 32 Bit - ESXi 3.5 32 Bit	13,92	14,19	13,96	14,14	14,21	14,26	14,19	7,08	7,07	4,06	3,04	3,00	2,97	2,95	2,95	2,96
Ubuntu 64 Bit - ESXi 3.5 32 Bit	34,25	36,57	35,03	35,52	37,54	36,61	34,14	7,66	7,57	6,31	3,27	3,28	3,28	3,28	3,25	3,28
Win2K3 32 Bit - ESXi 3.5 32 Bit	18,52	18,77	18,36	18,24	18,77	18,77	18,68	7,04	7,03	7,01	2,97	3,02	2,99	2,99	2,97	2,99
Win2K3 64 Bit - ESXi 3.5 32 Bit	18,60	18,68	18,36	18,60	18,60	18,85	18,60	7,06	7,06	6,93	3,04	3,01	2,98	2,96	2,96	2,91

Abbildung 4.12: Messwerte RAMspeed sequentielles Lesen
(kein AMD-V, keine paravirt. Treiber)



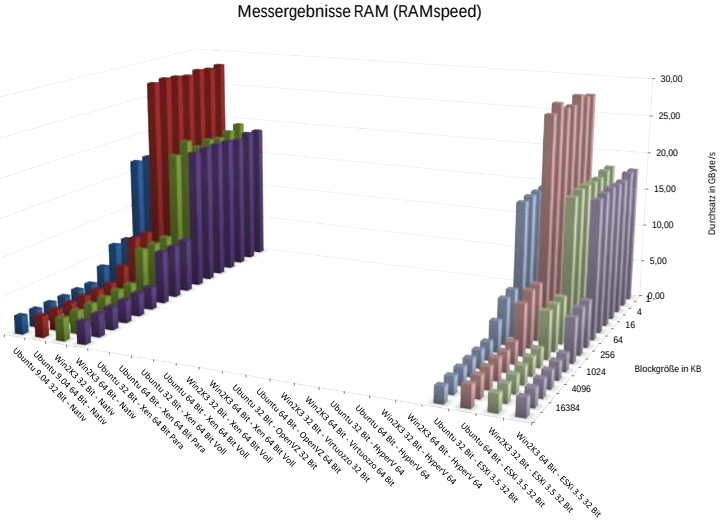
	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
Ubuntu 9.04 32 Bit - Nativ	16,62	16,79	16,51	16,73	16,85	16,90	16,81	6,27	6,28	3,96	2,36	2,40	2,34	2,39	2,40	2,45
Ubuntu 9.04 64 Bit - Nativ	28,20	28,04	28,28	27,89	28,20	28,36	28,12	7,50	7,50	4,37	2,77	2,78	2,79	2,79	2,82	2,89
Win2K3 32 Bit - Nativ	19,42	18,89	18,44	18,77	18,28	19,65	18,20	7,23	7,13	7,13	3,10	3,13	3,06	3,05	3,04	3,04
Win2K3 64 Bit - Nativ	18,72	18,89	18,44	18,77	18,89	18,98	18,85	7,13	7,13	7,12	3,11	3,11	3,05	3,04	3,03	3,03
Ubuntu 32 Bit - Xen 64 Bit Para																
Ubuntu 64 Bit - Xen 64 Bit Para																
Ubuntu 32 Bit - Xen 64 Bit Voll																
Ubuntu 64 Bit - Xen 64 Bit Voll																
Win2K3 32 Bit - Xen 64 Bit Voll	18,24	16,35	18,00	18,24	18,32	18,40	18,24	6,17	6,18	6,26	2,44	2,47	2,46	2,45	2,45	2,44
Win2K3 64 Bit - Xen 64 Bit Voll	19,93	20,02	19,65	19,88	19,97	20,07	19,93	6,75	6,74	6,86	2,58	2,55	2,65	2,69	2,66	2,67
Ubuntu 32 Bit - OpenVZ 32 Bit																
Ubuntu 64 Bit - OpenVZ 64 Bit																
Win2K3 32 Bit - Virtuozzo 32 Bit																
Win2K3 64 Bit - Virtuozzo 64 Bit																
Ubuntu 32 Bit - HyperV 64	15,51	15,47	15,91	15,83	16,13	15,89	16,02	5,93	5,90	4,05	2,19	2,21	2,11	2,18	2,18	2,15
Ubuntu 64 Bit - HyperV 64	26,75	26,31	27,94	27,49	26,59	27,61	27,73	7,22	0,09	0,10	0,30	0,51	0,87	1,30	1,72	0,36
Win2K3 32 Bit - HyperV 64	17,85	18,04	17,85	17,37	17,89	18,04	18,04	6,09	6,10	6,14	2,31	2,36	2,27	2,16	2,26	2,25
Win2K3 64 Bit - HyperV 64	17,85	18,24	17,15	17,92	18,12	17,92	17,59	6,14	6,09	6,09	2,34	0,50	2,25	2,28	2,19	2,20
Ubuntu 32 Bit - ESXi 3.5 32 Bit	16,35	16,74	16,43	16,68	16,68	16,85	16,70	6,24	6,18	6,13	2,32	2,32	2,36	2,38	2,12	2,03
Ubuntu 64 Bit - ESXi 3.5 32 Bit	27,71	27,87	28,14	27,71	27,87	28,19	27,87	7,51	7,48	6,43	2,73	2,79	2,81	2,81	2,84	2,82
Win2K3 32 Bit - ESXi 3.5 32 Bit	18,00	18,12	17,62	17,85	18,04	18,08	18,08	6,11	6,11	6,12	2,41	2,46	2,37	2,39	2,38	2,03
Win2K3 64 Bit - ESXi 3.5 32 Bit	18,08	18,08	17,81	17,70	17,92	18,16	17,89	6,09	6,10	6,16	2,06	2,45	2,38	2,39	2,35	2,39

Abbildung 4.13: Messwerte RAMspeed sequentielles Schreiben (AMD-V, paravirt. Treiber)



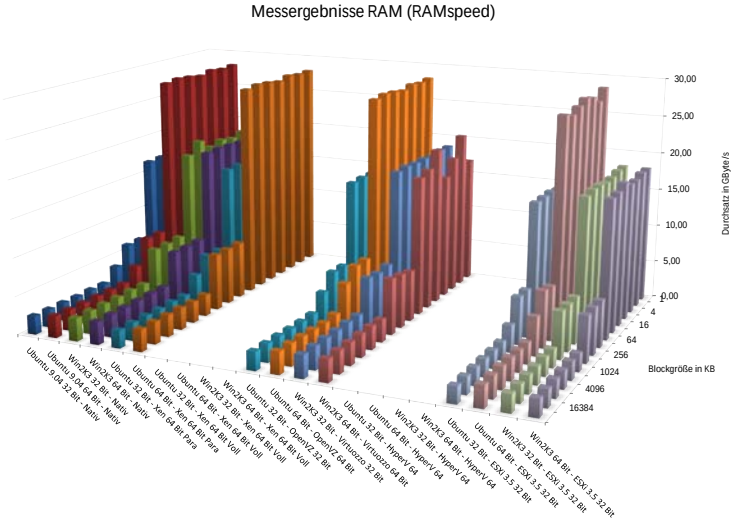
	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
Ubuntu 9.04 32 Bit - Nativ	16,62	16,79	16,51	16,73	16,85	16,90	16,81	6,27	6,28	3,96	2,36	2,40	2,34	2,39	2,40	2,45
Ubuntu 9.04 64 Bit - Nativ	28,20	28,04	28,28	27,89	28,20	28,36	28,12	7,50	7,50	4,37	2,77	2,78	2,79	2,79	2,82	2,89
Win2K3 32 Bit - Nativ	19,42	18,89	18,44	18,77	18,28	19,65	18,20	7,23	7,13	7,13	3,10	3,13	3,06	3,05	3,04	3,04
Win2K3 64 Bit - Nativ	18,72	18,89	18,44	18,77	18,89	18,98	18,85	7,13	7,13	7,12	3,11	3,11	3,05	3,04	3,03	3,03
Ubuntu 32 Bit - Xen 64 Bit Para																
Ubuntu 64 Bit - Xen 64 Bit Para																
Ubuntu 32 Bit - Xen 64 Bit Voll	16,17	16,77	16,50	16,72	16,83	16,88	16,79	6,27	6,26	4,48	2,44	2,30	2,41	2,43	2,41	2,35
Ubuntu 64 Bit - Xen 64 Bit Voll	25,45	28,38	28,62	28,22	28,54	28,70	28,41	7,58	7,58	5,02	2,81	2,81	2,79	2,81	2,86	2,88
Win2K3 32 Bit - Xen 64 Bit Voll	18,24	18,36	17,89	17,96	18,32	18,36	18,20	6,17	6,16	6,28	2,32	2,43	2,45	2,46	2,47	2,43
Win2K3 64 Bit - Xen 64 Bit Voll	19,83	19,93	19,60	19,78	19,93	19,97	19,78	6,72	6,69	6,82	2,47	2,61	2,63	2,64	2,62	2,64
Ubuntu 32 Bit - OpenVZ 32 Bit																
Ubuntu 64 Bit - OpenVZ 64 Bit																
Win2K3 32 Bit - Virtuozzo 32 Bit																
Win2K3 64 Bit - Virtuozzo 64 Bit																
Ubuntu 32 Bit - HyperV 64																
Ubuntu 64 Bit - HyperV 64																
Win2K3 32 Bit - HyperV 64																
Win2K3 64 Bit - HyperV 64																
Ubuntu 32 Bit - ESXi 3.5 32 Bit	16,36	16,71	16,43	16,65	16,31	16,56	16,71	6,14	6,25	6,08	2,32	2,34	2,36	2,39	2,38	2,32
Ubuntu 64 Bit - ESXi 3.5 32 Bit	27,84	27,91	28,16	27,70	27,85	27,99	27,96	7,45	7,44	7,30	2,85	2,73	2,76	2,80	2,83	2,82
Win2K3 32 Bit - ESXi 3.5 32 Bit	18,08	18,28	17,89	18,12	18,24	18,32	18,08	6,15	6,15	6,04	2,41	2,36	2,40	2,37	2,40	2,38
Win2K3 64 Bit - ESXi 3.5 32 Bit	18,52	18,16	17,66	18,04	18,12	18,24	18,12	6,13	6,13	6,16	2,40	2,43	2,38	2,38	2,36	2,34

Abbildung 4.14: Messwerte RAMspeed sequentielles Schreiben (AMD-V, keine paravirt. Treiber)



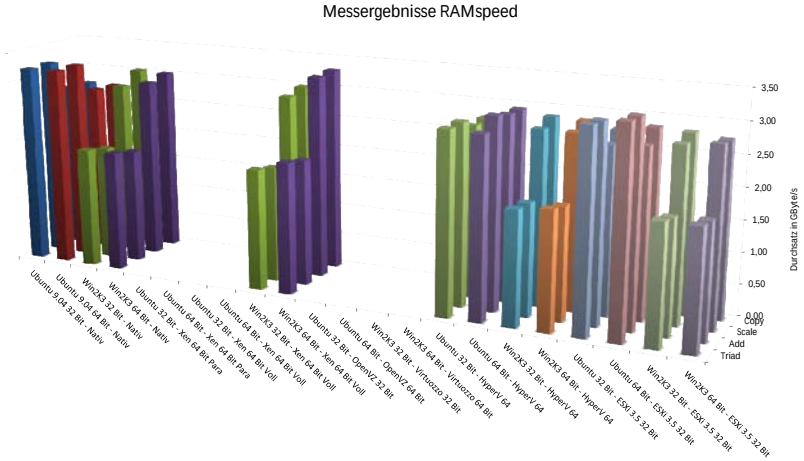
	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
Ubuntu 9.04 32 Bit - Nativ	16,62	16,79	16,51	16,73	16,85	16,90	16,81	6,27	6,28	3,96	2,36	2,40	2,34	2,39	2,40	2,45
Ubuntu 9.04 64 Bit - Nativ	28,20	28,04	28,28	27,89	28,20	28,36	28,12	7,50	7,50	4,37	2,77	2,78	2,79	2,79	2,82	2,89
Win2K3 32 Bit - Nativ	19,42	18,89	18,44	18,77	18,28	19,65	18,20	7,23	7,13	7,13	3,10	3,13	3,06	3,05	3,04	3,04
Win2K3 64 Bit - Nativ	18,72	18,89	18,44	18,77	18,89	18,98	18,85	7,13	7,13	7,12	3,11	3,11	3,05	3,04	3,03	3,03
Ubuntu 32 Bit - Xen 64 Bit Para																
Ubuntu 64 Bit - Xen 64 Bit Para																
Ubuntu 32 Bit - Xen 64 Bit Voll																
Ubuntu 64 Bit - Xen 64 Bit Voll																
Win2K3 32 Bit - Xen 64 Bit Voll																
Win2K3 64 Bit - Xen 64 Bit Voll																
Ubuntu 32 Bit - OpenVZ 32 Bit																
Ubuntu 64 Bit - OpenVZ 64 Bit																
Win2K3 32 Bit - Virtuozzo 32 Bit																
Win2K3 64 Bit - Virtuozzo 64 Bit																
Ubuntu 32 Bit - HyperV 64																
Ubuntu 64 Bit - HyperV 64																
Win2K3 32 Bit - HyperV 64																
Win2K3 64 Bit - HyperV 64																
Ubuntu 32 Bit - ESXi 3.5 32 Bit	14,21	16,71	16,44	16,68	16,78	16,85	16,72	6,25	6,02	4,05	2,36	2,34	2,31	2,27	2,25	2,17
Ubuntu 64 Bit - ESXi 3.5 32 Bit	27,64	27,85	28,54	27,66	27,82	28,97	28,06	7,39	7,43	6,62	2,81	2,64	2,84	2,79	2,86	2,83
Win2K3 32 Bit - ESXi 3.5 32 Bit	18,08	18,20	17,81	17,89	18,20	18,28	17,96	6,15	6,13	6,16	2,43	2,38	2,36	2,33	2,36	2,32
Win2K3 64 Bit - ESXi 3.5 32 Bit	17,96	18,20	17,73	17,92	18,16	18,12	18,00	6,10	6,10	5,80	2,30	2,36	2,29	2,34	2,37	2,36

Abbildung 4.15: Messwerte RAMspeed sequentielles Schreiben
(kein AMD-V, paravirt. Treiber)



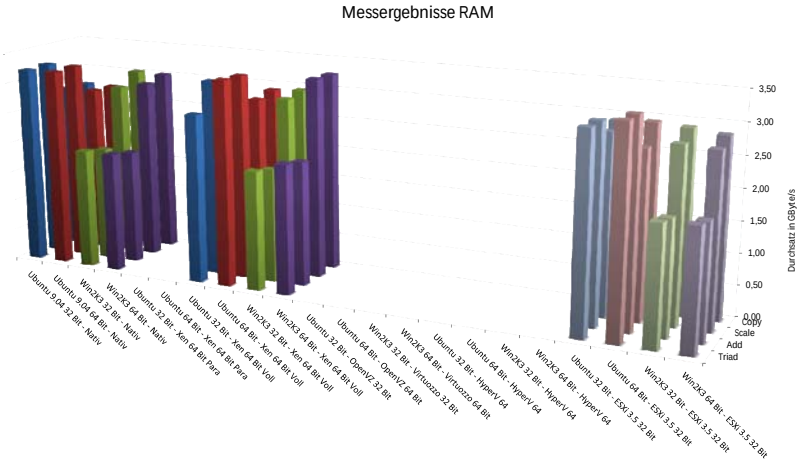
	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
Ubuntu 9.04 32 Bit - Nativ	16,62	16,79	16,51	16,73	16,85	16,90	16,81	6,27	6,28	3,96	2,36	2,40	2,34	2,39	2,40	2,45
Ubuntu 9.04 64 Bit - Nativ	28,20	28,04	28,28	27,89	28,20	28,36	28,12	7,50	7,50	4,37	2,77	2,78	2,79	2,79	2,82	2,89
Win2K3 32 Bit - Nativ	19,42	18,89	18,44	18,77	18,28	19,65	18,20	7,23	7,13	7,13	3,10	3,13	3,06	3,05	3,04	3,04
Win2K3 64 Bit - Nativ	18,72	18,89	18,44	18,77	18,89	18,98	18,85	7,13	7,13	7,12	3,11	3,11	3,05	3,04	3,03	3,03
Ubuntu 32 Bit - Xen 64 Bit Para	16,53	16,78	16,51	16,73	16,84	16,89	16,79	6,27	6,27	4,34	2,34	2,38	2,33	2,38	2,37	2,35
Ubuntu 64 Bit - Xen 64 Bit Para	28,09	28,04	28,28	27,89	28,20	28,36	28,12	7,50	7,49	7,49	2,87	2,80	2,89	2,92	2,94	2,93
Ubuntu 32 Bit - Xen 64 Bit Voll																
Ubuntu 64 Bit - Xen 64 Bit Voll																
Win2K3 32 Bit - Xen 64 Bit Voll																
Win2K3 64 Bit - Xen 64 Bit Voll																
Ubuntu 32 Bit - OpenVZ 32 Bit	16,36	16,78	16,49	16,73	16,84	16,89	16,79	6,27	6,28	4,37	2,39	2,34	2,36	2,44	2,44	2,44
Ubuntu 64 Bit - OpenVZ 64 Bit	28,20	28,05	28,28	27,88	28,20	28,36	28,12	7,51	7,50	5,99	2,79	2,59	2,69	2,80	2,89	2,92
Win2K3 32 Bit - Virtuozzo 32 Bit	18,72	18,85	18,48	18,77	18,77	18,94	18,81	7,11	6,89	7,10	3,09	3,09	3,01	3,03	3,03	2,99
Win2K3 64 Bit - Virtuozzo 64 Bit	16,95	21,02	18,44	16,42	20,76	18,77	18,24	6,65	7,06	7,37	2,97	3,05	3,02	2,97	2,97	2,95
Ubuntu 32 Bit - HyperV 64																
Ubuntu 64 Bit - HyperV 64																
Win2K3 32 Bit - HyperV 64																
Win2K3 64 Bit - HyperV 64																
Ubuntu 32 Bit - ESXi 3.5 32 Bit	16,37	16,65	16,45	16,44	16,77	16,73	16,71	6,23	6,23	3,51	2,34	2,29	2,28	2,26	2,24	2,21
Ubuntu 64 Bit - ESXi 3.5 32 Bit	28,58	27,19	28,20	28,62	28,04	27,43	28,00	7,08	7,44	5,04	2,73	2,78	2,87	2,88	2,87	2,83
Win2K3 32 Bit - ESXi 3.5 32 Bit	18,16	18,20	17,92	17,85	18,24	18,32	18,08	6,14	6,11	6,18	2,37	2,38	2,32	2,32	2,31	2,31
Win2K3 64 Bit - ESXi 3.5 32 Bit	18,12	18,20	17,85	17,96	18,81	18,24	18,12	6,12	6,14	6,11	2,31	2,36	2,29	2,30	2,29	2,27

Abbildung 4.16: Messwerte RAMspeed sequentielles Schreiben
(kein AMD-V, keine paravirt. Treiber)



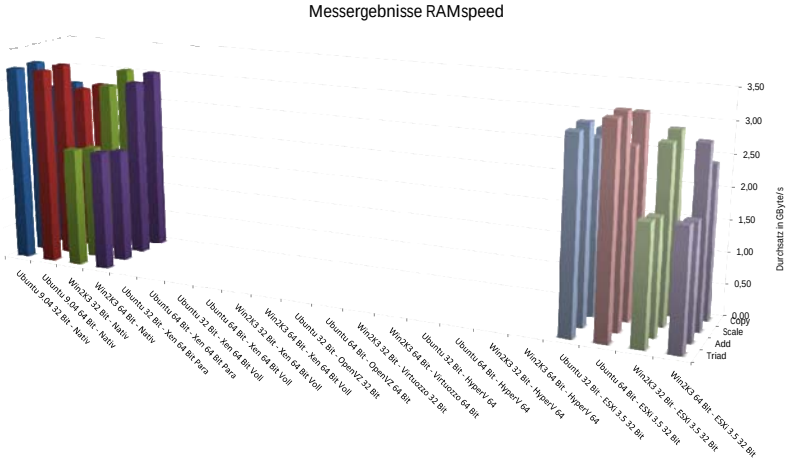
	Copy	Scale	Add	Triad
Ubuntu 9.04 32 Bit - Nativ	2,72	2,74	3,24	3,22
Ubuntu 9.04 64 Bit - Nativ	2,72	2,74	3,24	3,22
Win2K3 32 Bit - Nativ	3,02	2,84	1,87	1,95
Win2K3 64 Bit - Nativ	3,01	2,93	1,86	1,94
Ubuntu 32 Bit - Xen 64 Bit Para				
Ubuntu 64 Bit - Xen 64 Bit Para				
Ubuntu 32 Bit - Xen 64 Bit Voll				
Ubuntu 64 Bit - Xen 64 Bit Voll				
Win2K3 32 Bit - Xen 64 Bit Voll	2,99	2,92	1,87	1,95
Win2K3 64 Bit - Xen 64 Bit Voll	3,30	3,27	2,04	2,11
Ubuntu 32 Bit - OpenVZ 32 Bit				
Ubuntu 64 Bit - OpenVZ 64 Bit				
Win2K3 32 Bit - Virtuozzo 32 Bit				
Win2K3 64 Bit - Virtuozzo 64 Bit				
Ubuntu 32 Bit - HyperV 64	2,75	2,74	2,90	2,89
Ubuntu 64 Bit - HyperV 64	2,94	2,95	3,04	2,87
Win2K3 32 Bit - HyperV 64	2,86	2,78	1,77	1,82
Win2K3 64 Bit - HyperV 64	2,85	2,78	1,79	1,88
Ubuntu 32 Bit - ESXi 3.5 32 Bit	2,76	2,65	3,10	3,15
Ubuntu 64 Bit - ESXi 3.5 32 Bit	2,85	2,68	3,22	3,24
Win2K3 32 Bit - ESXi 3.5 32 Bit	2,82	2,75	1,81	1,89
Win2K3 64 Bit - ESXi 3.5 32 Bit	2,76	2,81	1,82	1,89

Abbildung 4.17: Messwerte RAMspeed nicht-sequentielle Zugriffe (AMD-V, paravirt. Treiber)



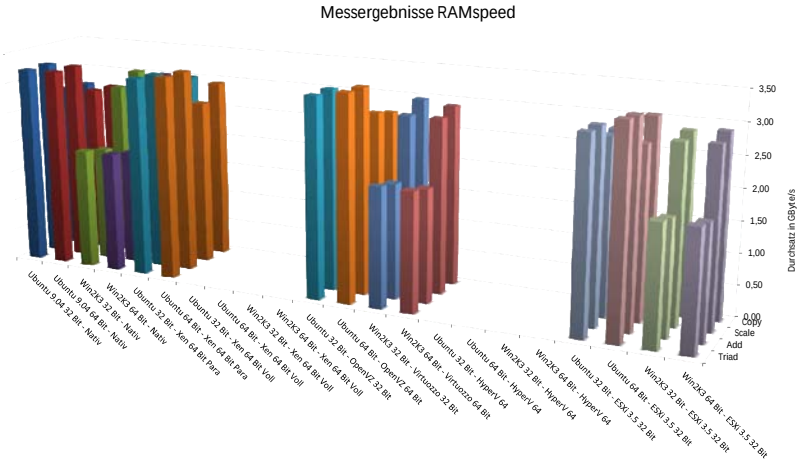
	Copy	Scale	Add	Triad
Ubuntu 9.04 32 Bit - Nativ	2,72	2,74	3,24	3,22
Ubuntu 9.04 64 Bit - Nativ	2,72	2,74	3,24	3,22
Win2K3 32 Bit - Nativ	3,02	2,84	1,87	1,95
Win2K3 64 Bit - Nativ	3,01	2,93	1,86	1,94
Ubuntu 32 Bit - Xen 64 Bit Para				
Ubuntu 64 Bit - Xen 64 Bit Para				
Ubuntu 32 Bit - Xen 64 Bit Voll	2,58	2,48	3,20	2,74
Ubuntu 64 Bit - Xen 64 Bit Voll	2,91	2,85	3,32	3,34
Win2K3 32 Bit - Xen 64 Bit Voll	2,95	2,90	1,85	1,94
Win2K3 64 Bit - Xen 64 Bit Voll	3,25	3,26	2,03	2,10
Ubuntu 32 Bit - OpenVZ 32 Bit				
Ubuntu 64 Bit - OpenVZ 64 Bit				
Win2K3 32 Bit - Virtuozzo 32 Bit				
Win2K3 64 Bit - Virtuozzo 64 Bit				
Ubuntu 32 Bit - HyperV 64				
Ubuntu 64 Bit - HyperV 64				
Win2K3 32 Bit - HyperV 64				
Win2K3 64 Bit - HyperV 64				
Ubuntu 32 Bit - ESXi 3.5 32 Bit	2,92	2,86	3,12	3,13
Ubuntu 64 Bit - ESXi 3.5 32 Bit	2,94	2,66	3,26	3,27
Win2K3 32 Bit - ESXi 3.5 32 Bit	2,91	2,77	1,81	1,89
Win2K3 64 Bit - ESXi 3.5 32 Bit	2,84	2,74	1,84	1,91

Abbildung 4.18: Messwerte RAMspeed nicht-sequentielle Zugriffe (AMD-V, keine paravirt. Treiber)



	Copy	Scale	Add	Triad
Ubuntu 9.04 32 Bit - Nativ	2,72	2,74	3,24	3,22
Ubuntu 9.04 64 Bit - Nativ	2,72	2,74	3,24	3,22
Win2K3 32 Bit - Nativ	3,02	2,84	1,87	1,95
Win2K3 64 Bit - Nativ	3,01	2,93	1,86	1,94
Ubuntu 32 Bit - Xen 64 Bit Para				
Ubuntu 64 Bit - Xen 64 Bit Para				
Ubuntu 32 Bit - Xen 64 Bit Voll				
Ubuntu 64 Bit - Xen 64 Bit Voll				
Win2K3 32 Bit - Xen 64 Bit Voll				
Win2K3 64 Bit - Xen 64 Bit Voll				
Ubuntu 32 Bit - OpenVZ 32 Bit				
Ubuntu 64 Bit - OpenVZ 64 Bit				
Win2K3 32 Bit - Virtuozzo 32 Bit				
Win2K3 64 Bit - Virtuozzo 64 Bit				
Ubuntu 32 Bit - HyperV 64				
Ubuntu 64 Bit - HyperV 64				
Win2K3 32 Bit - HyperV 64				
Win2K3 64 Bit - HyperV 64				
Ubuntu 32 Bit - ESXi 3.5 32 Bit	2,78	2,75	3,08	3,05
Ubuntu 64 Bit - ESXi 3.5 32 Bit	3,08	2,68	3,29	3,29
Win2K3 32 Bit - ESXi 3.5 32 Bit	2,86	2,78	1,81	1,88
Win2K3 64 Bit - ESXi 3.5 32 Bit	2,40	2,83	1,82	1,90

Abbildung 4.19: Messwerte RAMspeed nicht-sequentielle Zugriffe (kein AMD-V, paravirt. Treiber)

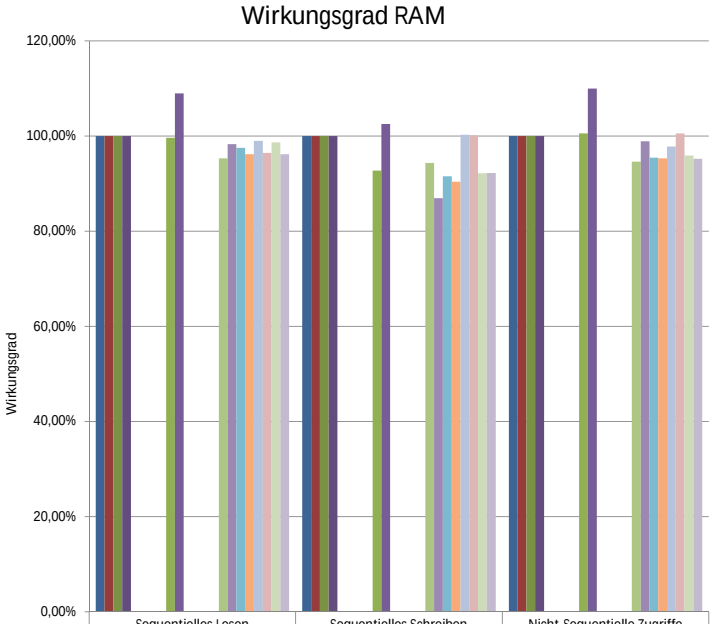


	Copy	Scale	Add	Triad
Ubuntu 9.04 32 Bit - Nativ	2,72	2,74	3,24	3,22
Ubuntu 9.04 64 Bit - Nativ	2,72	2,74	3,24	3,22
Win2K3 32 Bit - Nativ	3,02	2,84	1,87	1,95
Win2K3 64 Bit - Nativ	3,01	2,93	1,86	1,94
Ubuntu 32 Bit - Xen 64 Bit Para	3,00	2,96	3,22	3,24
Ubuntu 64 Bit - Xen 64 Bit Para	2,94	2,69	3,30	3,30
Ubuntu 32 Bit - Xen 64 Bit Voll				
Ubuntu 64 Bit - Xen 64 Bit Voll				
Win2K3 32 Bit - Xen 64 Bit Voll				
Win2K3 64 Bit - Xen 64 Bit Voll				
Ubuntu 32 Bit - OpenVZ 32 Bit	2,91	2,70	3,23	3,23
Ubuntu 64 Bit - OpenVZ 64 Bit	2,71	2,81	3,31	3,32
Win2K3 32 Bit - Virtuozzo 32 Bit	2,97	2,81	1,85	1,93
Win2K3 64 Bit - Virtuozzo 64 Bit	2,91	2,81	1,82	1,92
Ubuntu 32 Bit - HyperV 64				
Ubuntu 64 Bit - HyperV 64				
Win2K3 32 Bit - HyperV 64				
Win2K3 64 Bit - HyperV 64				
Ubuntu 32 Bit - ESXi 3.5 32 Bit	2,79	2,80	3,06	3,06
Ubuntu 64 Bit - ESXi 3.5 32 Bit	3,04	2,72	3,24	3,29
Win2K3 32 Bit - ESXi 3.5 32 Bit	2,86	2,81	1,82	1,90
Win2K3 64 Bit - ESXi 3.5 32 Bit	2,90	2,82	1,82	1,90

Abbildung 4.20: Messwerte RAMspeed nicht-sequentielle Zugriffe (kein AMD-V, keine paravirt. Treiber)

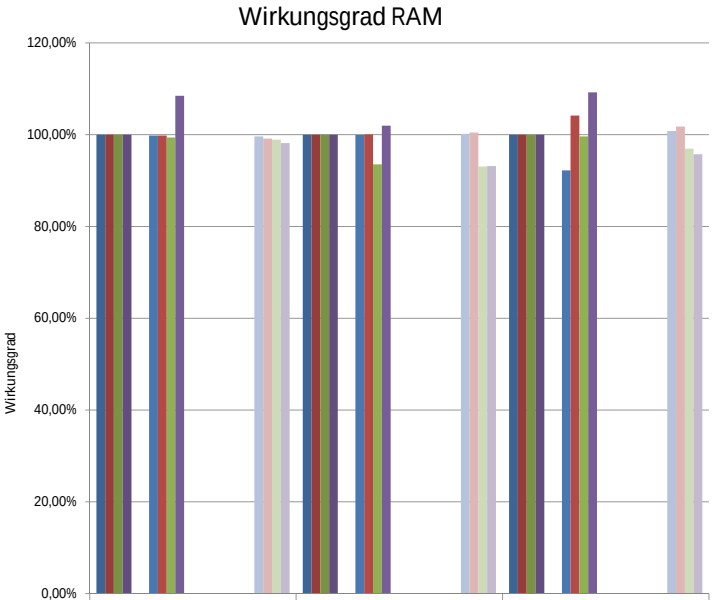
4.2.2.2 Ermittlung des Wirkungsgrades

Der Wirkungsgrad des Hauptspeicherzugriffs errechnet sich aus dem Quotienten des Durchsatzes in einer virtuellen Maschine und dem Durchsatz der assoziierten physischen Maschine. Da eine Wirkungsgradbetrachtung auf Basis von einzelnen Blockgrößen im Allgemeinen keinen Wert ergibt sondern den Sachverhalt in der Darstellung kompliziert, wurde als Durchsatz das Mittel über alle gemessenen Blockgrößen beziehungsweise über die Aktivitäten Copy, Scale, Add, und Triad herangezogen. Alternativ könnten auch Gruppen von Blockgrößen angelehnt an die Größen der vorhandenen Caches gebildet werden und der Wirkungsgrad über diese bestimmt werden. Da der Wirkungsgrad über diese Gruppen jedoch nur minimal von der Variante über alle Blockgrößen abweicht wurde aufgrund der einfacheren Darstellbarkeit darauf verzichtet. Der Wirkungsgrad des Zugriffes auf Hauptspeicher wird von den Abbildungen 4.21, 4.22, 4.23 und 4.24 dargestellt.



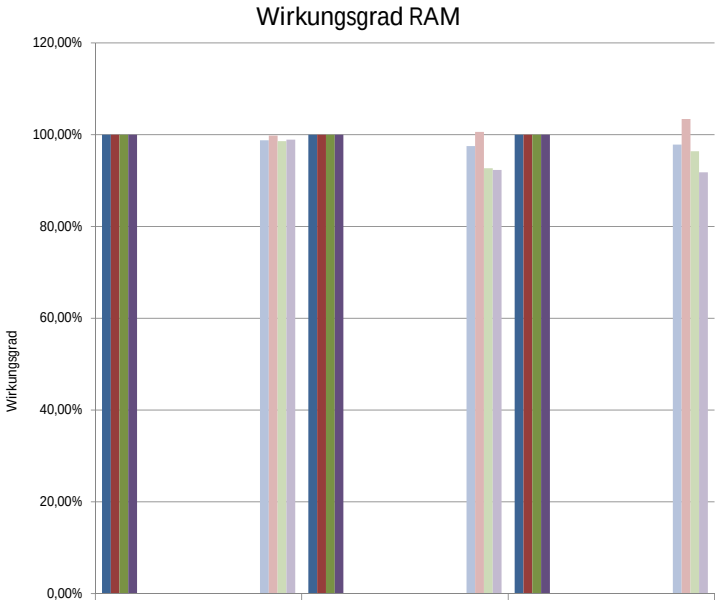
	Sequentielles Lesen	Sequentielles Schreiben	Nicht-Sequentielle Zugriffe
Ubuntu 9.04 32 Bit - Nativ	100,00%	100,00%	100,00%
Ubuntu 9.04 64 Bit - Nativ	100,00%	100,00%	100,00%
Win2K3 32 Bit - Nativ	100,00%	100,00%	100,00%
Win2K3 64 Bit - Nativ	100,00%	100,00%	100,00%
Ubuntu 32 Bit - Xen 64 Bit Para			
Ubuntu 64 Bit - Xen 64 Bit Para			
Ubuntu 32 Bit - Xen 64 Bit Voll			
Ubuntu 64 Bit - Xen 64 Bit Voll			
Win2K3 32 Bit - Xen 64 Bit Voll	99,62%	92,74%	100,53%
Win2K3 64 Bit - Xen 64 Bit Voll	108,98%	102,51%	109,97%
Ubuntu 32 Bit - OpenVZ 32 Bit			
Ubuntu 64 Bit - OpenVZ 64 Bit			
Win2K3 32 Bit - Virtuozzo 32 Bit			
Win2K3 64 Bit - Virtuozzo 64 Bit			
Ubuntu 32 Bit - HyperV 64	95,32%	94,32%	94,64%
Ubuntu 64 Bit - HyperV 64	98,29%	86,96%	98,89%
Win2K3 32 Bit - HyperV 64	97,51%	91,53%	95,45%
Win2K3 64 Bit - HyperV 64	96,19%	90,41%	95,31%
Ubuntu 32 Bit - ESXi 3.5 32 Bit	98,98%	100,29%	97,79%
Ubuntu 64 Bit - ESXi 3.5 32 Bit	96,48%	100,12%	100,55%
Win2K3 32 Bit - ESXi 3.5 32 Bit	98,67%	92,20%	95,92%
Win2K3 64 Bit - ESXi 3.5 32 Bit	96,18%	92,25%	95,21%

Abbildung 4.21: Wirkungsgrad RAM(AMD-V, paravirt. Treiber)



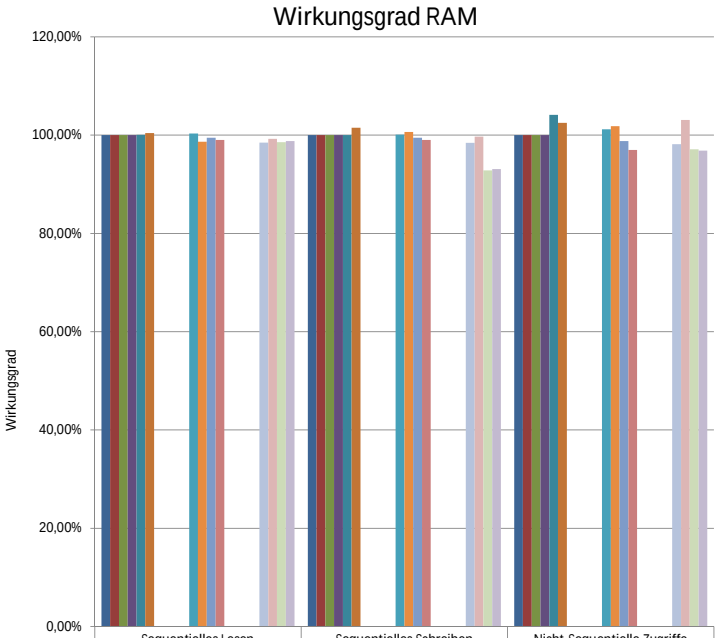
	Sequentielles Lesen	Sequentielles Schreiben	Nicht-Sequentielle Zugriffe
Ubuntu 9.04 32 Bit - Nativ	100,00%	100,00%	100,00%
Ubuntu 9.04 64 Bit - Nativ	100,00%	100,00%	100,00%
Win2K3 32 Bit - Nativ	100,00%	100,00%	100,00%
Win2K3 64 Bit - Nativ	100,00%	100,00%	100,00%
Ubuntu 32 Bit - Xen 64 Bit Para			
Ubuntu 64 Bit - Xen 64 Bit Para			
Ubuntu 32 Bit - Xen 64 Bit Voll	99,83%	99,95%	92,24%
Ubuntu 64 Bit - Xen 64 Bit Voll	99,83%	100,06%	104,16%
Win2K3 32 Bit - Xen 64 Bit Voll	99,41%	93,55%	99,62%
Win2K3 64 Bit - Xen 64 Bit Voll	108,48%	101,97%	109,21%
Ubuntu 32 Bit - OpenVZ 32 Bit			
Ubuntu 64 Bit - OpenVZ 64 Bit			
Win2K3 32 Bit - Virtuozzo 32 Bit			
Win2K3 64 Bit - Virtuozzo 64 Bit			
Ubuntu 32 Bit - HyperV 64			
Ubuntu 64 Bit - HyperV 64			
Win2K3 32 Bit - HyperV 64			
Win2K3 64 Bit - HyperV 64			
Ubuntu 32 Bit - ESXi 3.5 32 Bit	99,63%	100,17%	100,81%
Ubuntu 64 Bit - ESXi 3.5 32 Bit	99,15%	100,46%	101,78%
Win2K3 32 Bit - ESXi 3.5 32 Bit	98,93%	93,07%	96,94%
Win2K3 64 Bit - ESXi 3.5 32 Bit	98,16%	93,15%	95,75%

Abbildung 4.22: Wirkungsgrad RAM(AMD-V, keine paravirt. Treiber)



	Sequentielles Lesen	Sequentielles Schreiben	Nicht-Sequentielle Zugriffe
Ubuntu 9.04 32 Bit - Nativ	100,00%	100,00%	100,00%
Ubuntu 9.04 64 Bit - Nativ	100,00%	100,00%	100,00%
Win2K3 32 Bit - Nativ	100,00%	100,00%	100,00%
Win2K3 64 Bit - Nativ	100,00%	100,00%	100,00%
Ubuntu 32 Bit - Xen 64 Bit Para			
Ubuntu 64 Bit - Xen 64 Bit Para			
Ubuntu 32 Bit - Xen 64 Bit Voll			
Ubuntu 64 Bit - Xen 64 Bit Voll			
Win2K3 32 Bit - Xen 64 Bit Voll			
Win2K3 64 Bit - Xen 64 Bit Voll			
Ubuntu 32 Bit - OpenVZ 32 Bit			
Ubuntu 64 Bit - OpenVZ 64 Bit			
Win2K3 32 Bit - Virtuozzo 32 Bit			
Win2K3 64 Bit - Virtuozzo 64 Bit			
Ubuntu 32 Bit - HyperV 64			
Ubuntu 64 Bit - HyperV 64			
Win2K3 32 Bit - HyperV 64			
Win2K3 64 Bit - HyperV 64			
Ubuntu 32 Bit - ESXi 3.5 32 Bit	98,79%	97,54%	97,86%
Ubuntu 64 Bit - ESXi 3.5 32 Bit	99,80%	100,61%	103,43%
Win2K3 32 Bit - ESXi 3.5 32 Bit	98,59%	92,70%	96,42%
Win2K3 64 Bit - ESXi 3.5 32 Bit	98,93%	92,31%	91,79%

Abbildung 4.23: Wirkungsgrad RAM(kein AMD-V, paravirt. Treiber)



	Sequentielles Lesen	Sequentielles Schreiben	Nicht-Sequentielle Zugriffe
Ubuntu 9.04 32 Bit - Nativ	100,00%	100,00%	100,00%
Ubuntu 9.04 64 Bit - Nativ	100,00%	100,00%	100,00%
Win2K3 32 Bit - Nativ	100,00%	100,00%	100,00%
Win2K3 64 Bit - Nativ	100,00%	100,00%	100,00%
Ubuntu 32 Bit - Xen 64 Bit Para	100,04%	100,03%	104,14%
Ubuntu 64 Bit - Xen 64 Bit Para	100,40%	101,49%	102,52%
Ubuntu 32 Bit - Xen 64 Bit Voll			
Ubuntu 64 Bit - Xen 64 Bit Voll			
Win2K3 32 Bit - Xen 64 Bit Voll			
Win2K3 64 Bit - Xen 64 Bit Voll			
Ubuntu 32 Bit - OpenVZ 32 Bit	100,32%	100,11%	101,21%
Ubuntu 64 Bit - OpenVZ 64 Bit	98,64%	100,63%	101,83%
Win2K3 32 Bit - Virtuozzo 32 Bit	99,48%	99,48%	98,77%
Win2K3 64 Bit - Virtuozzo 64 Bit	99,01%	99,01%	97,00%
Ubuntu 32 Bit - HyperV 64			
Ubuntu 64 Bit - HyperV 64			
Win2K3 32 Bit - HyperV 64			
Win2K3 64 Bit - HyperV 64			
Ubuntu 32 Bit - ESXi 3.5 32 Bit	98,46%	98,41%	98,17%
Ubuntu 64 Bit - ESXi 3.5 32 Bit	99,26%	99,69%	103,09%
Win2K3 32 Bit - ESXi 3.5 32 Bit	98,58%	92,81%	97,10%
Win2K3 64 Bit - ESXi 3.5 32 Bit	98,77%	93,10%	96,83%

Abbildung 4.24: Wirkungsgrad RAM(kein AMD-V, keine paravirt. Treiber)

4.2.2.3 Auswertung

Zugriff auf Caches auch virtuell performant

Bei der Betrachtung der Grafiken mit den Messergebnissen (siehe Abbildungen 4.9, 4.10, 4.11, 4.12 und 4.13, 4.14, 4.15, 4.16) der sequentiellen Lese- und Schreibbenchmarks sticht als allererstes eine Stufung der Durchsatzraten bei 64KB und 512KB Blockgroße ins Auge. Diese Stufen treten sowohl bei nativen Systemen als auch virtuell auf und korrelieren mit den Cache-Größen des physischen Systems. Die Virtualisierung einzelner Systeme hat damit keinen grundsätzlichen Einfluss auf die Effektivität von Caches. In einigen Messreihen sind speziell im L1-Cache leichte Schwankungen im Durchsatz zu beobachten. Diese stammen von parallel ausgeführten Prozessen in der virtuellen oder physischen Maschine. Im virtuellen Fall kann die Schwankung auch von Kontextwechseln in den Hypervisor verursacht werden. Die Ursache für beide Phänomene liegt in der Verschmutzung des Caches durch parallel ausgeführte Prozesse, so dass in Einzelfällen Daten, die sich eigentlich noch im Cache befinden sollten, aber durch nebenläufige Prozesse von dort verdrängt wurden, wieder aus dem Arbeitsspeicher gelesen werden müssen.

Unterschiede bei 64-Bit Architekturen

Die höheren Transferraten im L1-Cache bei den 64 Bit Linux Systemen beruhen auf einer speziell für 64 Bit Architekturen angepassten RAMspeed Version, die 64-Bit Instruktionen verwendet und damit größere Blöcke in einer Instruktion lesen beziehungsweise schreiben kann. Für Windows existiert keine angepasste 64-Bit Version, so dass der Effekt hier nicht auftritt. Die Abbildungen 4.17, 4.18, 4.19, 4.20 zeigen das Verhalten von nicht-sequentiellen Zugriffen.

Hoher Wirkungsgrad für Speicherzugriffe

Betrachtet man die Wirkungsgrade über sequentielles Lesen und Schreiben sowie wahlfrei Zugriffe, so stellt man zunächst fest, dass der Wirkungsgrad sämtlicher Produkte und Techniken bis auf einzelne Ausnahmen größer als 95% im Lesen und 90% im Schreiben liegt. Alle Techniken eignen sich daher in der Praxis gut um performante Virtualisierung von Maschinen zu gewährleisten. Die Differenz im Wirkungsgrad von Lese- und Schreiboperationen lassen sich durch die eingesetzten Techniken zur Speicherabbildung erklären. Alle getesteten Virtualisierungsprodukte mit Focus auf Vollvirtualisierung setzen auf Shadow Page Tables zur Abbildung von virtuellem Speicher der virtuellen Maschine auf physischen Speicher des Hosts. Die Speicherverwaltung wird bei dieser Technik dem Gastbetriebssystem überlassen und erzeugt nahezu keine Verwaltungskosten. Einzige Ausnahme stellt ein Schreibzugriff auf eine noch nicht allokierte Seite dar. In diesem Fall muss über einen Mechanismus die Speicherverwaltung der virtuellen Maschine und die des physischen Systems synchronisiert werden, was einen kleinen Unterschied zwischen Lese- und Schreibrate messbar macht Die Details der Speicherverwaltung wurden in den Kapiteln 2.3, 2.4, 2.5 und 2.6 erläutert. Eine vergleichende Aussage über die Qualität der eingesetzten und sehr unterschiedlichen Speicherverwaltungstechnologien ist aufgrund der Messungen nicht möglich, da alle Produkte nahezu identische Wirkungsgrade erreichen.

Unterstützende Speicherverwaltungstechnologien in Hardware

Interessanterweise scheint keines der getesteten Produkte eine hardwarebasierte Implementierung von Speicherzugriffsverfahren zu verwenden. Vergleicht man

die Ergebnisse zwischen entsprechenden Messungen mit und ohne AMD-V, so zeigen sich im Rahmen der Messgenauigkeit keine Unterschiede zwischen den beiden Messungen. Eventuell sind die Unterschiede aber auch nicht so gravierend, als dass sie bei herkömmlichen Tests auffallen würden, und treten nur bei Anwendungen mit extrem hoher Pagefault-Rate auf. Die Existenz von Balloon-Treibern zur Anpassung der Hauptspeichergröße einer virtuellen Maschine im laufenden Betrieb zeigen ebenfalls keine negativen, aber auch keine positiven Auswirkungen auf die Leistung des Hauptspeicherdurchsatzes. Sie können damit gefahrlos eingesetzt werden, ermöglichen einen flexiblen Betrieb der virtuellen Maschinen in Bezug auf Speicherzuweisung und besitzen zumindest keine dauerhaft negativen Auswirkungen auf die Performanz einer virtuellen Maschine.

4.2.3 Messungen an der Komponente Netz

Beschreibung des Messverfahrens

Als Applikation für den Netz-Benchmark wurde das Programm Iometer [Dani 09] ausgewählt.Iometer ist ein frei verfügbares Werkzeug, das speziell für die Messung von Netz- und Festplatten-Durchsatz entwickelt wurde. Es besteht aus einer grafischen Komponente, die auf einer entfernten Maschine installiert wird, sowie einer kleinen Serveranwendung namens Dynamo, die auf der zu testenden virtuellen Maschine gestartet werden kann und als Kommunikationspartner für Messungen über das Netz dient. Für Netzmessungen wird zusätzlich zur Dynamo-Instanz in der virtuellen Maschine eine Instanz von Dynamo auf dem Rechner erzeugt, der die grafische Management-Komponente bereitstellt. Die Dynamo-Instanzen sind für die Ausführung der Benchmarks verantwortlich, während die grafische Iometer-Komponente für die Steuerung des Ablaufes der Benchmarks sowie für die Zeitmessung verantwortlich ist. Für die Tests lässt sich spezifizieren, wie hoch der prozentuale Anteil an lesendem und schreibendem Datenvolumen sein soll. Eine weitere Konfiguration, die eine Spezialisierung von Netzzugriffen auf bestimmte Framegrößen verhindert, ist die Spezifikation von unterschiedlich großen Testblöcken, mit denen der Test durchgeführt werden soll. Um eine ausgewogene Testkonfiguration zu erhalten, wurden Blockgrößen von 4KB, 32KB, 256KB, 1MB und 4MB mit einem Anteil von je 20% gewählt. Selbstverständlich können in der Praxis andere Framegrößen auftreten, insbesondere auch Framegrößen, deren Größe nicht einer Zweierpotenz entsprechen. Als Folge hieraus ist in der Praxis eventuell mit geringerem Durchsatz zu rechnen. Da auch bei diesen Messungen anschließend ein Wirkungsgrad gebildet wird, ist dieses Problem sowohl virtuell als auch physisch existent und hat auf den Wirkungsgrad der Netzvirtualisierung keinen Einfluss. Wichtig erscheint es in diesem Fall lediglich sowohl kleine als auch größere Datenpakete in der Testmenge zu haben.

Keine Anpassung des Benchmarks notwendig

Eine Anpassung des Iometer-Benchmarks auf Virtualisierung ist nicht notwendig, da die Komponente, die die Zeitmessung vornimmt, in der entfernten physischen Maschine sitzt und von der Problematik der Zeitmessung in virtuellen Maschinen nicht betroffen ist. Für die konkrete Messung wurde im Versuchsaufbau jeweils

ein dediziertes Cat6-Ethernetkabel zwischen dem Iometer-System und dem Virtualisierer, der die zu messende virtuelle Maschine bereitstellt, verlegt. Sowohl der Iometer-Rechner als auch der Virtualisierungs-Server hatten für diesen Zweck eine Intel Pro 1000 Gigabit Ethernetkarte eingebaut, die der Benchmark-Anwendung exklusiv zur Verfügung stand. Die virtuelle Maschine war mittels eines virtuellen Switches auf den Virtualisierern an dieses Netz gebunden. Sofern die Pakete während des Tests die physische Maschine verlassen mussten, stand ihnen hiermit eine eigens für den Benchmark eingerichtete Gigabit-Ethernetverbindung über ein Crossover-Kabel zum Kommunikationspartner zur Verfügung. Für den Fall, dass die kommunizierenden virtuellen Maschinen sich auf demselben Hostsystem befanden, wurde ausschließlich der virtuelle Switch des Virtualisierers genutzt.

Variable Einflussgrößen

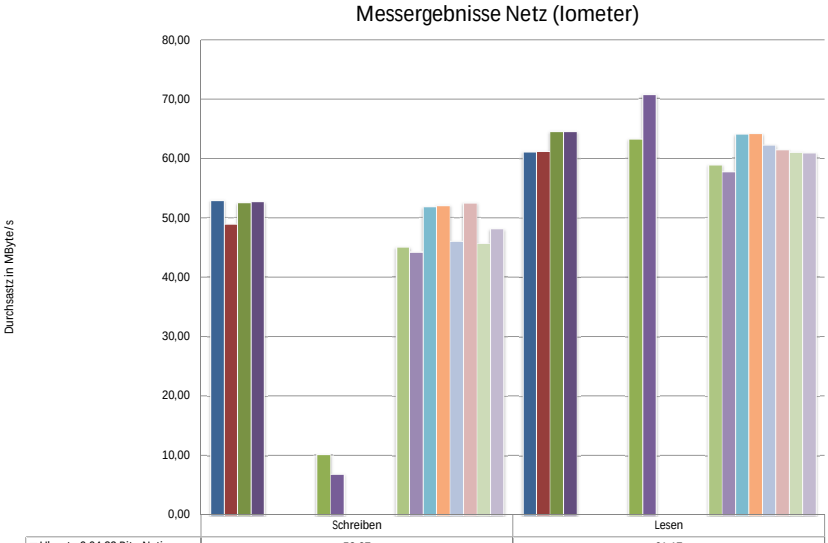
Die folgenden Einflussfaktoren bezogen auf die Standardkonfiguration wurden im Laufe der Netz basierten Messungen unabhängig voneinander variiert, um ihren Einfluss auf den Wirkungsgrad der untersuchten Virtualisierer zu messen. Aufgrund technischer Abhängigkeiten sind analog zu den CPU-Messungen nicht alle Kombinationen aller Parameter realisierbar. Nicht realisierbare Kombinationen werden in den Messergebnissen in Form leerer Felder dargestellt.

1. Virtualisierungslösung
   a) Nativ (keine Virtualisierung)
   b) Xen (Paravirtualisiert)
   c) Xen (Vollvirtualisiert)
   d) OpenVZ (OS-Virtualisierung)
   e) Virtuozzo (OS-Virtualisierung)
   f) Hyper-V (Vollvirtualisiert)
   g) VMware ESXi (Vollvirtualisiert)
2. Paravirtualisierende Treiber
   a) keine paravirtualisierenden Treiber
   b) win-pv Treiber (Xen)
   c) Integration Tools (Hyper-V)
   d) VMware Tools (ESXi)
3. Gast
   a) Windows Server 2003 32 Bit
   b) Windows Server 2003 64 Bit
   c) Ubuntu Linux 32 Bit
   d) Ubuntu Linux 64 Bit

4. Prozessorunterstützung
    a) AMD-V aktiviert
    b) AMD-V deaktiviert
5. Aktion
    a) Lesen
    b) Schreiben

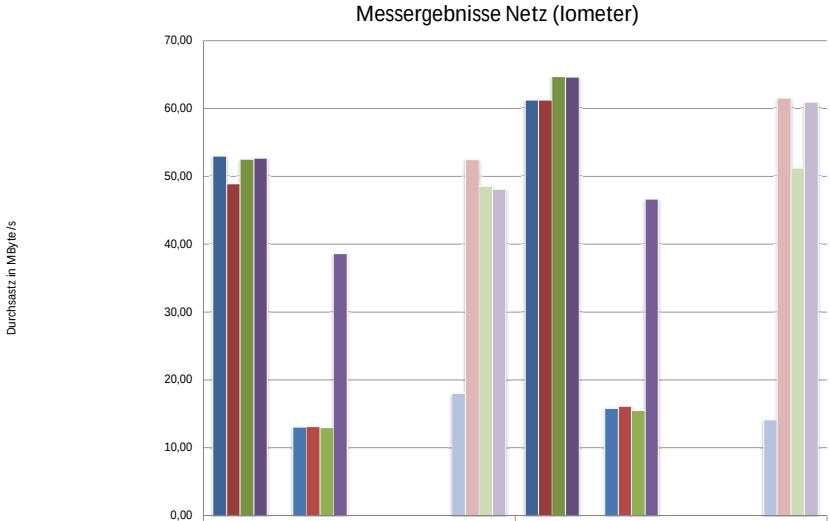
Zusätzlich wird für einige ausgewählte Konfigurationen untersucht, ob eine Platzierung der Kommunikationpunkte in physischen bzw. virtuellen Maschinen und im Falle von virtuellen Maschinen auf identischen bzw. verschiedenen physischen Maschinen Einflüsse hat. Diese zusätzlichen Parameter werden jedoch nicht hier untersucht, sondern im Abschnitt der nebenläufigen Messungen, da hierfür zum Teil mehrere virtuelle Maschinen betrieben werden müssen.

4.2.3.1 Messergebnisse



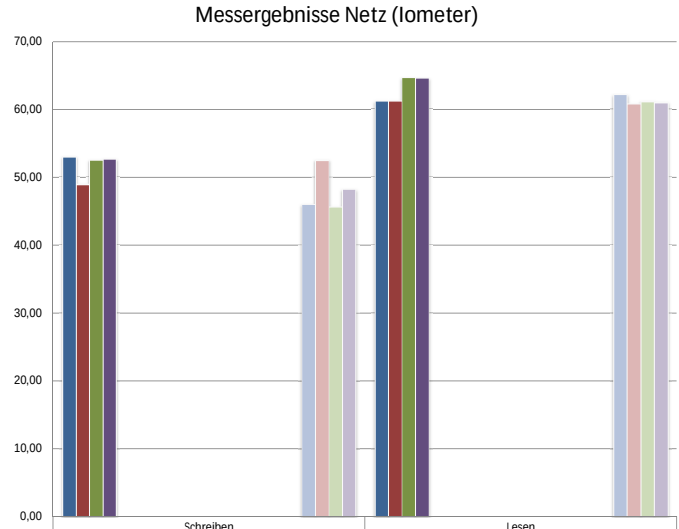
	Schreiben	Lesen
Ubuntu 9.04 32 Bit - Nativ	52,97	61,17
Ubuntu 9.04 64 Bit - Nativ	48,91	61,19
Win2K3 32 Bit - Nativ	52,50	64,59
Win2K3 64 Bit - Nativ	52,67	64,55
Ubuntu 32 Bit - Xen 64 Bit Para		
Ubuntu 64 Bit - Xen 64 Bit Para		
Ubuntu 32 Bit - Xen 64 Bit Voll		
Ubuntu 64 Bit - Xen 64 Bit Voll		
Win2K3 32 Bit - Xen 64 Bit Voll	10,09	63,24
Win2K3 64 Bit - Xen 64 Bit Voll	6,83	70,77
Ubuntu 32 Bit - OpenVZ 32 Bit		
Ubuntu 64 Bit - OpenVZ 64 Bit		
Win2K3 32 Bit - Virtuozzo 32 Bit		
Win2K3 64 Bit - Virtuozzo 64 Bit		
Ubuntu 32 Bit - HyperV 64	45,07	58,87
Ubuntu 64 Bit - HyperV 64	44,20	57,69
Win2K3 32 Bit - HyperV 64	51,79	63,98
Win2K3 64 Bit - HyperV 64	52,08	64,10
Ubuntu 32 Bit - ESXi 3.5 32 Bit	46,02	62,19
Ubuntu 64 Bit - ESXi 3.5 32 Bit	52,47	61,36
Win2K3 32 Bit - ESXi 3.5 32 Bit	45,64	61,03
Win2K3 64 Bit - ESXi 3.5 32 Bit	48,12	60,88

Abbildung 4.25: Messwerte Iometer (AMD-V, paravirt. Treiber)



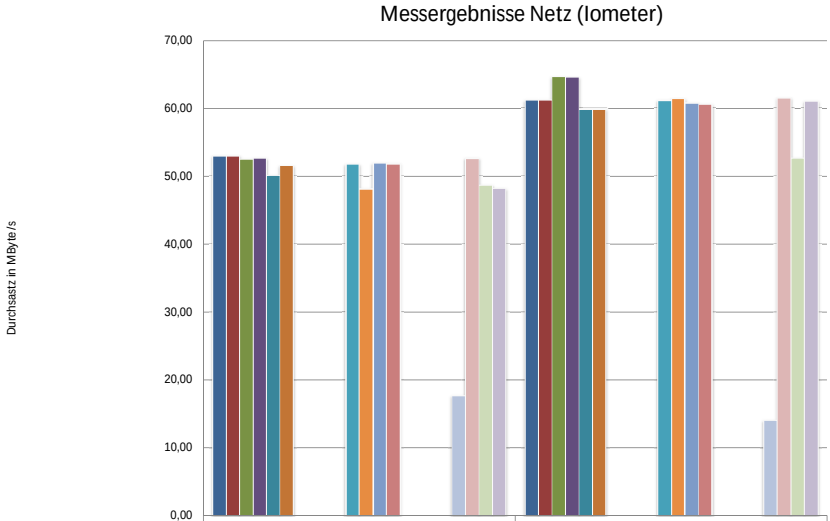
	Schreiben	Lesen
Ubuntu 9.04 32 Bit - Nativ	52,97	61,17
Ubuntu 9.04 64 Bit - Nativ	48,91	61,19
Win2K3 32 Bit - Nativ	52,50	64,59
Win2K3 64 Bit - Nativ	52,67	64,55
Ubuntu 32 Bit - Xen 64 Bit Para		
Ubuntu 64 Bit - Xen 64 Bit Para		
Ubuntu 32 Bit - Xen 64 Bit Voll	13,08	15,85
Ubuntu 64 Bit - Xen 64 Bit Voll	13,15	16,10
Win2K3 32 Bit - Xen 64 Bit Voll	12,93	15,57
Win2K3 64 Bit - Xen 64 Bit Voll	38,63	46,65
Ubuntu 32 Bit - OpenVZ 32 Bit		
Ubuntu 64 Bit - OpenVZ 64 Bit		
Win2K3 32 Bit - Virtuozzo 32 Bit		
Win2K3 64 Bit - Virtuozzo 64 Bit		
Ubuntu 32 Bit - HyperV 64		
Ubuntu 64 Bit - HyperV 64		
Win2K3 32 Bit - HyperV 64		
Win2K3 64 Bit - HyperV 64		
Ubuntu 32 Bit - ESXi 3.5 32 Bit	18,05	14,16
Ubuntu 64 Bit - ESXi 3.5 32 Bit	52,54	61,49
Win2K3 32 Bit - ESXi 3.5 32 Bit	48,62	51,19
Win2K3 64 Bit - ESXi 3.5 32 Bit	48,15	60,88

Abbildung 4.26: Messwerte Iometer (AMD-V, keine paravirt. Treiber)



	Schreiben	Lesen
Ubuntu 9.04 32 Bit - Nativ	52,97	61,17
Ubuntu 9.04 64 Bit - Nativ	48,91	61,19
Win2K3 32 Bit - Nativ	52,50	64,59
Win2K3 64 Bit - Nativ	52,67	64,55
Ubuntu 32 Bit - Xen 64 Bit Para		
Ubuntu 64 Bit - Xen 64 Bit Para		
Ubuntu 32 Bit - Xen 64 Bit Voll		
Ubuntu 64 Bit - Xen 64 Bit Voll		
Win2K3 32 Bit - Xen 64 Bit Voll		
Win2K3 64 Bit - Xen 64 Bit Voll		
Ubuntu 32 Bit - OpenVZ 32 Bit		
Ubuntu 64 Bit - OpenVZ 64 Bit		
Win2K3 32 Bit - Virtuozzo 32 Bit		
Win2K3 64 Bit - Virtuozzo 64 Bit		
Ubuntu 32 Bit - HyperV 64		
Ubuntu 64 Bit - HyperV 64		
Win2K3 32 Bit - HyperV 64		
Win2K3 64 Bit - HyperV 64		
Ubuntu 32 Bit - ESXi 3.5 32 Bit	45,99	62,17
Ubuntu 64 Bit - ESXi 3.5 32 Bit	52,55	60,79
Win2K3 32 Bit - ESXi 3.5 32 Bit	45,65	61,16
Win2K3 64 Bit - ESXi 3.5 32 Bit	48,24	60,99

Abbildung 4.27: Messwerte Iometer (kein AMD-V, paravirt. Treiber)

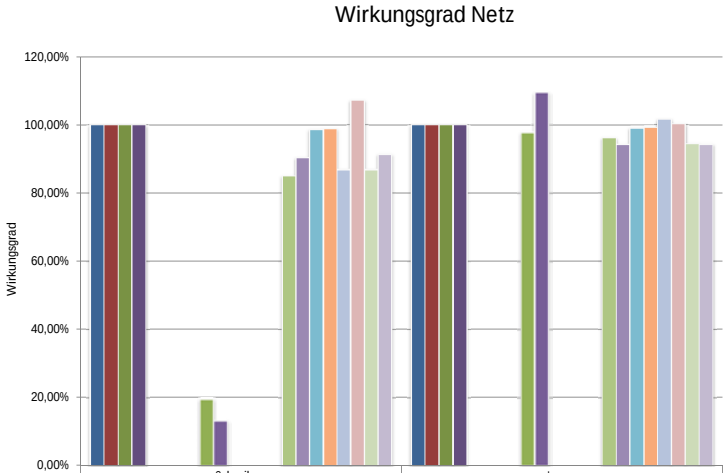


	Schreiben	Lesen
Ubuntu 9.04 32 Bit - Nativ	52,97	61,17
Ubuntu 9.04 64 Bit - Nativ	52,97	61,17
Win2K3 32 Bit - Nativ	52,50	64,59
Win2K3 64 Bit - Nativ	52,67	64,55
Ubuntu 32 Bit - Xen 64 Bit Para	50,11	59,80
Ubuntu 64 Bit - Xen 64 Bit Para	51,53	59,83
Ubuntu 32 Bit - Xen 64 Bit Voll		
Ubuntu 64 Bit - Xen 64 Bit Voll		
Win2K3 32 Bit - Xen 64 Bit Voll		
Win2K3 64 Bit - Xen 64 Bit Voll		
Ubuntu 32 Bit - OpenVZ 32 Bit	51,69	61,15
Ubuntu 64 Bit - OpenVZ 64 Bit	48,06	61,38
Win2K3 32 Bit - Virtuozzo 32 Bit	51,88	60,71
Win2K3 64 Bit - Virtuozzo 64 Bit	51,80	60,56
Ubuntu 32 Bit - HyperV 64		
Ubuntu 64 Bit - HyperV 64		
Win2K3 32 Bit - HyperV 64		
Win2K3 64 Bit - HyperV 64		
Ubuntu 32 Bit - ESXi 3.5 32 Bit	17,68	14,05
Ubuntu 64 Bit - ESXi 3.5 32 Bit	52,62	61,51
Win2K3 32 Bit - ESXi 3.5 32 Bit	48,68	52,67
Win2K3 64 Bit - ESXi 3.5 32 Bit	48,18	61,05

Abbildung 4.28: Messwerte Iometer (kein AMD-V, keine paravirt. Treiber)

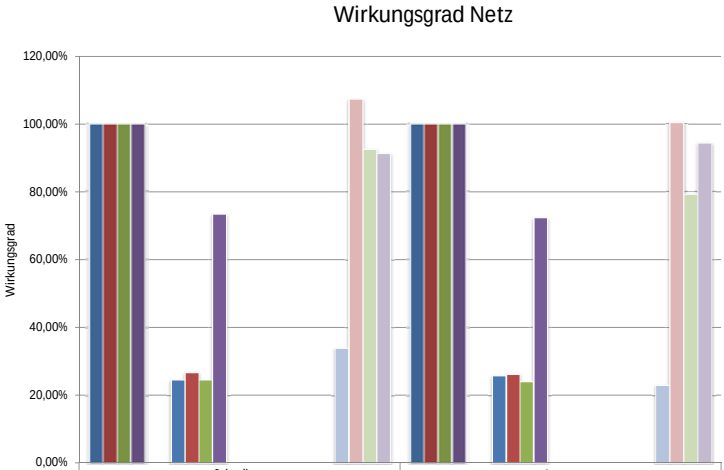
4.2.3.2 Ermittlung des Wirkungsgrades

Anders als in den CPU- und RAM-Messungen, sind die dargestellten Messergebnisse der Bechmarks im Netzbereich bereits Aggregate über eine Reihe unterschiedlicher Blockgrößen. Unterschieden wird in der Darstellung nur noch zwischen lesendem und schreibendem Zugriff auf das Netz. Auch für den Wirkungsgrad macht diese Unterteilung durchaus Sinn, so dass sich der Wirkungsgrad für das Lesen und Schreiben von Netzdaten berechnet aus dem Quotienten von Durchsatz im virtuellen und Durchsatz im nicht-virtuellen Fall. Die berechneten Wirkungsgrade werden von den Abbildungen 4.29, 4.30, 4.31, 4.32 dargestellt.



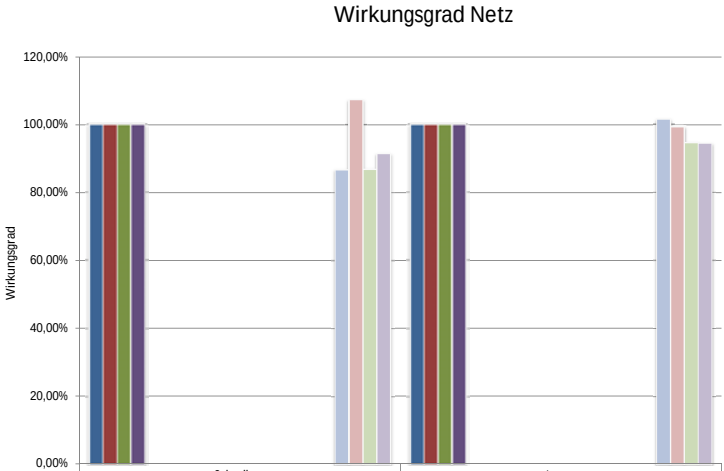
	Schreiben	Lesen
Ubuntu 9.04 32 Bit - Nativ	100,00%	100,00%
Ubuntu 9.04 64 Bit - Nativ	100,00%	100,00%
Win2K3 32 Bit - Nativ	100,00%	100,00%
Win2K3 64 Bit - Nativ	100,00%	100,00%
Ubuntu 32 Bit - Xen 64 Bit Para		
Ubuntu 64 Bit - Xen 64 Bit Para		
Ubuntu 32 Bit - Xen 64 Bit Voll		
Ubuntu 64 Bit - Xen 64 Bit Voll		
Win2K3 32 Bit - Xen 64 Bit Voll	19,21%	97,91%
Win2K3 64 Bit - Xen 64 Bit Voll	12,97%	109,63%
Ubuntu 32 Bit - OpenVZ 32 Bit		
Ubuntu 64 Bit - OpenVZ 64 Bit		
Win2K3 32 Bit - Virtuozzo 32 Bit		
Win2K3 64 Bit - Virtuozzo 64 Bit		
Ubuntu 32 Bit - HyperV 64	85,08%	96,23%
Ubuntu 64 Bit - HyperV 64	90,37%	94,28%
Win2K3 32 Bit - HyperV 64	98,65%	99,05%
Win2K3 64 Bit - HyperV 64	98,87%	99,29%
Ubuntu 32 Bit - ESXi 3.5 32 Bit	86,87%	101,67%
Ubuntu 64 Bit - ESXi 3.5 32 Bit	107,28%	100,28%
Win2K3 32 Bit - ESXi 3.5 32 Bit	86,93%	94,48%
Win2K3 64 Bit - ESXi 3.5 32 Bit	91,36%	94,31%

Abbildung 4.29: Wirkungsgrad Netz (AMD-V, paravirt. Treiber)



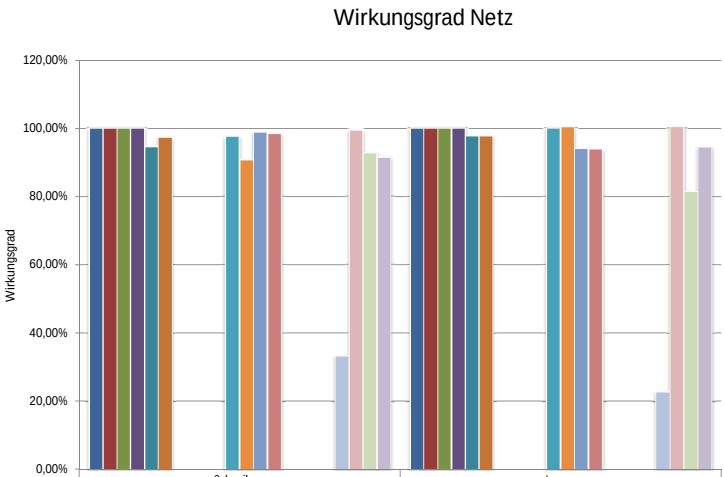
	Schreiben	Lesen
Ubuntu 9.04 32 Bit - Nativ	100,00%	100,00%
Ubuntu 9.04 64 Bit - Nativ	100,00%	100,00%
Win2K3 32 Bit - Nativ	100,00%	100,00%
Win2K3 64 Bit - Nativ	100,00%	100,00%
Ubuntu 32 Bit - Xen 64 Bit Para		
Ubuntu 64 Bit - Xen 64 Bit Para		
Ubuntu 32 Bit - Xen 64 Bit Voll	24,70%	25,91%
Ubuntu 64 Bit - Xen 64 Bit Voll	26,89%	26,31%
Win2K3 32 Bit - Xen 64 Bit Voll	24,62%	24,11%
Win2K3 64 Bit - Xen 64 Bit Voll	73,35%	72,27%
Ubuntu 32 Bit - OpenVZ 32 Bit		
Ubuntu 64 Bit - OpenVZ 64 Bit		
Win2K3 32 Bit - Virtuozzo 32 Bit		
Win2K3 64 Bit - Virtuozzo 64 Bit		
Ubuntu 32 Bit - HyperV 64		
Ubuntu 64 Bit - HyperV 64		
Win2K3 32 Bit - HyperV 64		
Win2K3 64 Bit - HyperV 64		
Ubuntu 32 Bit - ESXi 3.5 32 Bit	34,08%	23,14%
Ubuntu 64 Bit - ESXi 3.5 32 Bit	107,41%	100,48%
Win2K3 32 Bit - ESXi 3.5 32 Bit	92,61%	79,25%
Win2K3 64 Bit - ESXi 3.5 32 Bit	91,41%	94,32%

Abbildung 4.30: Wirkungsgrad Netz (AMD-V, keine paravirt. Treiber)



	Schreiben	Lesen
Ubuntu 9.04 32 Bit - Nativ	100,00%	100,00%
Ubuntu 9.04 64 Bit - Nativ	100,00%	100,00%
Win2K3 32 Bit - Nativ	100,00%	100,00%
Win2K3 64 Bit - Nativ	100,00%	100,00%
Ubuntu 32 Bit - Xen 64 Bit Para		
Ubuntu 64 Bit - Xen 64 Bit Para		
Ubuntu 32 Bit - Xen 64 Bit Voll		
Ubuntu 64 Bit - Xen 64 Bit Voll		
Win2K3 32 Bit - Xen 64 Bit Voll		
Win2K3 64 Bit - Xen 64 Bit Voll		
Ubuntu 32 Bit - OpenVZ 32 Bit		
Ubuntu 64 Bit - OpenVZ 64 Bit		
Win2K3 32 Bit - Virtuozzo 32 Bit		
Win2K3 64 Bit - Virtuozzo 64 Bit		
Ubuntu 32 Bit - HyperV 64		
Ubuntu 64 Bit - HyperV 64		
Win2K3 32 Bit - HyperV 64		
Win2K3 64 Bit - HyperV 64		
Ubuntu 32 Bit - ESXi 3.5 32 Bit	86,82%	101,63%
Ubuntu 64 Bit - ESXi 3.5 32 Bit	107,44%	99,34%
Win2K3 32 Bit - ESXi 3.5 32 Bit	86,95%	94,69%
Win2K3 64 Bit - ESXi 3.5 32 Bit	91,59%	94,49%

Abbildung 4.31: Wirkungsgrad Netz (kein AMD-V, paravirt. Treiber)



	Schreiben	Lesen
Ubuntu 9.04 32 Bit - Nativ	100,00%	100,00%
Ubuntu 9.04 64 Bit - Nativ	100,00%	100,00%
Win2K3 32 Bit - Nativ	100,00%	100,00%
Win2K3 64 Bit - Nativ	100,00%	100,00%
Ubuntu 32 Bit - Xen 64 Bit Para	94,59%	97,76%
Ubuntu 64 Bit - Xen 64 Bit Para	97,28%	97,80%
Ubuntu 32 Bit - Xen 64 Bit Voll		
Ubuntu 64 Bit - Xen 64 Bit Voll		
Win2K3 32 Bit - Xen 64 Bit Voll		
Win2K3 64 Bit - Xen 64 Bit Voll		
Ubuntu 32 Bit - OpenVZ 32 Bit	97,59%	99,95%
Ubuntu 64 Bit - OpenVZ 64 Bit	90,73%	100,33%
Win2K3 32 Bit - Virtuozzo 32 Bit	98,82%	93,99%
Win2K3 64 Bit - Virtuozzo 64 Bit	98,36%	93,82%
Ubuntu 32 Bit - HyperV 64		
Ubuntu 64 Bit - HyperV 64		
Win2K3 32 Bit - HyperV 64		
Win2K3 64 Bit - HyperV 64		
Ubuntu 32 Bit - ESXi 3.5 32 Bit	33,37%	22,96%
Ubuntu 64 Bit - ESXi 3.5 32 Bit	99,33%	100,54%
Win2K3 32 Bit - ESXi 3.5 32 Bit	92,72%	81,54%
Win2K3 64 Bit - ESXi 3.5 32 Bit	91,48%	94,57%

Abbildung 4.32: Wirkungsgrad Netz (kein AMD-V, keine paravirt. Treiber)

4.2.3.3 Auswertung

Ein erster Blick auf die Messergebnisse in den Abbildungen 4.25, 4.26, 4.27 und 4.28 lässt keine Systematik im Durchsatz in den virtuellen Netzen erkennen. Betrachtet man den Wirkungsgrad (Abbildungen 4.29, 4.30, 4.31, 4.32), welcher aus dem Verhältnis des Durchsatzes im Virtuellen zum nativen Durchsatz errechnet werden kann, so fällt auf, dass nahezu alle Wirkungsgrade über 80% liegen. Es existieren lediglich einige wenige, aber gravierende Ausreißer nach unten. Hierzu gehören nahezu alle Xen Instanzen, lediglich Windows 2003 64-Bit unter Xen ohne zusätzlich installierte Treiber erreicht mit einem Wirkungsgrad von knapp 75% noch einen halbwegs akzeptablen Wert. Alle anderen Xen Instanzen sowie Ubuntu 32-Bit unter VMware ESXi ohne paravirtualisierende Netztreiber erreichen lediglich einen Wirkungsgrad von 25% und sind damit in der Praxis meist untauglich. Auch die offenen PV-Treiber verbessern die Lage für Xen nicht. Zwar heben sie den Wirkungsgrad im Lesen auf 97% und mehr, jedoch zulasten der Schreibrate, die auf bis zu 12% fällt.

AMD-V bringt keinen Vorteil

Vergleicht man die Wirkungsgrade der Messungen mit aktivierter und deaktivierter Prozessorunterstützung untereinander, so stellt man fest, dass keine der Virtualisierungslösungen Gebrauch davon macht, beziehungsweise einen Vorteil aus der Verfügbarkeit dieser Technologie schöpft, denn die Wirkungsgrade sind bis auf Messungenauigkeiten identisch. Vergleicht man weiter die Messungen mit installierten VMware Tools mit denen ohne installierte VMware Tools, so zeigt sich eine leichte Verbesserung des Wirkungsgrades im Lesen um ca. 5%, die jedoch gleichzeitig im Schreiben verloren gehen. Eine generelle Notwendigkeit von speziellen Treibern für die Virtualisierung von Netzkomponenten kann aus diesen Messungen nicht abgeleitet werden, allerdings führen diese Treiber in keinem der untersuchten Fälle zu einer echten Verschlechterung des Verhaltens. Eine Installation dieser Treiber kann den nativ installierten Treibern, wie am Beispiel Ubuntu/ESXi zu sehen ist, jedoch gravierend überlegen sein, so dass diese durchaus empfehlenswert ist. Laut VMware greifen die Treiber hauptsächlich bei kleinen Blockgrößen, da dort viele kleine Pakete verschickt werden müssen, die ohne entsprechende Treiber alle eine Behandlung im Hypervisor erforderlich machen [VMwa 08]. Dieser Effekt ist in den abgebildeten Messreihen nicht ersichtlich, da sie einen Durchschnitt über viele Messreihen mit unterschiedlichen Blockgrößen darstellen. In der Praxis dürfte dieser Effekt in vielen Fällen nicht sichtbar sein, mit Ausnahme von Anwendungsfällen, die in der Regel sehr kleine Pakete verschicken. Hierunter könnten zum Beispiel Webserver fallen, deren bereitgestellte Inhalte meist eher kleine HTML-Dateien sind.

Lesen vs. Schreiben

Ein Unterschied der Wirkungsgrade im Lesen und Schreiben ist global nicht feststellbar. Zu je etwa 50% der Fälle besitzen die untersuchten Kandidaten entweder einen besseren Wirkungsgrad im Lesen- oder im Schreiben. Es scheint ein Trade-Off zwischen beiden Modi zu existieren, so dass je nach Implementierung entweder der eine oder der andere geringfügig bevorzugt wird. Ein Unterschied

zwischen den untersuchten Architekturen und eingesetzten Betriebssystem kann nicht abgeleitet werden.

Rückschluss auf die Virtualisierungstechnik

Ein Rückschluss auf die Leistungsfähigkeit der Virtualisierungstechniken im Bereich Netzvirtualisierung auf Basis der Messungen ist nur eingeschränkt möglich, da lediglich zwei Arten in der Praxis eingesetzt werden. Zum einen reine Emulation, welche zum Beispiel von Xen im HVM Modus eingesetzt wird, zum anderen Paravirtualisierung, welche von nahezu allen anderen Produkten eingesetzt wird und durch entsprechende Treiber im Gast unterstützt werden kann. Auch wenn optimierte Treiber für emulierte Netzkarten wie im Beispiel Xen HVM existieren, so zeigen die Messungen, dass dieser Ansatz Probleme bereitet und auch von der Theorie her keinen Sinn macht, da die virtuelle Netzkarte eine an Paravirtualisierung angepasste Schnittstelle bereitstellen müsste, was eine emulierte Ressource per Definition jedoch nicht leistet.

4.2.4 Messungen an der Komponente Disk

Beschreibung des Messverfahrens

Wie schon die Netzmessungen werden auch die Messungen des Durchsatzes von virtualisierten Hintergrundspeichern mit Iometer durchgeführt. Im Vergleich zu den Netzmessungen entfällt für diese Konstellation allerdings der Einsatz des Dienstprogrammes Dynamo auf dem Managementrechner, da die Dynamo-Instanz in der virtuellen Maschine statt einen entfernten Kommunikationspartner direkt den Hintergrundspeicher anspricht. Das Programm Iometer wird in diesem Fall direkt in der virtuellen Maschine aufgerufen und wurde für die Tests mit denselben Parametern konfiguriert wie bei den Netzmessungen, das heißt, während der Tests wird sowohl lesend als auch schreibend auf die virtuelle Festplatte zugegriffen, es werden sequentiell und zufällig Datenblöcke der Festplatte angesprochen und Blöcke unterschiedlicher Größe gelesen beziehungsweise geschrieben. Die virtuelle Festplatte selbst wird von einem File-Server bereitgestellt, dessen Speicher mittels iSCSI durch den Hypervisor oder das Host-Betriebssystem in den Speicherpool des Virtualisierers eingebunden ist. Bei der Netzverbindung zwischen File-Server und Virtualisierer handelt es sich um ein dediziertes Gigabit-Ethernet, bei dem iSCSI Initiator (Client) um Softwarelösungen, die durch die Hypervisor oder Host-Betriebssysteme bereitgestellt werden. Da auch Hardware iSCSI Initiator existieren, wurden zusätzlich exemplarisch auch einzelne Messungen mit diesen durchgeführt, um abschätzen zu können, ob die Investition in Hardware iSCSI Initiator lohnend sind oder softwarebasierte Lösungen für die gängigsten Fälle ausreichend sind.

Keine Anpassung des Benchmarks notwendig

Eine Anpassung des Iometer-Benchmarks auf Virtualisierung ist auch für Messungen des Durchsatzes von Hintergrundspeichern nicht notwendig, da die Iometer-Komponente selbst auf einem physischen Rechner ausgeführt werden kann. Die Zeitmessung ist damit unabhängig von der Virtualisierung möglich.

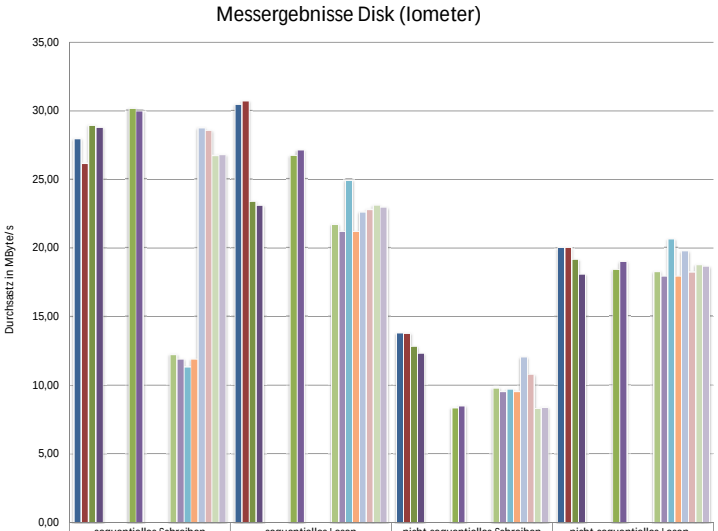
Variable Einflussgrößen

Die folgenden Einflussfaktoren bezogen auf die Standardkonfiguration werden im Laufe der Festplatten basierten Messungen unabhängig voneinander variiert, um ihren Einfluss auf den Wirkungsgrad der untersuchten Virtualisierer zu messen. Aufgrund technischer Abhängigkeiten sind analog zu den CPU-Messungen nicht alle Kombinationen aller Parameter realisierbar. Nicht realisierbare Kombinationen werden in den Messergebnissen in Form leerer Felder dargestellt.

1. Virtualisierungslösung
   a) Nativ (keine Virtualisierung)
   b) Xen (Paravirtualisiert)
   c) Xen (Vollvirtualisiert)
   d) OpenVZ (OS-Virtualisierung)
   e) Virtuozzo (OS-Virtualisierung)
   f) Hyper-V (Vollvirtualisiert)
   g) VMware ESXi (Vollvirtualisiert)
2. Paravirtualisierende Treiber
   a) keine paravirtualisierenden Treiber
   b) win-pv Treiber (Xen)
   c) Integration Tools (Hyper-V)
   d) VMware Tools (ESXi)
3. Gast
   a) Windows Server 2003 32 Bit
   b) Windows Server 2003 64 Bit
   c) Ubuntu Linux 32 Bit
   d) Ubuntu Linux 64 Bit
4. Prozessorunterstützung
   a) AMD-V aktiviert
   b) AMD-V deaktiviert
5. Aktion
   a) Lesen
   b) Schreiben
6. Typ
   a) Sequentiell
   b) Zufällig

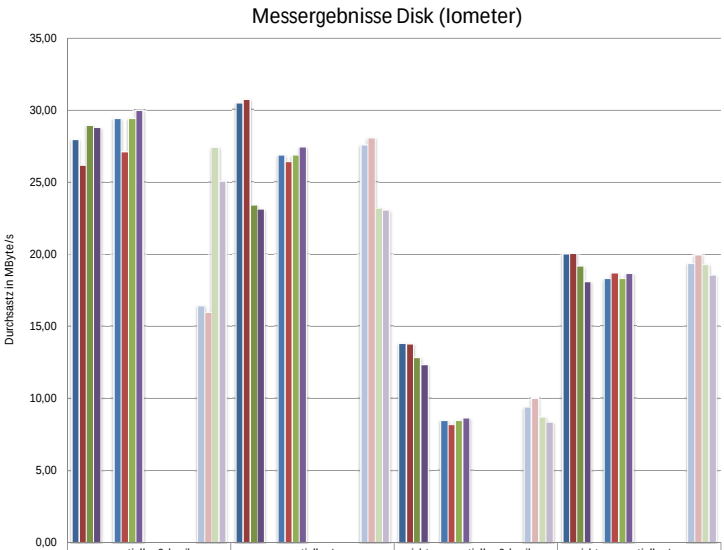
Weiterhin wird untersucht, welche Verbesserungen sich durch den Einsatz von Hardware HBAs gegenüber von Software basierten iSCSI Adaptern erzielen lassen. Hier ist neben dem erzielbaren Durchsatz insbesondere von Interesse, wie stark die CPUs des physischen Systems durch diese Technologie entlastet werden können und diese Ressourcen anderweitig eingesetzt werden können.

4.2.4.1 Messergebnisse



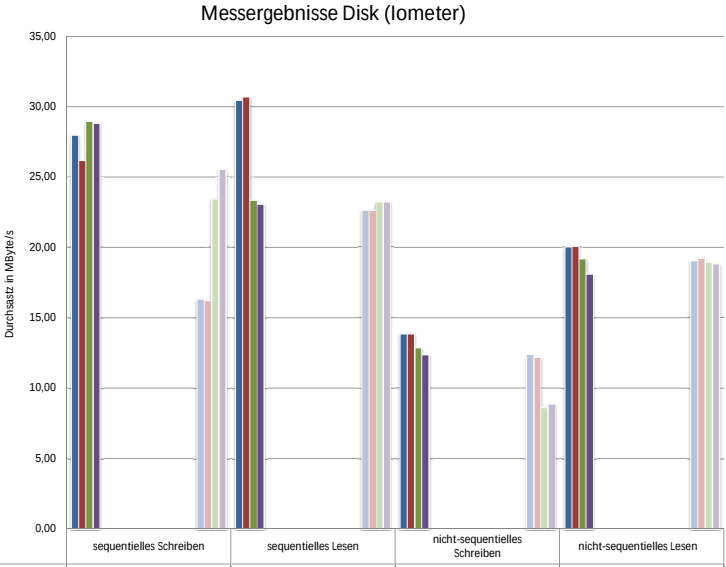
	sequentielles Schreiben	sequentielles Lesen	nicht-sequentielles Schreiben	nicht-sequentielles Lesen
Ubuntu 9.04 32 Bit - Nativ	27,98	30,46	13,82	20,03
Ubuntu 9.04 64 Bit - Nativ	26,18	30,71	13,80	20,04
Win2K3 32 Bit - Nativ	28,96	23,39	12,84	19,19
Win2K3 64 Bit - Nativ	28,79	23,09	12,34	18,11
Ubuntu 32 Bit - Xen 64 Bit Para				
Ubuntu 64 Bit - Xen 64 Bit Para				
Ubuntu 32 Bit - Xen 64 Bit Voll				
Ubuntu 64 Bit - Xen 64 Bit Voll				
Win2K3 32 Bit - Xen 64 Bit Voll	30,13	26,69	8,37	18,38
Win2K3 64 Bit - Xen 64 Bit Voll	29,95	27,11	8,51	18,96
Ubuntu 32 Bit - OpenVZ 32 Bit				
Ubuntu 64 Bit - OpenVZ 64 Bit				
Win2K3 32 Bit - Virtuozzo 32 Bit				
Win2K3 64 Bit - Virtuozzo 64 Bit				
Ubuntu 32 Bit - HyperV 64	12,26	21,72	9,81	18,26
Ubuntu 64 Bit - HyperV 64	11,91	21,18	9,54	17,91
Win2K3 32 Bit - HyperV 64	11,35	24,93	9,71	20,62
Win2K3 64 Bit - HyperV 64	11,91	21,18	9,54	17,91
Ubuntu 32 Bit - ESXi 3.5 32 Bit	28,73	22,61	12,06	19,75
Ubuntu 64 Bit - ESXi 3.5 32 Bit	28,52	22,80	10,77	18,21
Win2K3 32 Bit - ESXi 3.5 32 Bit	26,73	23,09	8,31	18,75
Win2K3 64 Bit - ESXi 3.5 32 Bit	26,79	22,95	8,40	18,63

Abbildung 4.33: Messwerte Iometer (AMD-V, paravirt. Treiber)



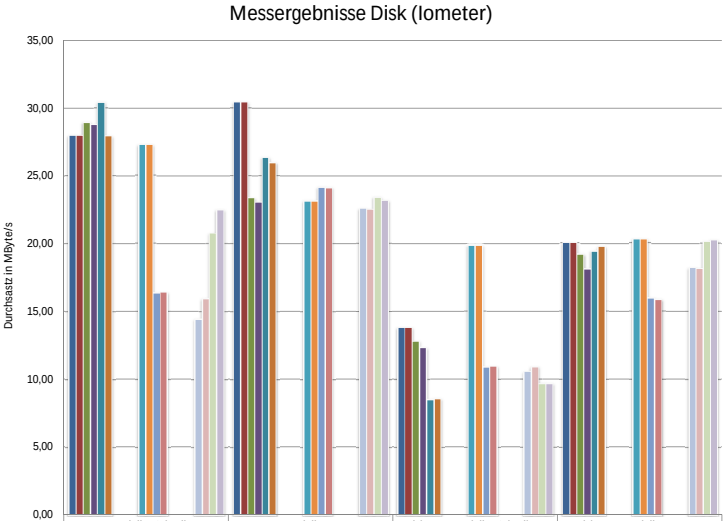
	sequentielles Schreiben	sequentielles Lesen	nicht-sequentielles Schreiben	nicht-sequentielles Lesen
Ubuntu 9.04 32 Bit - Nativ	27,98	30,46	13,82	20,03
Ubuntu 9.04 64 Bit - Nativ	26,18	30,71	13,80	20,04
Win2K3 32 Bit - Nativ	28,96	23,39	12,84	19,19
Win2K3 64 Bit - Nativ	28,79	23,09	12,34	18,11
Ubuntu 32 Bit - Xen 64 Bit Para				
Ubuntu 64 Bit - Xen 64 Bit Para				
Ubuntu 32 Bit - Xen 64 Bit Voll	29,41	26,85	8,42	18,27
Ubuntu 64 Bit - Xen 64 Bit Voll	27,12	26,40	8,11	18,68
Win2K3 32 Bit - Xen 64 Bit Voll	29,41	26,85	8,42	18,27
Win2K3 64 Bit - Xen 64 Bit Voll	29,99	27,42	8,60	18,62
Ubuntu 32 Bit - OpenVZ 32 Bit				
Ubuntu 64 Bit - OpenVZ 64 Bit				
Win2K3 32 Bit - Virtuozzo 32 Bit				
Win2K3 64 Bit - Virtuozzo 64 Bit				
Ubuntu 32 Bit - HyperV 64				
Ubuntu 64 Bit - HyperV 64				
Win2K3 32 Bit - HyperV 64				
Win2K3 64 Bit - HyperV 64				
Ubuntu 32 Bit - ESXi 3.5 32 Bit	16,44	27,57	9,38	19,36
Ubuntu 64 Bit - ESXi 3.5 32 Bit	15,97	28,04	10,00	19,96
Win2K3 32 Bit - ESXi 3.5 32 Bit	27,43	23,17	8,70	19,30
Win2K3 64 Bit - ESXi 3.5 32 Bit	25,09	23,04	8,34	18,55

Abbildung 4.34: Messwerte Iometer (AMD-V, keine paravirt. Treiber)



	sequentielles Schreiben	sequentielles Lesen	nicht-sequentielles Schreiben	nicht-sequentielles Lesen
Ubuntu 9.04 32 Bit - Nativ	27,98	30,46	13,82	20,03
Ubuntu 9.04 64 Bit - Nativ	26,18	30,71	13,80	20,04
Win2K3 32 Bit - Nativ	28,96	23,39	12,84	19,19
Win2K3 64 Bit - Nativ	28,79	23,09	12,34	18,11
Ubuntu 32 Bit - Xen 64 Bit Para				
Ubuntu 64 Bit - Xen 64 Bit Para				
Ubuntu 32 Bit - Xen 64 Bit Voll				
Ubuntu 64 Bit - Xen 64 Bit Voll				
Win2K3 32 Bit - Xen 64 Bit Voll				
Win2K3 64 Bit - Xen 64 Bit Voll				
Ubuntu 32 Bit - OpenVZ 32 Bit				
Ubuntu 64 Bit - OpenVZ 64 Bit				
Win2K3 32 Bit - Virtuozzo 32 Bit				
Win2K3 64 Bit - Virtuozzo 64 Bit				
Ubuntu 32 Bit - HyperV 64				
Ubuntu 64 Bit - HyperV 64				
Win2K3 32 Bit - HyperV 64				
Win2K3 64 Bit - HyperV 64				
Ubuntu 32 Bit - ESXi 3.5 32 Bit	16,34	22,59	12,36	19,00
Ubuntu 64 Bit - ESXi 3.5 32 Bit	16,22	22,60	12,18	19,20
Win2K3 32 Bit - ESXi 3.5 32 Bit	23,44	23,18	8,64	18,89
Win2K3 64 Bit - ESXi 3.5 32 Bit	25,55	23,19	8,89	18,82

Abbildung 4.35: Messwerte Iometer (kein AMD-V, paravirt. Treiber)

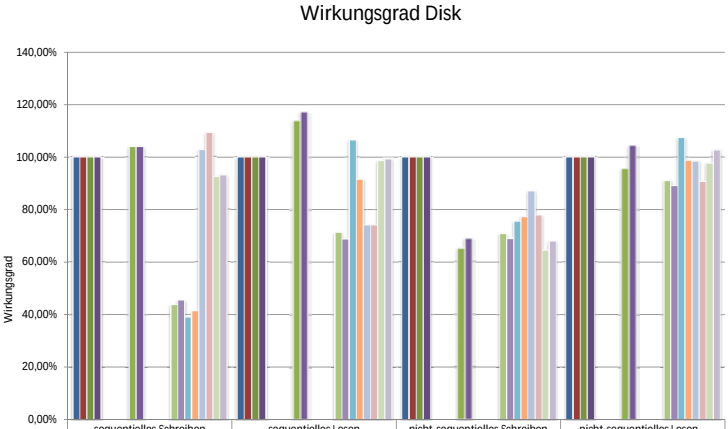


	sequentielles Schreiben	sequentielles Lesen	nicht-sequentielles Schreiben	nicht-sequentielles Lesen
Ubuntu 9.04 32 Bit - Nativ	27,98	30,46	13,82	20,03
Ubuntu 9.04 64 Bit - Nativ	27,98	30,46	13,82	20,03
Win2K3 32 Bit - Nativ	28,96	23,39	12,84	19,19
Win2K3 64 Bit - Nativ	28,79	23,09	12,34	18,11
Ubuntu 32 Bit - Xen 64 Bit Para	30,43	26,38	8,55	19,42
Ubuntu 64 Bit - Xen 64 Bit Para	27,94	25,99	8,60	19,74
Ubuntu 32 Bit - Xen 64 Bit Voll				
Ubuntu 64 Bit - Xen 64 Bit Voll				
Win2K3 32 Bit - Xen 64 Bit Voll				
Win2K3 64 Bit - Xen 64 Bit Voll				
Ubuntu 32 Bit - OpenVZ 32 Bit	27,27	23,17	19,84	20,30
Ubuntu 64 Bit - OpenVZ 64 Bit	27,27	23,17	19,84	20,30
Win2K3 32 Bit - Virtuozzo 32 Bit	16,36	24,18	10,92	15,97
Win2K3 64 Bit - Virtuozzo 64 Bit	16,41	24,15	10,96	15,87
Ubuntu 32 Bit - HyperV 64				
Ubuntu 64 Bit - HyperV 64				
Win2K3 32 Bit - HyperV 64				
Win2K3 64 Bit - HyperV 64				
Ubuntu 32 Bit - ESXi 3.5 32 Bit	14,46	22,59	10,57	18,22
Ubuntu 64 Bit - ESXi 3.5 32 Bit	15,97	22,53	10,91	18,15
Win2K3 32 Bit - ESXi 3.5 32 Bit	20,79	23,43	9,69	20,16
Win2K3 64 Bit - ESXi 3.5 32 Bit	22,50	23,19	9,69	20,26

Abbildung 4.36: Messwerte Iometer (kein AMD-V, keine paravirt. Treiber)

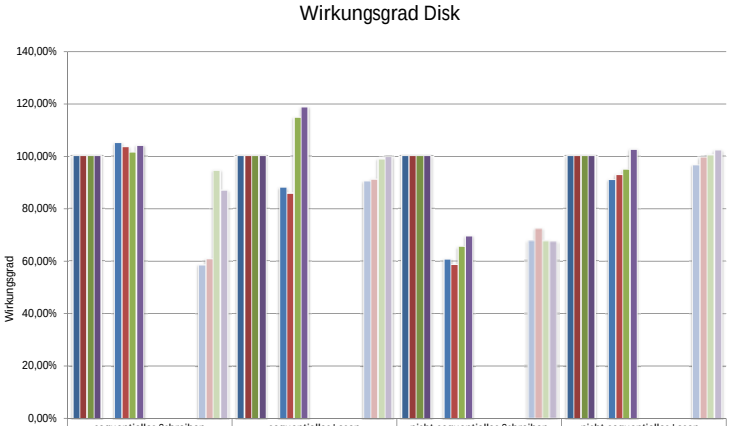
4.2.4.2 Ermittlung des Wirkungsgrades

Die Darstellung der Messergebnisse für die Disk-Messungen erfolgt auf Basis der Kombinationen „sequentielles Schreiben", „sequentielles Lesen", „nicht-sequentielles Schreiben" und „nicht-sequentielles Lesen". Die Bildung des Wirkungsgrades erfolgt in derselben Gruppierung. Der Wirkungsgrad errechnet sich damit aus dem Quotienten des Durchsatzes einer Gruppe im virtuellen Fall und dem dazugehörigen Durchsatz im physischen Fall. Eine Betrachtung des Wirkungsgrades auf Basis einzelner Blockgrößen ist nicht möglich, da Iometer die erzielten Übertragungsraten als Durchschnitt über alle gemessenen Blockgrößen protokolliert.



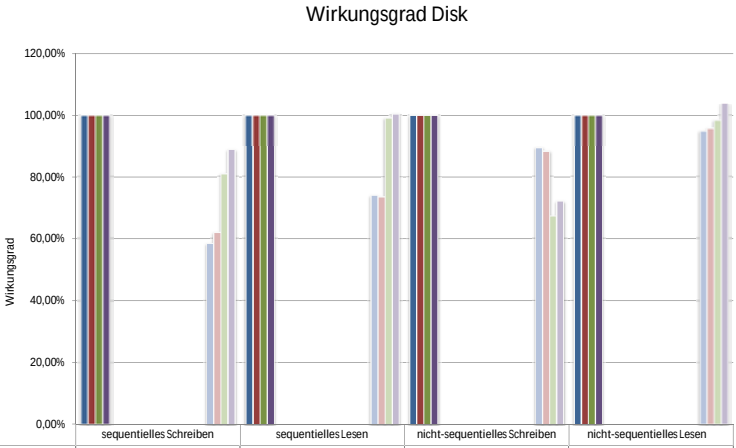
	sequentielles Schreiben	sequentielles Lesen	nicht-sequentielles Schreiben	nicht-sequentielles Lesen
Ubuntu 9.04 32 Bit - Nativ	100,00%	100,00%	100,00%	100,00%
Ubuntu 9.04 64 Bit - Nativ	100,00%	100,00%	100,00%	100,00%
Win2K3 32 Bit - Nativ	100,00%	100,00%	100,00%	100,00%
Win2K3 64 Bit - Nativ	100,00%	100,00%	100,00%	100,00%
Ubuntu 32 Bit - Xen 64 Bit Para				
Ubuntu 64 Bit - Xen 64 Bit Para				
Ubuntu 32 Bit - Xen 64 Bit Voll				
Ubuntu 64 Bit - Xen 64 Bit Voll				
Win2K3 32 Bit - Xen 64 Bit Voll	104,03%	114,12%	65,18%	95,76%
Win2K3 64 Bit - Xen 64 Bit Voll	104,04%	117,42%	68,92%	104,69%
Ubuntu 32 Bit - OpenVZ 32 Bit				
Ubuntu 64 Bit - OpenVZ 64 Bit				
Win2K3 32 Bit - Virtuozzo 32 Bit				
Win2K3 64 Bit - Virtuozzo 64 Bit				
Ubuntu 32 Bit - HyperV 64	43,80%	71,31%	70,96%	91,20%
Ubuntu 64 Bit - HyperV 64	45,51%	68,98%	69,15%	89,35%
Win2K3 32 Bit - HyperV 64	39,18%	106,57%	75,68%	107,44%
Win2K3 64 Bit - HyperV 64	41,39%	91,75%	77,33%	98,89%
Ubuntu 32 Bit - ESXi 3.5 32 Bit	102,68%	74,25%	87,22%	98,62%
Ubuntu 64 Bit - ESXi 3.5 32 Bit	108,96%	74,24%	78,04%	90,88%
Win2K3 32 Bit - ESXi 3.5 32 Bit	92,29%	98,74%	64,73%	97,70%
Win2K3 64 Bit - ESXi 3.5 32 Bit	93,05%	99,41%	68,06%	102,88%

Abbildung 4.37: Wirkungsgrad Disk (AMD-V, paravirt. Treiber)



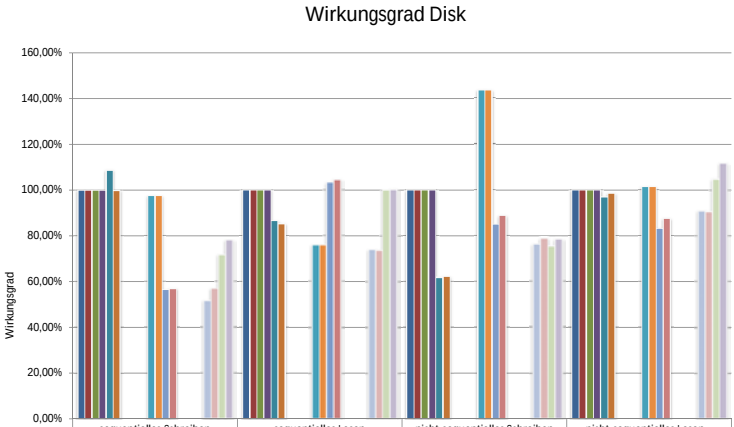
	sequentielles Schreiben	sequentielles Lesen	nicht-sequentielles Schreiben	nicht-sequentielles Lesen
Ubuntu 9.04 32 Bit - Nativ	100,00%	100,00%	100,00%	100,00%
Ubuntu 9.04 64 Bit - Nativ	100,00%	100,00%	100,00%	100,00%
Win2K3 32 Bit - Nativ	100,00%	100,00%	100,00%	100,00%
Win2K3 64 Bit - Nativ	100,00%	100,00%	100,00%	100,00%
Ubuntu 32 Bit - Xen 64 Bit Para				
Ubuntu 64 Bit - Xen 64 Bit Para				
Ubuntu 32 Bit - Xen 64 Bit Voll	105,11%	88,17%	60,93%	91,23%
Ubuntu 64 Bit - Xen 64 Bit Voll	103,60%	85,96%	58,77%	93,19%
Win2K3 32 Bit - Xen 64 Bit Voll	101,56%	114,81%	65,61%	95,19%
Win2K3 64 Bit - Xen 64 Bit Voll	104,18%	118,74%	69,70%	102,82%
Ubuntu 32 Bit - OpenVZ 32 Bit				
Ubuntu 64 Bit - OpenVZ 64 Bit				
Win2K3 32 Bit - Virtuozzo 32 Bit				
Win2K3 64 Bit - Virtuozzo 64 Bit				
Ubuntu 32 Bit - HyperV 64				
Ubuntu 64 Bit - HyperV 64				
Win2K3 32 Bit - HyperV 64				
Win2K3 64 Bit - HyperV 64				
Ubuntu 32 Bit - ESXi 3.5 32 Bit	58,74%	90,53%	67,90%	96,70%
Ubuntu 64 Bit - ESXi 3.5 32 Bit	61,00%	91,31%	72,48%	99,60%
Win2K3 32 Bit - ESXi 3.5 32 Bit	94,71%	99,07%	67,79%	100,56%
Win2K3 64 Bit - ESXi 3.5 32 Bit	87,15%	99,79%	67,56%	102,42%

Abbildung 4.38: Wirkungsgrad Disk (AMD-V, keine paravirt. Treiber)



	sequentielles Schreiben	sequentielles Lesen	nicht-sequentielles Schreiben	nicht-sequentielles Lesen
Ubuntu 9.04 32 Bit - Nativ	100,00%	100,00%	100,00%	100,00%
Ubuntu 9.04 64 Bit - Nativ	100,00%	100,00%	100,00%	100,00%
Win2K3 32 Bit - Nativ	100,00%	100,00%	100,00%	100,00%
Win2K3 64 Bit - Nativ	100,00%	100,00%	100,00%	100,00%
Ubuntu 32 Bit - Xen 64 Bit Para				
Ubuntu 64 Bit - Xen 64 Bit Para				
Ubuntu 32 Bit - Xen 64 Bit Voll				
Ubuntu 64 Bit - Xen 64 Bit Voll				
Win2K3 32 Bit - Xen 64 Bit Voll				
Win2K3 64 Bit - Xen 64 Bit Voll				
Ubuntu 32 Bit - OpenVZ 32 Bit				
Ubuntu 64 Bit - OpenVZ 64 Bit				
Win2K3 32 Bit - Virtuozzo 32 Bit				
Win2K3 64 Bit - Virtuozzo 64 Bit				
Ubuntu 32 Bit - HyperV 64				
Ubuntu 64 Bit - HyperV 64				
Win2K3 32 Bit - HyperV 64				
Win2K3 64 Bit - HyperV 64				
Ubuntu 32 Bit - ESXi 3.5 32 Bit	58,41%	74,16%	89,39%	94,89%
Ubuntu 64 Bit - ESXi 3.5 32 Bit	61,96%	73,60%	88,22%	95,80%
Win2K3 32 Bit - ESXi 3.5 32 Bit	80,93%	99,13%	67,32%	98,41%
Win2K3 64 Bit - ESXi 3.5 32 Bit	88,74%	100,43%	72,05%	103,92%

Abbildung 4.39: Wirkungsgrad Disk (kein AMD-V, paravirt. Treiber)



	sequentielles Schreiben	sequentielles Lesen	nicht-sequentielles Schreiben	nicht-sequentielles Lesen
Ubuntu 9.04 32 Bit - Nativ	100,00%	100,00%	100,00%	100,00%
Ubuntu 9.04 64 Bit - Nativ	100,00%	100,00%	100,00%	100,00%
Win2K3 32 Bit - Nativ	100,00%	100,00%	100,00%	100,00%
Win2K3 64 Bit - Nativ	100,00%	100,00%	100,00%	100,00%
Ubuntu 32 Bit - Xen 64 Bit Para	108,76%	86,61%	61,83%	96,98%
Ubuntu 64 Bit - Xen 64 Bit Para	99,87%	85,33%	62,22%	98,58%
Ubuntu 32 Bit - Xen 64 Bit Voll				
Ubuntu 64 Bit - Xen 64 Bit Voll				
Win2K3 32 Bit - Xen 64 Bit Voll				
Win2K3 64 Bit - Xen 64 Bit Voll				
Ubuntu 32 Bit - OpenVZ 32 Bit	97,44%	76,09%	143,56%	101,36%
Ubuntu 64 Bit - OpenVZ 64 Bit	97,44%	76,09%	143,56%	101,36%
Win2K3 32 Bit - Virtuozzo 32 Bit	56,49%	103,38%	85,05%	83,20%
Win2K3 64 Bit - Virtuozzo 64 Bit	57,02%	104,60%	88,79%	87,62%
Ubuntu 32 Bit - HyperV 64				
Ubuntu 64 Bit - HyperV 64				
Win2K3 32 Bit - HyperV 64				
Win2K3 64 Bit - HyperV 64				
Ubuntu 32 Bit - ESXi 3.5 32 Bit	51,69%	74,16%	76,44%	90,99%
Ubuntu 64 Bit - ESXi 3.5 32 Bit	57,07%	73,97%	78,92%	90,64%
Win2K3 32 Bit - ESXi 3.5 32 Bit	71,78%	100,16%	75,45%	105,03%
Win2K3 64 Bit - ESXi 3.5 32 Bit	78,18%	100,45%	78,53%	111,91%

Abbildung 4.40: Wirkungsgrad Disk (kein AMD-V, keine paravirt. Treiber)

4.2.4.3 Auswertung

Die Abbildungen 4.33, 4.34, 4.35 und 4.36 zeigen die Ergebnisse der Disk-Messungen. Bei der Betrachtung lässt sich auch bei genauerer Analyse kein global gültiges Muster extrahieren, das alle Messungen abdeckt. Auch eine Betrachtung des zugehörigen Wirkungsgrades (siehe Abbildungen 4.37, 4.38, 4.39 und 4.40) ändert nichts an dieser Tatsache. Der Wirkungsgrad schwankt über eine Bandbreite von 50% bis gut 100%. Dennoch lassen sich partiell einige Aussagen treffen. Generell ist der Wirkungsgrad im Lesen höher als im Schreiben und 32-Bit und 64-Bit Versionen einer Betriebssystemfamilie erzielen sehr ähnliche Wirkungsgrade. Unterschiedliche Betriebssysteme können auf demselben Virtualisierungsprodukt signifikant abweichende Transferraten und Wirkungsgrade erreichen. Ebenso verhält sich die Anwesenheit von paravirtualisierenden Treibern. Diese können die Performanz erheblich steigern, können aber auch keinen messbaren Effekt zeigen. In einigen Fällen ist sogar ein leichter Einbruch des Wirkungsgrades erkennbar. Der Nutzen von paravirtualisierenden Festplattentreibern ist daher abhängig von den mitgelieferten Treibern des Betriebssystems und der Schnittstelle des Virtualisierers. Eine globale Aussage über den Einfluss solcher Treiber auf den Wirkungsgrad ist nicht möglich, sondern muss individuell erfolgen. Die Abwesenheit von Prozessorunterstützung verursacht bei VMware ESXi teilweise Einbrüche im Wirkungsgrad von bis zu 15%, während andere Wirkungsgrade konstant bleiben. Ebenso wenig können allgemeingültige Aussagen über sequentielle und nicht-sequentielle Zugriffe gefolgert werden. Während sich sequentielle Zugriffe theoretisch einfacher und damit mit weniger Aufwand realisieren lassen sollten als nicht-sequentielle Zugriffe, stellt sich die Tatsache bei beinahe allen Produkten exakt invers dar. Der Grund für dieses Verhalten liegt vermutlich in der notwendigen Übersetzung von Adressen, die nicht zwangsweise linear ist. Sequentielle Zugriffe aus Sicht der virtuellen Maschine werden durch diese Übersetzung von Adressen teilweise zu nicht-sequentiellen Zugriffen auf den Schichten darunter. Insbesondere Plattenzugriffe verlangsamen sich durch diese Tatsache, sodass sie gegenüber den nativen Systemen schlechter abschneiden. Auch der Einsatz von Hardware iSCSI HBAs ändert an diesem Verhalten nichts. Der Durchsatz bleibt nahezu konstant.

4.3 Messung des Wirkungsgrades nebenläufiger Maschinen

Motivation für nebenläufige Messungen

Neben dem Wirkungsgrad von Virtualisierungstechniken und Lösungen, der bereits sehr wesentlichen Einfluss auf die Virtualisierbarkeit von vorhandenen Maschinen und Anwendungen besitzt, ist zusätzlich auch der gegenseitige Einfluss virtueller Maschinen auf ihre Nachbarn zu beachten. Anders als im physischen Szenario kann eine virtuelle Maschine nicht davon ausgehen, alle vorhandenen Ressourcen jederzeit uneingeschränkt verwenden zu können. Zwar versucht der Hypervisor

den Zugriff auf vorhandene physische Ressourcen zu abstrahieren und ihre Leistungsfähigkeit optimal an alle virtuellen Maschinen zu verteilen, kann dabei jedoch einen gewissen Leistungseinbruch nicht verhindern, da die Gesamtleistung der vorhandenen Komponenten beschränkt ist und er keinen Einfluss auf in Hardware implementierte Funktionen wie zum Beispiel das Caching besitzt. Aus diesem Grund werden in diesem Kapitel Messungen an parallel auf einem physischen System ausgeführten virtuellen Maschinen durchgeführt.

Einschränkung der Parameter

Um den Aufwand in Grenzen zu halten, werden die Messungen nicht mehr über alle identifizierten Parameter variiert, sondern nur noch in der bereits durch die erfolgten Messungen identifizierten optimalen Konfiguration. Das heißt, paravirtualisierende Treiber werden installiert, AMD-V wird aktiviert und kann von den Maschinen, die diese Unterstützung benötigen, verwendet werden. Als Architektur der Gastbetriebssysteme wurden der Einfachheit wegen nur 32-Bit Systeme gewählt. Die Virtualisierer werden wie folgt konfiguriert:

- Xen wird nur als Paravirtualisierer konfiguriert und benötigt daher zwangsweise einen Linux-Gast.
- Virtuozzo setzt auf Betriebssystemvirtualisierung und benötigt zwangsweise einen Windows Gast.

Um das Mischverhältnis der Gastbetriebssysteme ausgewogen zu halten und Vergleichsmöglichkeiten zu schaffen, wurden die verbleibenden Virtualisierer wie folgt mit Gastbetriebssystemen belegt:

- VMware's ESXi bekommt einen Linux Gast.
- Microsoft's Hyper-V bekommt einen Windows Gast.

Durchgeführte Messungen und deren Bewertung

Die Konfiguration der virtuellen Maschinen wurde gegenüber den Einzelmessungen nicht verändert, die Maschinen wurden lediglich in ihrer Anzahl verdreifacht um entsprechend parallel ausgeführte Tests durchführen zu können. Als Benchmarks werden die bereits bekannten und beschriebenen Tools eingesetzt und exemplarisch angelehnt an die Tabelle 3.2 parallel instanziiert. Der Wirkungsgrad der Virtualisierung könnte ermittelt werden, wenn anschließend vergleichbare Messungen auf einem einzelnen Hostsystem durchgeführt werden. Da die mehrfache Ausführung von zum Teil identischen Benchmarks auf einer einzelnen Maschine potentiell problembehaftet ist und in der Praxis keinen Wert zeigt, wird der Wirkungsgrad bezogen auf eine einzelne virtuelle Maschine ermittelt. Idealerweise ergibt sich bei zweifacher paralleler Ausführung der Benchmarks keine Laufzeitverlängerung, da die physischen Maschinen über zwei unabhängige Rechenkerne verfügen und daher beide virtuellen Maschinen echt parallel ausgeführt werden können. Bei drei parallel ausgeführten virtuellen Maschinen muss der Scheduler für eine gleichmäßige Verteilung sorgen. Die Laufzeit der einzelnen Tests verlängert sich hierbei idealerweise um maximal den Faktor $3/2$, so dass diese Zeit als Referenzzeit zum Einsatz kommt.

4.3.1 Nebenläufige Messungen an der Komponente CPU

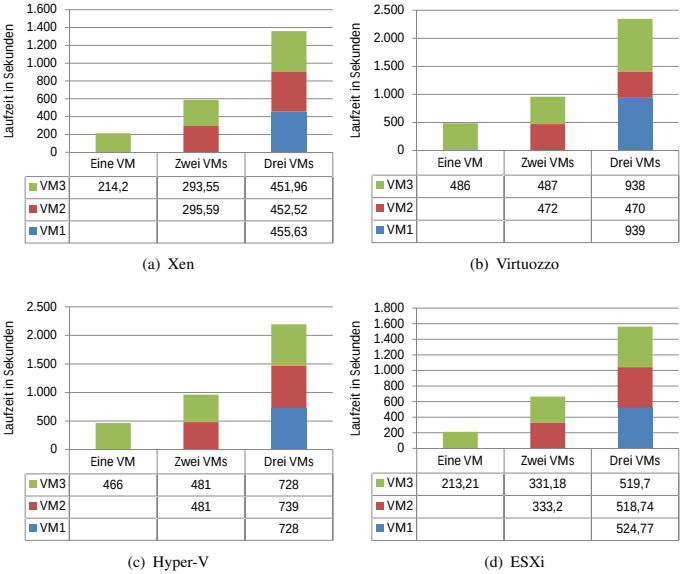


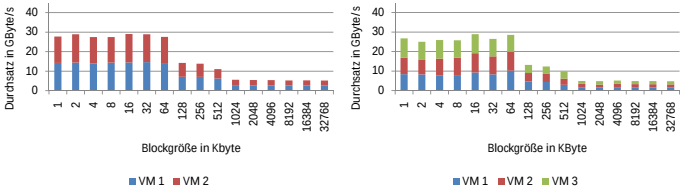
Abbildung 4.41: Nebenläufige CPU-Messungen

Scheduling von VMs zeigt Analogien zum Scheduling in Betriebssystemen

Zur Untersuchung von Effekten, die nebenläufige virtuelle Maschinen im Bereich CPU aufeinander besitzen, werden zwei beziehungsweise drei virtuelle Maschinen auf demselben Host instanziiert und der Linpack-Benchmark auf ihnen gestartet. Die Auswertung der Messergebnisse erfolgt anhand der Matrixgröße 4000, da Tests dieser Größe lange genug laufen und eine hinreichende Genauigkeit besitzen. Die erhaltenen Messergebnisse können den Abbildungen 4.41(a), 4.41(b), 4.41(c) und 4.41(d) entnommen werden. Die Laufzeit der einzelnen Benchmarks verteilt sich gleichmäßig an alle laufenden Instanzen, lediglich Virtuozzo besitzt eine statische Zuordnung von Maschinen auf physische Recheneinheiten, so dass bei drei aktiven virtuellen Maschinen eine Maschine schneller ist als die beiden übrigen. Die beiden Virtualisierer mit Windows-Gästen weisen einen sehr hohen durchschnittlichen Wirkungsgrad von 93% und höher auf, während die Linux VMs deutlich schlechter abschneiden. Das positive Verhalten der Windows-Gäste ist allerdings nicht zwangsweise den Virtualisierern zuzuschreiben, sondern könnte

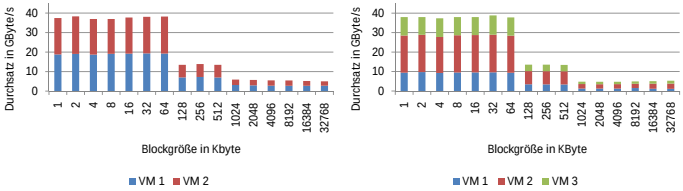
im Gastbetriebssystem zu suchen sein. Da die Windows-basierten Linpack-Läufe eine fast doppelt so lange Laufzeit benötigen wie die Linux-Variante, aber derselbe Quellcode zugrunde liegt, könnte der Grund für dieses Verhalten bei I/O-Waits zu suchen sein, die vom Virtualisierer genutzt werden, um in der Wartezeit andere virtuelle Maschinen auszuführen. Grundsätzlich ist als Trend erkennbar, dass mit steigender Anzahl virtueller Maschinen die Rechenzeit pro virtueller Maschine mehr als proportional abnimmt, da der Verwaltungsaufwand steigt. Der Wirkungsgrad fällt dementsprechend mit der Anzahl der parallel ausgeführten virtuellen Maschinen.

4.3.2 Nebenläufige Messungen an der Komponente RAM



(a) zwei nebenläufige virtuelle Maschinen (b) drei nebenläufige virtuelle Maschinen

Abbildung 4.42: Nebenläufige RAM/Cache-Messungen Xen

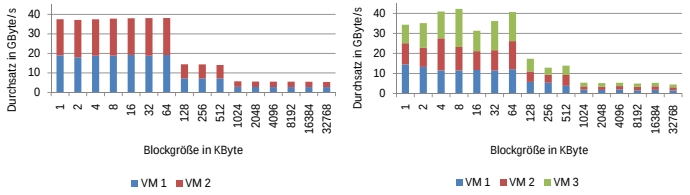


(a) zwei nebenläufige virtuelle Maschinen (b) drei nebenläufige virtuelle Maschinen

Abbildung 4.43: Nebenläufige RAM/Cache-Messungen Virtuozzo

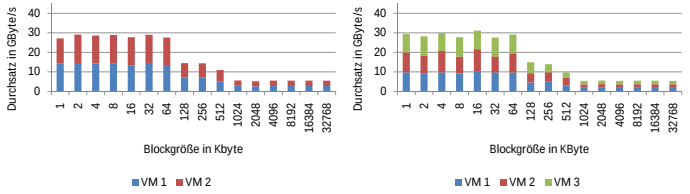
Ziel der Messung

Ziel dieser Messung ist es den Einfluss virtueller Maschinen auf die Leistungsfähigkeit des Caches beziehungsweise des Arbeitsspeichers einer nebenläufigen virtuellen Maschine zu untersuchen. Hierzu wurden zwei beziehungsweise



(a) zwei nebenläufige virtuelle Maschinen (b) drei nebenläufige virtuelle Maschinen

Abbildung 4.44: Nebenläufige RAM/Cache-Messungen Hyper-V



(a) zwei nebenläufige virtuelle Maschinen (b) drei nebenläufige virtuelle Maschinen

Abbildung 4.45: Nebenläufige RAM/Cache-Messungen ESXi

drei virtuelle Maschinen parallel instanziiert und führen den in 4.2.2 beschriebenen RAMspeed-Benchmark aus. Exemplarisch wird diese Messung am Beispiel des lesenden INTmark-Tests durchgeführt. Interessant ist in diesem Fall die Schwankung der Zugriffsraten der einzelnen Blockgrößen bei mehreren Maschinen. Die gewonnenen Messergebnisse werden durch die Abbildungen 4.42, 4.43, 4.44 und 4.45 dargestellt.

Zwei parallele Instanzen

Die Messungen mit zwei parallel aktiven, virtuellen Maschinen zeigen einen Wirkungsgrad von 97% - 100% bezogen auf eine einzelnen virtuelle Maschine. Dieser Wirkungsgrad ist unabhängig vom Hypervisor und in der Architektur des Speichersystems der Testmaschine begründet. Die AMD Direct Connect Architektur erlaubt echten parallelen Zugriff auf Arbeitsspeicher und Caches über Speichercontroller und Caches, die dediziert einzelnen Kernen zugeordnet sind. Damit ist eine echt parallele Ausführung der beiden virtuellen Maschinen möglich, die nur durch vereinzelte Hypervisor-Aktivitäten wie zum Beispiel dem Scheduling unterbrochen wird. Diese Hypervisor-Aktivitäten konnten in den Einzeltests auf dem freien Kern des Prozessors ausgeführt werden und beeinflussten die virtuellen Maschinen daher kaum. Bei zwei aktiven Maschinen muss das Scheduling auf denselben Kernen durchgeführt werden, auf denen auch die virtuellen Maschinen

aktiv sind. Auf diese Art kommt es zu minimalen Einbrüchen im Durchsatz und dem Wirkungsgrad der parallelen Instanzen, da die Caches durch den Kontextwechsel in den Hypervisor verschmutzt werden und Speicherinhalte zum Teil aus dem Hauptspeicher neu geladen werden müssen.

Drei parallele Instanzen

Bei drei nebenläufigen virtuellen Maschinen kann stets nur maximal zwei von ihnen durch den Hypervisor Zugriff auf die CPU gewährt werden. Eine virtuelle Maschine ist daher stets inaktiv und im Vergleich zu zwei parallel gemessenen Maschinen ist ein deutlich häufigeres Scheduling erforderlich, um gleiche Rechenzeit und quasi-parallele Aktivitäten der Maschinen gewährleisten zu können. Dies resultiert in einem Wirkungsgrad von nur noch 85%-90% bei Hyper-V und Xen und 95% bei ESXi und Virtuozzo, jeweils bezogen auf eine einzelne virtuelle Maschine des assoziierten Virtualisierers. Bei Virtuozzo fällt wie bereits bei den CPU-Messungen in 4.3.1 auf, dass das Mapping der virtuellen Maschinen auf physische Kerne statisch organisiert ist. Eine virtuelle Maschine läuft daher mit annähernd 100% Wirkungsgrad während die beiden anderen in der Performanz entsprechend deutlicher einbrechen. Die drei anderen getesteten Virtualisierer realisieren eine gleichmäßige Verteilung und erzielen damit auch gleichmäßige Wirkungsgrade für alle aktiven virtuellen Maschinen.

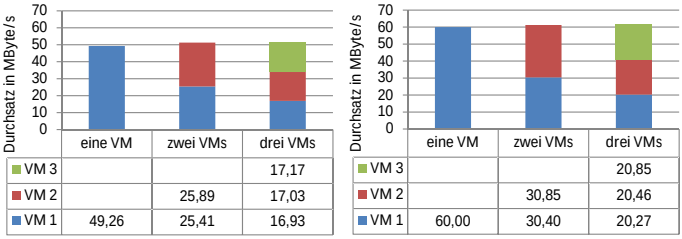
4.3.3 Nebenläufige Messungen an der Komponente Netz

Für Benchmarks nebenläufiger virtueller Maschinen im Bereich Netz wurden drei Szenarien ausgewählt. Im ersten Szenario (Abschnitt 4.3.3.1) wird der Durchsatz von zwei beziehungsweise drei parallel aktiven, virtuellen Maschinen mithilfe von Iometer getestet. Im zweiten Szenario (Abschnitt 4.3.3.2) wird die Platzierung der Kommunikationspartner untersucht. Im dritten Szenario (Abschnitt 4.3.3.3) wird der Netz-Durchsatz einer virtuellen Maschine mit zwei nebenläufigen, CPU-intensiven virtuellen Maschinen gemessen.

4.3.3.1 Netz x Netz

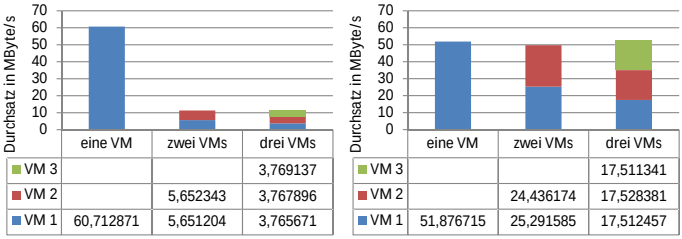
Konstante Gesamtleistung des Netz-Durchsatzes bei parallelen Instanzen

Die in 4.46, 4.47, 4.48 und 4.49 abgebildeten Messergebnisse zeigen den Durchsatz von zwei beziehungsweise drei virtuellen Maschinen, die jeweils synchron den bereits in den Einzelmessungen spezifizierten Iometer-Benchmark ausführen. Dabei ist erkennbar, dass der prozentuale Anteil aller parallelen Instanzen einer Messung sehr gleichmäßig verteilt ist und die Summe aller Durchsätze dem Durchsatz einer einzelnen virtuellen Maschine entspricht. Dies gilt sowohl im Lesen als auch im Schreiben. Lediglich Hyper-V und Virtuozzo zeigen lesend einen massiven Einbruch, wenn mehrere virtuelle Maschinen aktiv sind. Dies liegt vermutlich am Scheduling der Systeme, da eine virtuelle Maschine keine Daten empfangen kann, wenn sie nicht aktiv ist. Xen implementiert für diesen Fall ein Credit-basiertes



(a) Lesen (b) Schreiben

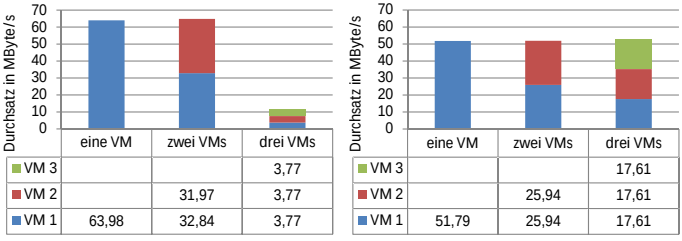
Abbildung 4.46: Nebenläufige Netz-Messungen Xen



(a) Lesen (b) Schreiben

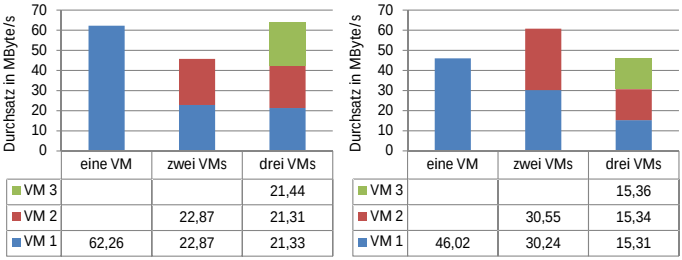
Abbildung 4.47: Nebenläufige Netz-Messungen Virtuozzo

Scheduling, das grundsätzlich auf Zeitscheiben basiert, jedoch einer virtuellen Maschine zusätzlich Credits einräumt, die für eintreffende I/O-Operationen zeitnah eingesetzt werden. Über diesen Mechanismus ist eine virtuelle Maschine beim Eintreffen von Daten über das Netz schnell aktiv und kann eventuell Pakete bestätigen, ehe zum Beispiel TCP-Übertragungsfenster zu klein werden und die Verbindung in ihrer Geschwindigkeit gedrosselt wird. VMware scheint ein ähnliches Konzept zu implementieren, da auch bei den Messungen am ESXi keine Einbrüche zu verzeichnen sind. Hyper-V und Virtuozzo implementieren solche Konzepte offensichtlich nicht und besitzen daher ein Problem beim Empfangen von Daten, wenn eine virtuelle Maschine nicht aktiv ist, weil die CPUs des Hosts durch andere virtuelle Maschinen oder Emulations-Routinen belegt ist. Ab einer gewissen Anzahl aktiver virtueller Maschinen kann jedoch auch dieses Verfahren nicht mehr greifen, da alle physisch vorhandenen Recheneinheiten ausgelastet sind.



(a) Lesen (b) Schreiben

Abbildung 4.48: Nebenläufige Netz-Messungen Hyper-V



(a) Lesen (b) Schreiben

Abbildung 4.49: Nebenläufige Netz-Messungen ESXi

4.3.3.2 Platzierung von Kommunikationspartnern

Kommunikation zwischen zwei virtuellen Maschinen

Ein weiterer Faktor, der den Netz-Durchsatz von virtuellen Maschinen beeinflussen kann, ist die Platzierung der kommunizierenden Maschinen in der Infrastruktur. Im Abschnitt 4.2.3 wurden bereits Messungen zwischen einer virtuellen und einer physischen Maschine durchgeführt. Dieses Szenario erfordert eine Emulation des gesamten Protokollstacks in der virtuellen Maschine und weitere Aktivitäten im Hypervisor, um die Pakete in das physische Netz zu routen oder switchen, während Pakete, die zwischen virtuellen Maschinen auf demselben Host ausgetauscht werden sollen, deutlich weniger Aktivität des Hypervisors benötigen und damit günstiger zu realisieren sind. Die abgebildeten Messergebnisse 4.50(a), 4.50(b), 4.50(c) und 4.50(d) verdeutlichen diesen Unterschied von virtuellen und physischen Kommunikationspartnern.

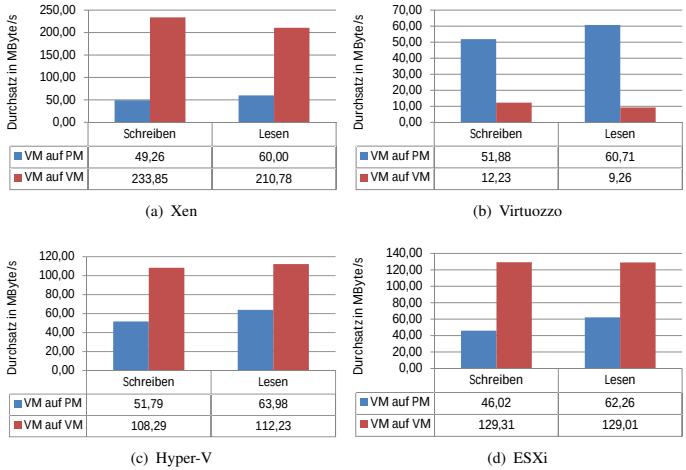


Abbildung 4.50: Kommunikationswege in virtuellen Netzen

Deutlicher Leistungsgewinn

Xen, Hyper-V und ESXi können ihre Durchsatzraten mit virtuellen gegenüber physischen Kommunikationspartnern um den Faktor zwei bis vier steigern und ihre Schreib- und Leseraten erreichen ein gleiches Niveau. Dies ist möglich, da statt der Emulation von Netzkomponenten einfach der Speicherinhalt, der die zu versendenden Daten enthält, der empfangenden virtuellen Maschine zugeordnet werden kann. Die zu versendenden Daten müssen damit nicht einmal kopiert werden. Xen erreicht mit dieser Technik Transferraten von über zwei Gigabit/s. Grundvoraussetzung für dieses Verfahren sind allerdings zwei Maschinen, die beide gleichzeitig aktiv sind. Im Beispiel von Virtuozzo ist dies offensichtlich nicht der Fall, da hier das Binden eines Containers an eine CPU wenig intelligent gelöst ist. Aus diesem Grund bricht die Übertragungsrate bei Virtuozzo in diesem Szenario sogar ein.

Kommunikation zwischen zwei virtuellen Maschinen über ein physisches Netz

Ein weiteres denkbares Szenario ist die Kommunikation von virtuellen Maschinen über ein physisches Netz mit zwei involvierten Hypervisor-Instanzen. Da die sendende virtuelle Maschine immer senden kann, solange sie aktiv ist und die empfangende virtuelle Maschine empfängt beziehungsweise solange der Hypervisor puffert, ist der beschränkende Faktor in diesem Szenario die Zeit, in der beide Maschinen aktiv sind beziehungsweise bis die Puffer des Hypervisors erschöpft ist. Implizit wird dieser Faktor durch die Anzahl der rechenbereiten Maschinen auf einem Host beeinflusst, da damit die zugeteilten Zeitscheiben an eine virtuelle Maschine seltener oder kürzer werden. Der maximale Durchsatz in diesem Szenario

ist daher der gleiche wie im Fall der Kommunikation mit einer physischen Maschine, tendenziell jedoch etwas geringer, da ein zweiter Virtualisierungs-Stack durchlaufen werden muss.

4.3.3.3 Netz x CPU

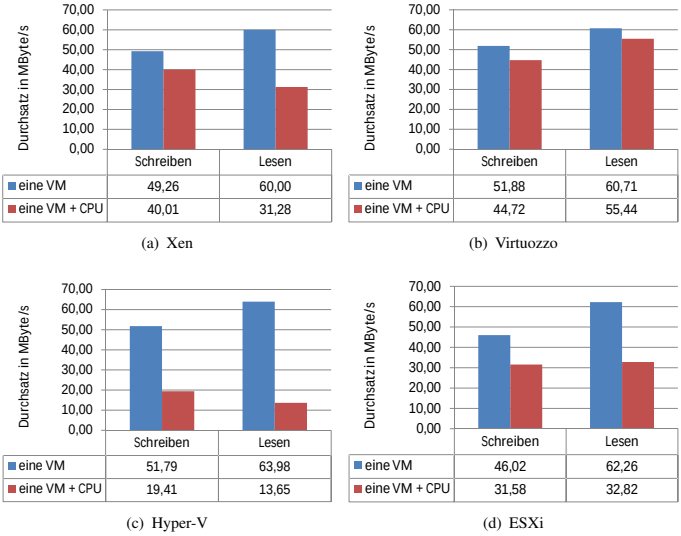


Abbildung 4.51: Netz-Messungen unter CPU-Last

Ziel der Messung

Die folgenden Messergebnisse in den Abbildungen 4.51(a), 4.51(b), 4.51(c) und 4.51(d) zeigen den Netz-Durchsatz einer virtuellen Maschine, während zeitgleich in zwei weiteren Maschinen CPU-Last erzeugt wird. Zur Messung des übertragenen Datenvolumens kommt Iometer zum Einsatz, während die CPU-Last durch den Linpack-Benchmark generiert wird. Das Ziel dieser Messung ist es herauszufinden, ob die CPU Last der beiden virtuellen Linpack-Benchmarks den Netz-Durchsatz der dritten Maschine schmälern können.

Netzvirtualisierung ist rechenintensiv

Betrachtet man die abgebildeten Messwerte der Maschine, deren Durchsatz mit Hilfe des Linpack-Benchmarks beeinflusst werden soll, so stellt man fest, dass sie verglichen mit einer einzelnen Maschine deutlich geringer ausfallen. Dies liegt zum einen an der gesunkenen Dauer, die der Maschine CPU-Zeit zugewiesen werden kann. Jedoch lässt sich ein Unterschied erkennen, ob es sich um einen

Vollvirtualisierer oder ein Paravirtualisierungsprodukt handelt. Durch die paravirtualisierende Netz-Schnittstelle ist weniger CPU-Leistung zur Emulation der Netz-Schnittstelle erforderlich als dies bei Vollvirtualisierern erforderlich wäre. Infolgedessen konkurriert die Übertragung von Netzdaten weniger um CPU mit den parallel laufenden Linpack-Tests und kann gegenüber Vollvirtualisierern einen höheren Durchsatz erzielen. Absolut gesehen sinkt der Schreib-Durchsatz bei Vollvirtualisierern um 30% bis 70%, während der Verlust bei Paravirtualisierern im Bereich 10% bis 20% liegt. Die Leserate fällt in diesem Szenario proportional stärker, da der virtuellen Maschine zum Empfangen proportional gesehen weniger Zeit zur Verfügung steht und das Empfangen eines definierten Volumens daher länger dauert.

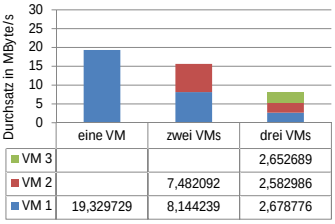
4.3.4 Nebenläufige Messungen an der Komponente Disk

Für Benchmarks nebenläufiger virtueller Maschinen im Bereich des Hintergrundspeichers wurden zwei Szenarien ausgewählt. Im ersten Szenario (Abschnitt 4.3.4.1) wird der Durchsatz von zwei beziehungsweise drei parallel aktiven, virtuellen Maschinen mithilfe von Iometer getestet, während im zweiten Szenario (Abschnitt 4.3.4.2) der Durchsatz einer virtuellen Maschine auf einen per iSCSI angebundenen, externen Speicher mit zwei nebenläufigen, CPU-intensiven virtuellen Maschinen untersucht wird.

4.3.4.1 Disk x Disk

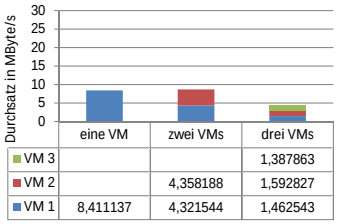
Schlechtes Leistungsverhalten von virtualisierten Hintergrundspeichern

Analog zu den Netz-Messungen in 4.3.3.1 wird in diesem Szenario das Verhalten mehrerer virtueller Maschinen untersucht, die mittels iSCSI gleichzeitig auf denselben Hintergrundspeicher zugreifen wollen. Die Iometer-Konfiguration ist identisch zu den Messungen im Abschnitt 4.2.4 gewählt. Betrachtet man die Messergebnisse in den Abbildungen 4.52, 4.53, 4.54 und 4.55, so findet man analog zu den Messungen in 4.2.4 kein durchgängiges Verhaltensmuster. Der Gesamt-Durchsatz orientiert sich grob am Durchsatz einer einzelnen Maschine, kann jedoch signifikant davon abweichen (siehe z.B. ESXi). Lediglich Virtuozzo zeigt ein deterministisches Verhalten, so dass der Gesamt-Durchsatz verlässlich im Bereich des Durchsatzes einer einzelnen Maschine liegt. Vermutlich liegt eine Ursache dieses Verhaltens darin, dass sequentielle Zugriffe auf den Speicher aus Sicht einer virtuellen Maschine sich aufgrund der Adressabbildung physisch nicht zwangsweise als sequentielle Zugriffe darstellen. Weiterhin können durch Optimierungen des iSCSI-Initiators, des zugrunde liegenden Raid-Systems und Native Command Queuing der Festplatten Optimierungen in der Reihenfolge gleichzeitiger Zugriffe vorgenommen werden, so dass Abweichungen vom erwarteten Standardverhalten durchaus möglich sind. Dies erklärt jedoch nicht die massiven



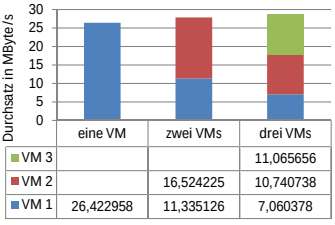
	eine VM	zwei VMs	drei VMs
VM 3			2,652689
VM 2		7,482092	2,582986
VM 1	19,329729	8,144239	2,678776

(a) nicht-sequentielles Lesen



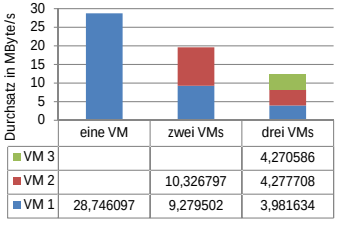
	eine VM	zwei VMs	drei VMs
VM 3			1,387863
VM 2		4,358188	1,592827
VM 1	8,411137	4,321544	1,462543

(b) nicht-sequentielles Schreiben



	eine VM	zwei VMs	drei VMs
VM 3			11,065656
VM 2		16,524225	10,740738
VM 1	26,422958	11,335126	7,060378

(c) sequentielles Lesen

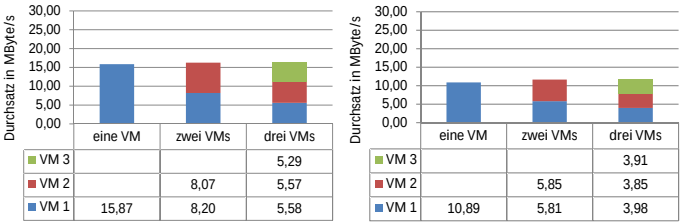


	eine VM	zwei VMs	drei VMs
VM 3			4,270586
VM 2		10,326797	4,277708
VM 1	28,746097	9,279502	3,981634

(d) sequentielles Schreiben

Abbildung 4.52: Nebenläufige Disk-Messungen Xen

Ausreißer nach oben hin und sollte eigentlich auch nicht zum Einbrechen des Durchsatzes einzelner Maschinen führen.



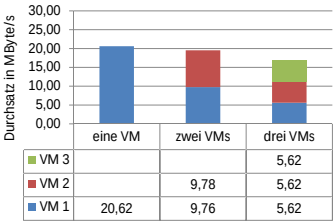
(a) nicht-sequentielles Lesen

(b) nicht-sequentielles Schreiben

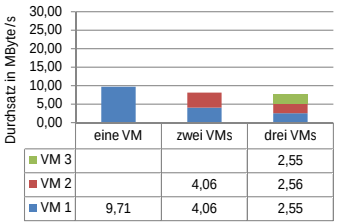
(c) sequentielles Lesen

(d) sequentielles Schreiben

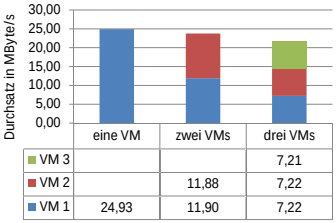
Abbildung 4.53: Nebenläufige Disk-Messungen Virtuozzo



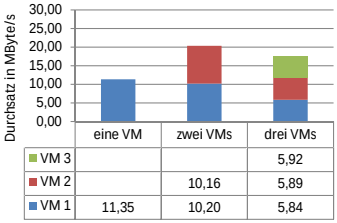
(a) nicht-sequentielles Lesen



(b) nicht-sequentielles Schreiben

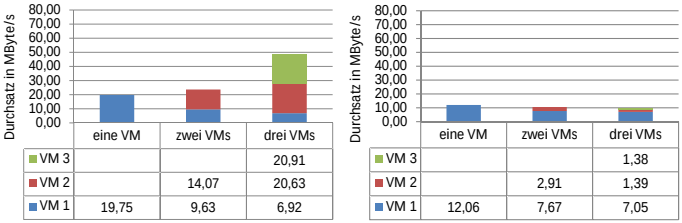


(c) sequentielles Lesen



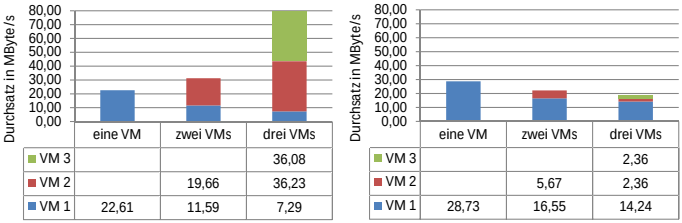
(d) sequentielles Schreiben

Abbildung 4.54: Nebenläufige Disk-Messungen Hyper-V



(a) nicht-sequentielles Lesen

(b) nicht-sequentielles Schreiben

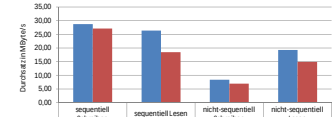


(c) sequentielles Lesen

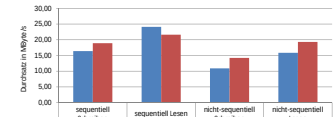
(d) sequentielles Schreiben

Abbildung 4.55: Nebenläufige Disk-Messungen ESXi

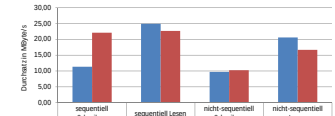
4.3.4.2 Disk x CPU



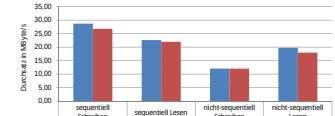
(a) Xen



(b) Virtuozzo



(c) Hyper-V



(d) ESXi

Abbildung 4.56: Disk-Messungen unter CPU-Last

Virtualisierung von Hintergrundspeichern ist CPU-freundlicher als Netzvirtualisierung

Analog zu den Messungen in 4.3.3.3 wird in diesem Szenario der Durchsatz einer virtuellen Maschine auf den entsprechenden Hintergrundspeicher untersucht, während die CPUs des Hostsystems von zwei weiteren virtuellen Maschinen beschäftigt werden. Die Ergebnisse der Messungen sind in den Abbildungen 4.56(a), 4.56(b), 4.56(c) und 4.56(d) zu sehen. Im Gegensatz zu den vergleichbaren Netz-Messungen ist hier die Abweichung zu den Tests einer einzelnen Maschine sehr viel geringer, woraus man folgern kann, dass die Emulation beziehungsweise Virtualisierung eines Disk-Controllers und der assoziierten, virtuellen Festplatte deutlich weniger Ressourcen benötigt als die Emulation von virtuellen Netzwerkkarten. Der Einfluss CPU-intensiver Anwendungen oder virtueller Maschinen auf Disk-I/O ist daher vernachlässigbar, wenn auch messbar.

4.4 Auswertung und Interpretation der Ergebnisse

Die in den beiden vorstehenden Teilkapiteln durchgeführten Messungen bestätigen im Wesentlichen die zuvor aufgestellten Theorien und quantifizieren sie. Aufgrund der Vielzahl an Virtualisierern und untersuchten Einflussfaktoren werden an dieser Stelle die wesentlichen Ergebnisse noch einmal zusammengefasst.

CPU

Der Wirkungsgrad der Virtualisierung von CPUs ist unabhängig von den getesteten Virtualisierungstechnologien - Vollvirtualisierung, Paravirtualisierung oder OS-Virtualisierung - sehr hoch. Werden mehrere CPU-lastige virtuelle Maschinen gleichzeitig betrieben, so kann die Leistung der virtuellen Maschinen überproportional fallen. Gemessen wurden bei drei Maschinen nur noch knapp 60% Leistung. Produzieren die virtuellen Maschinen viele I/O-Waits, so ist weiterhin eine gute Skalierung möglich, da der Hypervisor das Warten einer virtuellen Maschine zum Anlass nimmt, die CPU an eine andere virtuelle Instanz zu binden und die vorhandenen Ressourcen damit optimal genutzt werden können. Das Scheduling zeigt im Virtualisierungskontext daher analog zum Scheduling von Betriebssystemen ähnliche Eigenschaften. Als Tendenz kann beobachtet werden, dass der Wirkungsgrad der CPU-Virtualisierung mit steigender Anzahl an Maschinen abnimmt, sobald die Anzahl der physischen CPUs annähernd erreicht wird. In diesem Fall sind keine Reserven für den Hypervisor selbst mehr vorhanden, was zu Einbußen in der Performanz führt.

RAM/Cache

Der Wirkungsgrad von RAM- und Cache-Zugriffen ist bei einzelnen Maschinen sehr hoch. Caches können mit nahezu ungedrosselter Leistung auch in der virtuellen Maschine genutzt werden. Von den Adressübersetzungsstrategien, die beim Zugriff auf Arbeitsspeicher erforderlich sind, ist in den getesteten Szenarien kaum etwas zu spüren. Um sie zu bewerten, müssten künstliche Szenarien konstruiert werden, die gezielt sehr häufig Speicher allokieren und wieder frei geben. Dieses Verhalten ist allerdings unnatürlich und daher für die Praxis uninteressant. Die Leistung des Arbeitsspeichers ist konstant über die Anzahl der gleichzeitigen Zugriffe, da dank der Direct Connect Architektur jeder Kern einen eigenen Speicher-Controller besitzt. Bus-Architekturen würden hier voraussichtlich schlechter abschneiden. Caches leisten auch mit mehreren parallel aktiven Maschinen gute Arbeit, allerdings sind hier leichte Schwankungen im Durchsatz zu beobachten, die von wechselseitigem Verschmutzen des Caches durch andere virtuelle Maschinen oder dem Hypervisor selbst stammen. Ähnlich wie bei der CPU bricht die Leistung gering ein, wenn die Anzahl der rechenbereiten, virtuellen Maschinen größer als die Anzahl der vorhandenen physischen Kerne ist.

Netz

Der Wirkungsgrad von paravirtualisierten Netzen ist sehr gut, er liegt in der Regel bei 90% und höher, sofern der Kommunikationspartner ständig erreichbar ist. Emulierte Netze zeigen ein schlechtes Leistungsverhalten mit einem Wirkungsgrad in der Nähe von 25%, weshalb diese Technologie vermieden werden sollte.

Um dies zu garantieren müssen bei Vollvirtualisierern paravirtualisierende Treiber installiert werden. Die schreibende Gesamt-Leistung bleibt auch bei mehreren parallel aktiven virtuellen Maschinen konstant, die Übertragungsrate einer einzelnen Maschine fällt hierbei unabhängig vom Virtualisierer gleichmäßig auf allen Instanzen. Lesend können je nach Scheduler des VMMs gravierende Einbußen in der Übertragungsrate auf bis zu 5% entstehen. Diese Tatsache legt an der Inaktivität des virtuellen Kommunikationspartners und kann durch eine große Anzahl an virtuellen Maschinen, die um die CPU konkurrieren, weiter verschlechtert werden. Bei Platzierung der Kommunikationspartner auf einen Host sollte daher gleichzeitige Aktivität garantiert werden, was selbst bei einer kleinen Menge an virtuellen Maschinen nicht selbstverständlich ist, wie das Beispiel Virtuozzo zeigt. In diesem Fall erhöht sich die Übertragungsrate drastisch, da das Netz nicht mehr vollständig virtualisiert werden muss, sondern lediglich Verweise auf den Speicherbereich ausgetauscht werden müssen. Xen erreichte auf diese Art und Weise eine Übertragungsrate von über zwei GigaBit/s netto. Andere Virtualisierer lieferten ähnliche Ergebnisse. Ein zu beachtender Punkt ist, dass die Virtualisierung von Netzen sehr CPU-intensiv ist und daher potentiell Konfliktpotential mit CPU-intensiven virtuellen Maschinen besteht.

Disk

Der Wirkungsgrad von Zugriffen auf Hintergrundspeicher ist wenig performant. Der Durchsatz bei nebenläufigen Maschinen weist abhängig vom Szenario und Virtualisierer starke Schwankungen auf, einzig Virtuozzo als Repräsentant der Betriebssystemvirtualisierung produziert konstante Durchsatzraten in allen Szenarien. Ein Grund hierfür könnten technologische Vorteile der Betriebssystemvirtualisierung sein, die eine Adressübersetzung zum Zugriff auf den Hintergrundspeicher unnötig machen. Alle anderen Speichervirtualisierungstechnologien sind praxisuntauglich für Hochleistungssysteme. Die Virtualisierung des Hintergrundspeichers ist verhältnismäßig kostengünstig und kann weitestgehend problemlos in Kombination mit CPU-intensiven Anwendungen realisiert werden.

Hardware basierte Virtualisierungsunterstützung

Prozessorunterstützende Technologien konnten in den durchgeführten Einzelmessungen keine leistungssteigernden Effekte erzielen. Der theoretische Vorteil dieser Technologien liegt in der Verkürzung von Kontextwechseln zwischen virtuellen Maschinen und beziehungsweise oder dem Hypervisor. Dieser Kontextwechsel konnte seit der ersten Prozessorgeneration mit Virtualisierungsunterstützung drastisch verringert werden. Der für die Messung verwendete Prozessor gehört zu einer der ersten Generationen mit hardwareseitiger Unterstützung für Virtualisierungsfunktionen. Neuere Prozessoren können also durchaus positive Effekte auf das Leistungsverhalten von Virtualisierung besitzen. Der Nutzen dieser Technologie steigt mit der Anzahl der durchzuführenden Kontextwechsel, also mit der Anzahl der parallel aktiven virtuellen Instanzen, indem die Zeit für einen Kontextwechsel reduziert werden kann. Der Wirkungsgrad fällt bei Einsatz von unterstützenden Technologien mit steigender Anzahl virtueller Maschinen demnach trotzdem, allerdings weniger stark.

5 Existierende Optimierungsansätze

Inhalt

Optimierung von Technologien, Verteilungen und Konfigurationen

Seit der Wiedergeburt der Virtualisierung auf der PC-Architektur haben sowohl Hersteller von Virtualisierungsprodukten, freie Projekte als auch Wissenschaft daran gearbeitet, die Performanz und Funktionalität von Virtualisierungsansätzen zu erhöhen. Dabei existieren mehrere Ansätze, die alle ihre Berechtigung besitzen und unabhängig voneinander verfolgt werden können. Im Wesentlichen handelt es sich bei den Ansätzen um die Optimierung von Virtualisierungstechnologien, die Optimierung von Ressourcenverteilung und die Optimierung der Konfiguration. Der Ansatz dieser Arbeit ist letzterer Kategorie zuzuordnen, hat in seinen Ergebnissen aber durchaus Einfluss auf das Gebiet der Optimierung von Ressourcenverteilung. Im Folgenden werden einige konkrete Ansätze dieser Kategorien kurz erläutert.

5.1 Optimierung von Virtualisierungstechniken

Hybride Virtualisierungsprodukte

Seit der Entwicklung der VMware Workstation 1998 arbeitet der Konzern und seine Konkurrenten intensiv daran, Virtualisierungsprodukte für den Heim- und Serverbereich mit weiteren Funktionen auszustatten und deren Performanz zu verbessern. Dabei ist zu beobachten, dass konzeptionelle Verbesserungen eines Produktes oft in Produkten eines anderen Ansatzes übernommen werden. Ein Beispiel hierfür ist die Entwicklung von speziellen Treibern, die den Ansatz der Paravirtualisierung verfolgen. Damit wanderte eine Technik, deren Ursprung in User Mode Linux lag und durch Xen verbreitet wurde, in weitere Produkte, so dass heute kein Hersteller mehr Produkte eines einzigen Ansatzes vertreibt, sondern eine Mischform an Ansätzen entstanden ist. Ein weiterer großer Schritt zur Optimierung von Virtualisierungsperformanz lag in der Entwicklung entsprechender Befehlssätze für Prozessoren, so dass die PC-Architektur nach Goldberg virtualisierbar wurde. Neben diesen globalen Optimierungsansätzen wurden

auch Verbesserungen an einzelnen Implementierungen vorgenommen. Exemplarisch wird hier die Verbesserung von I/O Performance am Beispiel von Xen in Verbindung mit Intel-VT [ZhDo 08] genannt. Durch das Verschieben von Code, der aus dem Quemu Projekt zur Emulation von Netz- und Disk-Komponenten übernommen wurde, in den Hypervisor sowie den Einsatz von Event-Mechanismen statt regelmäßigem Polling reduzierte sich der Verlust zur Behandlung von VM-Exit Ereignissen im Hypervisor gravierend, so dass annähernd der Durchsatz eines nativen Systems erreicht werden kann. Der Grund hierfür liegt darin, dass die Behandlung der VM-Exit Ereignisse mittels des ursprünglichen Quemu-Emulators sehr viele CPU-Zyklen benötigt und diese Zeit aktiv gewartet wird, bis die Behandlung abgeschlossen ist. Um dieses Problem zu umgehen, wurde eine aus mehreren Threads bestehende Fehlerbehandlungsroutine implementiert, die während des eigentlichen I/O-Vorgangs im Hintergrund wartet und die Abarbeitung nebenläufiger Prozesse ermöglicht. Ein weiterer Ansatz im Bereich Netzoptimierung am Beispiel von Xen beschreiben Menon et al. in [MCZ 06]. Sie erweiterten die virtuelle Netzwerkkarte von Xen 2 um die Fähigkeit des TCP Segmentation offload (TSO). Der Vorteil dieser Methode besteht darin, dass der Aufwand, ein einzelnes Datenpaket von einer domU in die Driver Domain zu kopieren bzw. zu verlinken, gemessen am Datenvolumen, geringer wird, da in Summe weniger Pakete behandelt werden müssen. Desweiteren entwickelten sie Mechanismen, die es im Normalfall erlauben komplett auf ein Kopieren bzw. Verlinken der Daten im Sendefall zu verzichten. Zuguterletzt führen die Autoren übergroße Speicherseiten ein, so dass die Kapazitätsprobleme des TLBs im Falle einer großen Menge an virtuellen Maschinen kompensiert werden können und weniger Seitenfehler auftreten.

Optimierung von Netz-Durchsatz durch TCP Segmentation offload

5.2 Optimierung der Ressourcenverteilung

Effiziente Konsolidierung durch Binpacking

Ein weiterer Ansatz, Virtualisierungslösungen effizienter zu machen, ist, die Zuordnung von virtuellen auf physische Ressourcen zu optimieren. Für diesen Ansatz existieren mehrere Möglichkeiten, die sich in der ihnen zugrunde liegenden Metrik von Optimalität unterscheiden. So verfolgen Khanna et al in [KBKK 06] den Ansatz einer möglichst dichten Konsolidierung, das heißt möglichst viele virtuelle Maschinen auf möglichst wenig physische Rechner zu packen. Als Messkriterium postulieren sie eine optimal dichte Verteilung. Da diese praktisch nicht erreicht werden kann, da die benötigten Kosten zur Berechnung einer solchen exponentiell mit der Anzahl der virtuellen Maschinen wachsen, verzichten sie auf eine optimale Verteilung und begnügen sich statt dessen mit einer sehr guten Verteilung, die sich mit möglichst wenig Migrationsaufwand erreichen lässt. Als Anlass, eine Umverteilung der virtuellen Maschinen anzustoßen, dienen erkannte Quality of Service (QoS) Verletzungen. Eine virtuelle Maschine wie auch eine physische Ressource werden dabei als mehrdimensionale Vektoren betrachtet, die nach dem First-Fit Binpacking Algorithmus verteilt werden. Im Falle einer QoS Verletzung

in einer virtuellen Maschine wird eine Maschine desselben Hosts verschoben, um weitere Ressourcen auf der entsprechenden physischen Maschine zu akquirieren. Die Auswahl dieser virtuellen Maschine erfolgt basierend auf minimalen Migrationskosten. In einem Feldversuch mit drei physischen und elf virtuellen Maschinen waren mit dem vorgeschlagen Verfahren nur zwischen ein viertel und ein zwölftel der Anzahl an Migrationen notwendig, die im optimalen Fall benötigt wurden. Angaben, wie viel dichter die Verteilung in Simulationen gegenüber dem vorgestellten Algorithmus war, sind dem Papier nicht zu entnehmen. Ein Problem dieser Lösung dürften jedoch die Abhängigkeiten der einzelnen betrachteten Komponenten CPU, RAM, Hintergrundspeicher und Netz darstellen. Anders als im physischen Fall, wird für einen Zugriff auf den Hintergrundspeicher je nach Virtualisierungsansatz Rechenkapazität im Virtualisierer benötigt, um eine Übersetzung der gewünschten virtuellen Speicheradresse in eine Adresse des physischen Systems zu realisieren. Eine Betrachtung auf Ebene der einzelnen Komponenten ist daher nicht sinnvoll. Weiterhin sind all diese Komponenten abhängig vom Systembus der physischen Maschine, welcher durch seinen maximalen Durchsatz beschränkt sein kann, während einzelne Ressourcen, die an diesem Systembus angeschlossen sind, noch nicht am Ende ihrer Leistungsfähigkeit sind. Ein weiteres Problem dieses Verfahrens kann das Oszillieren von kleinen virtuellen Maschinen zwischen mehreren physischen Hosts sein, wenn Monitoringdaten nahe einem QoS Schwellwert schwanken.

Alternative Strategien für Live-Migration

Die Kosten einer Ressourcenumverteilung in Form von Migrationen ist ein weiterer Punkt, an dem zur Optimierung angesetzt werden kann. Haikun et al stellen hierfür in [LJL+ 09] ein neuartiges Verfahren zur Live-Migration von virtuellen Maschinen bereit. Anstatt rekursiv geänderte Seiten des Hauptspeichers einer virtuellen Maschine vom Quellsystem auf den Zielhost zu übertragen, wird anfangs ein Checkpoint des Quellsystems erstellt und übertragen. Anschließend werden Traces aller relevanten, ausgeführten Befehle auf dem Quellsystem erstellt und auf das Ziel angewendet. Der Vorteil dieses Verfahrens gegenüber traditionellen Verfahren ist ein deutlich gesunkener Bedarf an Netzbandbreite nach der Übertragung des Checkpoints, so dass Live-Migrationen voraussichtlich bald über schmalbandige Internetverbindungen hinweg ausgeführt werden können.

5.3 Optimierung der Konfiguration

Richtlinien zur effizienten Konfiguration

Eine sehr kostengünstige und für jeden Nutzer individuell durchführbare Maßnahme zur Erhöhung des Wirkungsgrades von virtualisierten Systemen und Infrastrukturen ist die Optimierung der Konfiguration dieser Systeme. Hierzu existieren einige Empfehlungen des Marktführers VMware, die jedoch nur qualitativer und nicht quantitativer Art sind und sich ausschließlich auf VMware Produkte beziehen. Ein Vergleich mehrerer Produkte ist daher nicht möglich. Das auf der VMworld 2008 vorgestellte Papier [MBT 08] gibt beispielsweise Empfehlungen zur Konfiguration des VMware ESX Servers. Darunter befinden sich allgemeine Aussagen,

wie die Empfehlung VMware Tools zu installieren und neuere Prozessoren zu verwenden, da sich seit Erscheinen der ersten VT-x Befehlssätze von Intel die VMexit-Latenz bis zur heutigen Generation gesechstelt hat. Konkreter werden die Empfehlungen im Bereich der Netzkonfiguration. VMware empfiehlt Netzwerkkarten mit Unterstützung für TSO, da diese, wie im Abschnitt 5.1 schon erläutert, die Anzahl der Kontextwechsel zwischen Hypervisor und virtueller Maschine reduzieren können und somit den Netz-Durchsatz deutlich steigern. Ebenso empfiehlt VMware den Einsatz von Large Pages, da ein einzelner Eintrag im TLB dann größere Speicherbereiche abdeckt und es seltener zu TLB Misses kommt. Da in [MCZ 06] identische Aussagen auch für Xen gemacht werden, kann davon ausgegangen werden, dass dies eine generelle Empfehlung für virtuelle Systeme darstellt. Kritischer muss hingegen die Empfehlung betrachtet werden, Hardwareunterstützung würde in beinahe allen Fällen zur Performanzsteigerung beitragen. Wie die Messungen zeigen, ist dies nicht der Fall. In [VMwa 07] werden weiterhin Aussagen über die Optimierung von virtuellen Infrastrukturen getroffen. So sollen etwa Maschinen, die viel miteinander kommunizieren, auf einem Host gruppiert werden, da die Latenzen dann geringer seien. Bei hoher I/O-Last hingegen sollten virtuelle Maschinen auf mehrere physische Server verteilt werden, um keinen Flaschenhals in der zugrundeliegenden Hardware zu produzieren. In [Doro 07] werden Details zum Scheduling-Mechanismus des ESX Servers erläutert und der Hinweis gegeben, dass, falls eine virtuelle Maschine weniger virtuelle CPUs als physisch vorhandene CPUs besitzt und die Maschine nicht statisch an bestimmte Kerne oder CPUs gebunden ist, automatisch ein Mapping an einen sogenannten Home Node vorgenommen wird, um kürzere Zugriffszeiten auf assoziierte Speicherbereiche bei Verwenden einer Non-Uniform Memory Architecture (NUMA) garantieren zu können. Generell wird empfohlen, so wenig wie möglich virtuelle CPUs in eine virtuelle Maschine zu konfigurieren, auf keinen Fall aber mehr als physisch vorhanden sind, da dies den Scheduling-Aufwand erhöht. Weiterhin wird der Einsatz von 64-Bit Software und Betriebssystemen empfohlen, da diese einen Geschwindigkeitsvorteil gegenüber 32-Bit Code bringen würde. Konkret mit Messwerten wird jedoch keine dieser Aussagen belegt.

Anleitung zu eigenen Messungen

In [VMwa 06] gibt VMware weiterhin Hilfe zur Selbsthilfe, indem Hilfestellungen gegeben werden, wie individuell Messungen vorgenommen werden können und wie dabei Fehler vermieden werden können. Die gemachten Aussagen beziehen sich konkret auf Benchmarks mit der VMware Workstation 5.5, so dass das Papier wohl eher Laien ansprechen soll. Kein objektiv ausgelegtes Messverfahren würde mehr als einen Parameter zwischen zwei Messungen verändern, da damit keine konkreten Aussagen über die Ursache der Veränderung der beiden Messwerte getroffen werden könnte. Hilfreich hingegen ist der Hinweis, sich nicht auf Zeitmessungen innerhalb der virtuellen Maschine zu verlassen, da die durchgereichten Counter nicht hundertprozentig korrekt synchronisiert werden können. Dieses Verfahren wird jedoch von fast allen Benchmark-Sammlungen eingesetzt, so dass eine objektive Messung nur mit angepassten Benchmarks erfolgen kann.

5.4 Bewertung

Koordinierter Ansatz notwendig

Alle genannten Ansätze sind an sich legitim und notwendig, um Virtualisierungsansätze, virtuelle Systeme und virtuelle Infrastrukturen zu optimieren. Da all die vorgestellten Ansätze und Maßnahmen nicht koordiniert durchgeführt wurden und deren Ergebnisse nicht zentral vorliegen, ist es ein sehr zeitaufwändiges bis unmögliches Unterfangen jeden einzelnen Ansatz qualifiziert zu bewerten und zu vergleichen. Wie bereits anfangs erläutert, existieren im übertragenen Sinn viele Puzzleteile, die aufgrund unterschiedlicher Messverfahren und Bedingungen für Performanz nicht zu einem großen Puzzle zusammengesetzt werden können, da ihre Teile zu verschiedenen Puzzles gehören. Die in dieser Arbeit entwickelten Messverfahren und durchgeführten Messungen legen die Grundlage, für eine standardisierte Herangehensweise, die Vergleiche zwischen einzelnen Messungen zulässt und es ermöglicht das Puzzle mit weiteren Messungen, die im Rahmen dieser Arbeit nicht durchgeführt werden konnten oder deren Bedarf sich erst zukünftig ergib, zu erweitern. Insbesondere den Ansätzen von VMware fehlt es noch an quantitativen Aussagen, so dass der Wert der einzelnen vorgeschlagenen Empfehlungen individuell erkennbar ist und für jede Situation eine Kosten-Nutzen Analyse durchgeführt werden kann.

6 Möglichkeiten zur Erhöhung des Wirkungsgrades

Abgeleitete Empfehlungen zur Steigerung der Effizienz

Inhalt

6.1 Ansätze

In diesem Kapitel werden Möglichkeiten aufgezeigt, wie der Wirkungsgrad virtueller Maschinen im Kleinen beziehungsweise virtueller Infrastrukturen im Großen gesteigert werden kann. Hierzu ist es notwendig sich noch einmal die in Kapitel 3 aufgezeigten Metriken, nach denen optimiert werden soll, zu verinnerlichen. Konkret wurden dort die Metriken

- Ausführungszeit
- Antwortzeit
- Durchsatz

erwähnt. Für die durchgeführten Messungen an RAM, Netz und Disk wurde in dieser Arbeit auf die Metrik des Durchsatzes zurückgegriffen, während zur Bestimmung der CPU-Leistung durch den Linpack-Benchmark auf die Antwortzeit gesetzt wurde. Auch der Begriff des Wirkungsgrades, wie er in dieser Arbeit aufgestellt wurde, basiert auf diesen Metriken. Bei allen Aktivitäten, die nachfolgend beschrieben werden und der Erhöhung des Wirkungsgrades dienen, muss beachtet werden, dass die Optimierung an sich zum Teil kostenbehaftet ist, also Ressourcen für ihre Berechnung und Realisierung benötigt werden. Im Falle der statischen Optimierung (siehe Abschnitt 6.1.1) sind das Ressourcen in Form von Aktionen, die ein Administrator ausführen muss, im dynamischen Fall (siehe Abschnitt 6.1.2) können diese Ressourcen auch technologischer Form sein, zum Beispiel Netzbandbreite für die Live-Migration virtueller Maschinen oder der Berechnung von Optimierungsfunktionen. Der Wirkungsgrad eines Systems steigt jedoch erst dann wirtschaftlich, wenn die Optimierung mehr Leistung freisetzt als sie Ressourcen kostet. Die Bewertung, wann dieser Punkt erreicht ist, muss von Fall zu Fall individuell definiert werden, eine pauschale Bewertung insbesondere im statischen Fall ist nicht möglich. Im dynamischen Fall wird dieser Aspekt im Zuge der Automatisierbarkeit der Realisierung der vorgeschlagenen Richtlinien erneut aufgegriffen und allgemein berücksichtigt, jedoch nur exemplarisch mit Werten belegt.

6.1.1 Statische Optimierung

Richtlinien für Planung einer virtuellen Infrastruktur

Anhand der durchgeführten Analysen und Messungen lässt sich ein Katalog von Empfehlungen ableiten, der die Planung von neuen virtuellen Infrastrukturen erleichtert und Flaschenhälse in der Architektur vermeidet. Weiterhin können mit den gewonnenen Erkenntnissen Anwendungsklassen identifiziert werden, welche effizient virtualisierbar sind beziehungsweise welche sich nicht leistungserhaltend virtualisieren lassen. Ein weiterer Punkt, der bei der Planung von virtuellen Infrastrukturen ein bisher nicht ohne weiteres lösbares Problem darstellte, war die Dimensionierung der erforderlichen physischen Ressourcen. Die Messungen in

dieser Arbeit können dieses Problem nicht vollständig lösen, jedoch können sie eine grobe Tendenz des Verhaltens von Virtualisierern erkennen lassen, die eine Abschätzung des Ressourcenbedarfs erleichtert. Die folgenden Absätze fassen die Erkenntnisse dieser Arbeit unter dem Blickwinkel der für den Aufbau einer virtuellen Infrastruktur notwendigen Aspekte zusammen. Die Gliederung erfolgt nach den Komponenten CPU, RAM, Netz und Disk, welche auch die Gliederung innerhalb dieser Arbeit darstellen.

CPU

Der Wirkungsgrad von einzelnen CPU-lastigen virtuellen Maschinen ist sehr hoch. Er beträgt annähernd 100% unabhängig von der Virtualisierungsart. Virtualisierungstechnologien sind damit in der Lage, CPU-intensive virtuelle Maschinen effizient zu betreiben. Diese Erkenntnis ermöglicht zum Beispiel einen Einsatz von Virtualisierung in ausgewählten Bereichen des High Performance Computings. Virtualisierung wird bis heute in diesem Bereich mit der Begründung abgelehnt, sie sei nicht effizient genug. Auch wenn Virtualisierung einen minimalen Leistungsverlust verursacht, so gewinnt man durch die Abstraktion von Hardware Management-Vorteile, die einfachere Failover und Hochverfügbarkeitsmaßnahmen ermöglichen oder flexibleres Staging ermöglichen, da zur Berechnung benötigte Bibliotheken nicht mehr systemweit vorinstalliert werden müssen, sondern vom Ersteller des Jobs nach Bedarf in den auszuführenden virtuellen Maschinen und in der gewünschten Version installiert werden können. Beim Betrieb mehrerer virtueller Maschinen auf einem physischen System ist darauf zu achten, dass die Summe der benötigten Computing-Ressourcen kleiner ist als die physisch vorhandene Kapazität, da ansonsten Ressourcen fehlen, die der Hypervisor benötigt, um Scheduling und Virtualisierungsleistung zu erbringen, welche einen nicht zu vernachlässigen Anteil darstellen können. Missachtet man diese Regel, so kann der Wirkungsgrad der CPU-Virtualisierung auf 60% oder weniger fallen.

RAM/Cache

Der Zugriff auf RAM und Cache aus einer virtuellen Maschine heraus erweist sich ebenfalls als performant. Entscheidend für das Leistungsverhalten ist es hier, eine Systemarchitektur des physischen Systems analog zur Direct Connect Architektur zu wählen. Je mehr voneinander unabhängige Speicherbusse und Caches existieren, desto höher ist der zu erwartende Durchsatz in der Praxis. Reine Bus-Architekturen mit nur einem Memory-Controller sind zu vermeiden. Werden mehr virtuelle Maschinen gleichzeitig betrieben als physisch Prozessoren oder Kerne vorhanden sind, so beginnt der Hypervisor mit verstärktem Scheduling, was sich in verschmutzten Caches niederschlägt. Je größer die Anzahl an virtuellen Maschinen im Verhältnis zur Anzahl der vorhandenen Prozessoren ist, desto häufiger wird eine einzelne virtuelle Maschine aktiviert beziehungsweise pausiert und ihre Daten werden aus dem Cache verdrängt. Die Folgen reichen von leichten Schwankungen im Durchsatz auf Cache-Ebene bei wenigen parallel betriebenen virtuellen Maschinen bis hinzu massiven Performanzeinbußen. Bei drei aktiven Maschinen auf zwei vorhandenen physischen Rechenkernen wurden bereits 15% Leistungsverlust gemessen. Da die Caches moderner Rechner direkt in die Prozessoren integriert sind, können Caches und Prozessoren nur gemeinsam in ihrer Anzahl dimensioniert werden. Für die Dimensionierung der Größe des Arbeitsspeichers gilt, dass die

Summe des von virtuellen Maschinen benötigten Arbeitsspeichers auch physisch zur Verfügung stehen muss. Auf Auslagerungsdateien in virtuellen Maschinen auf Hintergrundspeicher sollte verzichtet werden, da durch die zusätzliche Virtualisierungsschicht der Zugriff auf den ausgelagerten Speicher gegenüber der Auslagerungsdatei einer physischen Maschine noch einmal deutlich verlangsamt wird. Auch der Hypervisor selbst benötigt einen kleinen Teil des Hauptspeichers, um Adressübersetzungstabellen zu speichern und sich selbst im Arbeitsspeicher halten zu können.

Netz

Die Virtualisierung von Netzkomponenten ist eine der teuersten Operationen, unabhängig davon, welche Virtualisierungstechnologie eingesetzt wird. Emulierte Netzwerkkarten, wie sie etwa von Xen in der vollvirtualisierten Variante eingesetzt werden, besitzen zudem einen schlechten Wirkungsgrad. Der Wirkungsgrad paravirtualisierter Netzwerkkarten ist mit zwischen 80% und 100% deutlich besser. Der Betrieb dieser virtuellen Hardware benötigt jedoch nicht zu vernachlässigende CPU-Leistung, selbst wenn paravirtualisierende Treiber im Gastbetriebssystem der virtuellen Maschine installiert wurden. Am Beispiel VMware ESXi wurden 50% CPU bei schreibenden Zugriffen und 150% CPU-Auslastung bei lesenden Zugriffen gemessen. Stichproben bei anderen Produkten bestätigen die Größenordnung des Berechnungsaufwands. Im Parallelbetrieb teilt sich die verfügbare Netzbandbreite nach außen gleichmäßig auf alle virtuellen Maschinen auf. Dies gilt für alle Virtualisierer bei schreibenden Netzzugriffen. Das Empfangen von Daten liegt abhängig von der Implementierung des Schedulers ebenfalls in dieser Größenordnung oder kann auf 5% des möglichen Durchsatzes einbrechen, wenn der Scheduler die empfangende virtuelle Maschine nicht zeitnah aktiviert, damit die ankommenden Pakete entgegengenommen und, falls nötig, bestätigt werden können. Hierzu ist es erforderlich, dass der Hypervisor selbst stets aktiv ist und nicht durch Ressourcenengpässe gezwungen wird, alle verfügbaren Recheneinheiten an virtuelle Maschinen abzutreten. Bei hoher geplanter Netzaktivität sollte daher an freie CPU-Ressourcen gedacht werden, da andernfalls der Durchsatz beeinträchtigt werden kann und damit zwangsläufig verbunden die Latenz der virtuellen Netzverbindung steigt. Aufgrund der Tatsache, dass Dienste, die von virtuellen Maschinen erbracht werden, nahezu ausschließlich über das Netz bezogen werden können, sollte, um Dienstgüte garantieren zu können, eine gewisse Echtzeit-Fähigkeit oder zumindest nahezu Echzeitfähigkeiten gewährleisten können. Aufgrund der Problematik beim Empfangen von Daten über das Netz ist von der Virtualisierung von Anwendungen abzuraten, die regelmäßig größere Datenvolumen über das Netz empfangen. Hierzu gehören unter anderem Router oder Streaming-Clients. Von der nativ nutzbaren Bandbreite kann aufgrund des begrenzten Wirkungsgrades der Netzvirtualisierung nur 80% bis 100% verwendet werden. Dies ist bei der Kapazitätsplanung zu berücksichtigen.

Disk

Der Wirkungsgrad der Virtualisierung von Hintergrundspeicher schwankt je nach Anwendungsszenario stark. Es existiert kein echter Virtualisierer, der in den durchgeführten Tests bestehend aus den Kombinationen sequentielles und nichtsequentielles Lesen und Schreiben konstant gute Leitungen erzielt. Lediglich

Virtuozzo erzielt hier durchwegs gute Ergebnisse, was dadurch zu erklären ist, dass Virtuozzo keine echte Virtualisierung des Hintergrundspeichers implementiert, sondern lediglich Quota überwacht und den einzelnen virtuellen Maschinen Zugriffsrechte auf Teilbäume der Verzeichnisstruktur des Host-Betriebssystems einräumt. Da der Hintergrundspeicher ein essentieller Bestandteil von Serversystemen ist, benötigen virtuelle Maschinen entsprechende Möglichkeiten zum Zugriff auf Hintergrundspeicher. Dies stellt in der Praxis solange kein Problem dar, solange der Hintergrundspeicher für das Ablegen des Betriebssystems und dessen Anwendungen genutzt wird. Von der Virtualisierung von Anwendungen, die massiven Gebrauch von Hintergrundspeicher machen, ist abzuraten. Hierunter fallen unter anderem File-Server und Datenbank-Server. Kapazitätsplanungen bezüglich Durchsatz sind aus diesem Grunde für Hintergrundspeicher eher unwichtig, da virtuelle Maschinen nach dem Start des Betriebssystems und dem Laden ihrer Anwendungen und Dienste nur noch sporadisch Zugriff auf den Hintergrundspeicher benötigen. Das Anlegen von Auslagerungsdateien auf Hintergrundspeicher sollte aus analogen Gründen ebenfalls unterlassen werden. Stattdessen sollte einer virtuellen Maschine genug Arbeitsspeicher zugewiesen werden. Der Aufwand beim Zugriff auf eine virtuelle Disk unter VMware ESXi liegt bei ungefähr 25% CPU-Auslastung bei Verwendung eines Software-basierten iSCSI-Initiators unabhängig davon, ob gelesen oder geschrieben wird. Wird stattdessen ein Hardware-basierter iSCSI-Initiator eingesetzt, so sinkt die CPU-Last gegen Null. Bei der Planung einer virtuellen Infrastruktur sollte daher abgewägt werden, ob sich bei hoher Disk-I/O-Rate die Investition in HBAs lohnt, oder ob stattdessen die Anzahl der Rechenkerne erhöht werden sollte. Da viele Virtualisierungslösungen aktuell Lizensierungsmodelle auf Sockelbasis verwenden, könnte eine Investition in HBAs trotz des recht hohen Preises in Summe günstiger ausfallen.

Übersicht

Die Tabelle 6.1 fasst die im Laufe der Arbeit ermittelten Wirkungsgrade der untersuchten Komponenten noch einmal zusammen. Die Spalte Typ beschreibt hierbei die Art der Virtualisierungstechnik, die für die Virtualisierung der assoziierten Komponente aufgrund ihrees Wirkungsgrades bevorzugt verwendet werden sollte. Die Spalte Aufwand zeigt die zur Virtualisierung der entsprechenden Komponente notwendigen Ressourcen in Form von CPU-Anteilen, wenn die virtualisierte Ressource unter volle Last gesetzt wird. Die Wirkungsgrade beziehen sich auf die im Idealfall erreichbaren Werte. Tatsächliche Werte können massiv geringer ausfallen, wenn das physische System an seine Lastgrenze hinsichtlich verfügbarer Rechenressourcen stößt.

	Wirkungsgrad	Typ	Aufwand
CPU	bis zu 100%	alle	kaum
RAM	bis zu 100%	alle	kaum
Cache	bis zu 100%	alle	Null
Netz	80%-100%	Paravirt.	Schreiben 50% Lesen 150%
Disk **mit HBA**	stark schwankend stark schwankend	alle alle	25% kaum

Tabelle 6.1: Übersicht über den Wirkungsgrad von virtualisierten Komponenten

6.1.2 Dynamische Optimierung

Im Gegensatz zur statischen Optimierung, können dynamische Optimierungsansätze nicht zur Entwurfszeit realisiert werden, sondern müssen zur Laufzeit umgesetzt werden. Entscheidungen, welche die Art der Virtualisierungstechnik, die Vernetzung oder die Ausstattung physischer Server betreffen, sind damit nicht mehr kurzfristig beeinflussbar. Der einzige Ansatz, der zur Laufzeit Einfluss auf die Effizienz von virtuellen Infrastrukturen besitzen kann, ist daher eine effiziente Strategie der Ressourcen-Zuweisung. Die Entscheidung über eine Ressourcen-Zuweisung an eine bestimmte virtuelle Instanz kann sowohl lokal als auch global von Bedeutung sein, so dass sich die nächsten beiden Abschnitte mit den lokalen und globalen Möglichkeiten des Ressourcen-Schedulings auseinandersetzen.

6.1.2.1 Lokale Optimierung

Erschöpftes Optimierungspotential

Ansätze für lokale, das heißt auf einen Server beschränkte Ansätze zur Optimierung der Ressourcenverteilung sind in beinahe allen Virtualisierungslösungen integriert. So lassen sich etwa Hauptspeicherkontingente flexibel an virtuelle Maschinen zuteilen. Im Bedarfsfall kann durch diese Technik der Hauptspeicher einer virtuellen Maschine bis zu einem angegebenen Maximalwert erhöht oder bis zu einem Minimalwert verringert werden. Freiwerdende Ressourcen können so den kurzzeitigen Ressourcenbedarf anderer Instanzen überbrücken. Weiterhin lassen sich Gewichte für virtuelle Maschinen spezifizieren, die beeinflussen, wie viel Rechenzeit einer virtuellen Maschine zugewiesen wird. Somit ist es möglich wichtige Maschinen und Dienste bei der Vergabe von Rechenzeit bevorzugt zu behandeln und deren Verfügbarkeit zu gewährleisten. Auch das Abbilden einer virtuellen CPU auf die günstigste physische CPU im Sinne des Zugriffs auf Arbeitsspeicher bei NUMA-Architekturen beherrschen manche VMware-Produkte inzwischen. Die Ressourcen eines einzelnen Servers werden daher bei nahezu allen Virtualisierern unabhängig von deren Wirkungsgrad effizient genutzt. Optimierungspotential besteht hier nur noch global gesehen. Ein jedoch noch existierendes Problem, das auch in den Messungen ersichtlich wird, betrifft das

Scheduling. Treffen Netzpakete für eine gerade nicht aktive virtuelle Maschine ein, so dauert deren Zustellung zum Teil so lange, bis die Maschine wieder Rechenzeit zugewiesen bekommt. Diese Phase vom Eintreffen der Pakete auf der physischen Maschine bis zu ihrer Verarbeitung erhöht die Latenz des Netzverkehrs enorm und der Durchsatz bricht ein. Hier müssen viele Produkte noch Nachbesserungen an ihrem Scheduling leisten. Bis zur Implementierung dieser Nachbesserungen müssen Workarounds über statische Konfigurationen realisiert werden, um solche Situationen zu vermeiden. Dazu müssen virtuelle Maschinen nötigenfalls exklusiv an einzelne Kerne gebunden oder ihre Priorität erhöht werden. Eine automatische Erkennung entsprechender Situationen durch den Hypervisor selbst wäre wünschenswert, denn dieser könnte selbst passende Gegenmaßnahmen einleiten.

6.1.2.2 Globale Optimierung

Separieren von virtuellen Maschinen

Im Gegenzug zur lokalen Optimierung existieren bislang keine Empfehlungen, welche Typen von virtuellen Maschinen auf einem Hostsystem effizienterweise kombiniert werden können und wie in der Folge eine optimale globale Verteilung aller virtuellen Maschinen auf physische Systeme bestimmt werden kann. Im Abschnitt 6.1.1 wurde bereits erläutert, dass es ein wichtiges Kriterium für die Verteilung von virtuellen Maschinen ist, dass nicht zu viele CPU-intensive Anwendungen auf einem Hostsystem ausgeführt werden, sondern die Last gleichmäßig auf alle Systeme verteilt wird. Im Extremfall können alle CPU-intensiven virtuellen Maschinen auf unterschiedliche physische Systeme separiert werden, wenn die Anzahl der CPU-intensiven virtuellen Maschinen kleiner als die Anzahl der physischen Hosts ist. Anderenfalls sollten die Maschinen so verteilt werden, dass die auf den physischen Systemen induzierte Last gleichmäßig verteilt ist. Weiterhin sollten virtuelle Maschinen mit viel Netz-I/O von Maschinen mit CPU-Last getrennt werden, da auch die Virtualisierung von Netzwerkkarten CPU-intensiv ist und diese mit den rechenintensiven Maschinen in direkte Konkurrenz treten. Der Netz-Durchsatz dieser Instanzen kann anderenfalls beeinträchtigt werden. Eine sinnvolle Strategie scheint es daher zu sein, einen Kern des Hosts nicht durch virtuelle Maschinen zu belegen, sondern diesen für den Hypervisor und I/O-Virtualisierung zu reservieren.

Gruppieren von virtuellen Maschinen

Weiterhin sind Kommunikationsbeziehungen zwischen virtuellen Maschinen bislang nicht in der Verteilung auf physische Systeme berücksichtigt. So können virtuelle Maschinen, die auf demselben physischen System ausgeführt werden, im Idealfall über gemeinsam genutzten Speicher kommunizieren. Eine Abbildung der virtuellen auf physische Netzkarten ist in diesem Fall nicht notwendig und spart somit zum einen Ressourcen, die andernfalls für die komplette Virtualisierung des I/O-Vorganges notwendig gewesen wären, andererseits sinkt die Latenz des Netzverkehrs und der Durchsatz kann erhöht werden. Ein weiterer Vorteil hierbei ist, dass der Scheduler des Virtualisierers eine virtuelle Maschine aktivieren kann,

sobald Pakete empfangen werden können. Bei mehreren beteiligten Hypervisor-Instanzen auf mehreren Servern wäre hierfür eine Synchronisation über das Netz notwendig, die selbst gewissen Latenzen unterliegen würde und daher nur begrenzt hilfreich wäre. Kommunikationsbeziehungen zwischen zwei virtuellen Maschinen sollten also dazu führen, dass die beteiligten virtuellen Maschinen gruppiert auf einem einzelnen Virtualisierer ausgeführt werden.

Binden virtueller Maschinen an physische Systeme

Neben dem Gruppieren und Separieren von virtuellen Maschinen können in einzelnen Fällen weitere Maßnahmen sinnvoll erscheinen, um die Performanz zu erhöhen oder andere Funktionalitäten zu gewährleisten. So kann es etwa sinnvoll sein, einige virtuelle Maschinen fest an ausgewählte physische Systeme zu binden, da ihre Migrationskosten als sehr hoch eingestuft werden oder sie auf bestimmte Hardware angewiesen sind. Ebenso kann es notwendig werden, einzelne Maschinen aus Performanz- oder Sicherheitsgründen von allen anderen virtuellen Maschinen zu isolieren. Die Existenz weiterer sinnvoller Maßnahmen ist nicht ausgeschlossen. Exemplarisch wird im Folgenden zusätzlich zu Separierungs- und Gruppierungsrichtlinien nur noch das Binden einer virtuellen Maschine an ein physisches System betrachtet. Weitere Regeln können jedoch sehr einfach in das im Abschnitt 6.2 spezifizierte Modell integriert werden.

Zusammenfassung der benötigten Operationen für eine optimale Verteilung

Die Tabelle 6.2 fasst die empfohlenen Richtlinien zur Platzierung von virtuellen Maschinen übersichtlich zusammen. Die Aussagen der Tabelle gelten jedoch nur, wenn die betrachteten virtuellen Maschinen überdurchschnittlich hohen Gebrauch der entsprechenden Ressource vorweist. Für den seltenen oder mäßigen Gebrauch von Komponenten in einer virtuellen Maschine haben die Aussagen der Tabelle keine oder nur vernachlässigbare Bedeutung.

	CPU	**RAM**	**Netz**	**Disk**
CPU	seperate	-	seperate	seperate / - [1]
RAM	-	-	-	-
Netz	seperate	-	seperate / group[2]	sepeate / - [1]
Disk	seperate / - [1]	-	seperate / - [1]	seperate / - [1]

Tabelle 6.2: Richtlinien für die Kombination von virtuellen Maschinen abhängig von ihrem Ressourcenbedarf

[1] Bei Verwendung eines HBAs
[2] Bei Kommunikationspartnern

Im Wesentlichen lässt sich demnach eine optimale Verteilung der virtuellen Maschinen in der Infrastruktur durch die folgenden drei Regeln beschreiben.

- Gruppieren von kommunizierenden, virtuellen Maschinen auf einem physischen System zur Erhöhung des Netz-Durchsatzes und zur Verringerung der Latenz
- Separieren von CPU-intensiven, virtuellen Maschinen zur gleichmäßigen Verteilung der Rechenlast auf alle physischen Systeme
- Binden von virtuellen Maschinen an physische Systeme, um Migrationskosten in Form von lange andauernden Ressourcenbelegungen und Beeinträchtigung der Dienste der zu migrierenden virtuellen Maschine zu vermeiden. Dies ist insbesondere bei Maschinen sinnvoll die sehr viel oder sehr stark veränderlichen Hauptspeicher besitzen.

Live-Migration als Mittel zur Optimierung

Nahezu alle getesteten Virtualisierer sind in der Lage durch Live-Migration eine Neuverteilung der virtuellen Maschinen auf den physischen Servern im laufenden Betrieb vorzunehmen. Lediglich Microsofts Hyper-V beherrscht in der vorliegenden Version nur Warm-Migration, die ein temporäres Pausieren der virtuellen Maschine notwendig macht. TCP -Verbindungen dürften das in der Regel nicht überleben. Auch Microsoft arbeitet jedoch an einer Live-Migrations Komponente in Hyper-V, so dass die technischen Voraussetzung für eine Neuverteilung von virtuellen Maschinen bald in allen Produkten vorhanden sein werden.

Zeitpunkte für Optimierung

Strategisch muss man sich jedoch die Frage stellen, wann denn eine Neuverteilung der virtuellen Maschinen sinnvollerweise angestoßen werden sollte und welche Maschinen dabei wohin migriert werden sollen. Eine global gültige Migrationsstrategie zu spezifizieren ist nicht möglich, da je nach Ziel des Virtualisierungsgedankens entweder maximale Konsolidierung oder maximale Performanz zu erreichen sind. Strebt man nach maximaler Konsolidierung, wird man versuchen möglichst viele virtuelle Maschinen auf möglichst wenige Server zu packen. Ein geringfügiges Wachsen der Ressourcenanforderungen einzelner virtueller Maschinen kann in diesem Fall bereits zu hohen Performanzeinbußen führen und sehr viele Migrationen erforderlich machen. Optimiert man rein auf Performanz, so kann man weniger virtuelle Maschinen pro physischem Server betreiben. Als Konsequenz davon wird die Anzahl der Migrationen in diesem Szenario deutlich geringer ausfallen, es werden aber deutlich mehr physische Server benötigt, um dieselbe Anzahl an virtuellen Maschinen zu betreiben. Für die meisten Anwendungsfälle wird man im Normalfall einen Kompromiss aus beiden Ansätzen wählen. Im Bereich des HPC Computings ist es jedoch üblich auf Performanz zu optimieren, Konsolidierung spielt hier nur eine Nebenrolle, die man gerne als kostenlosen Nebeneffekt der Virtualisierung verwendet, welche in diesem Fall aber hauptsächlich zur Hardwareabstraktion eingesetzt wird.

Schwellwerte

Um zwischen den beiden Extremvarianten differenzieren zu können, bietet es sich an, den Administrator einen maximalen Füllgrad spezifizieren zu lassen, bis zu dem virtuelle Maschinen dem Server hinzugefügt werden dürfen. Ein Füllgrad von zum

Beispiel 90%, der natürlich auch für jede Ressource des Servers separat spezifiziert werden könnte, bedeutet, dass nur 90% der Ressourcen des Servers aktiv vergeben werden, und die restlichen 10% als Puffer für kurzzeitig erhöhten Ressourcenbedarf reserviert werden. Bei Überschreiten eines weiteren Schwellwertes, zum Beispiel 95% wird eine Neuverteilung der Maschinen angestoßen. Auch eine heuristische Prognose auf Basis von periodisch wiederkehrenden Lastspitzen oder erkannten steigenden beziehungsweise fallenden Tendenzen aus dem Monitoring in der nahen Vergangenheit könnte das Anstoßen von Migrationen rechtfertigen.

Auswahl der zu migrierenden Maschinen

Neben der Strategie, wann Migrationen angestoßen werden sollten, werden auch Strategien benötigt, die Empfehlungen darüber geben, welche virtuellen Maschinen migriert werden sollen und wohin sie migriert werden sollen. Da bei einer Migration ein gewisser zusätzlicher Aufwand in Form von Netzbandbreite und Rechenaufwand im Hypervisor notwendig ist, sollten jene virtuelle Maschinen ausgewählt werden, deren Migration die geringsten Kosten verursacht. Da eine Migration in der Regel so abläuft, dass der komplette Arbeitsspeicher der aktiven virtuellen Maschine über das Netz auf die Zielmaschine kopiert wird und anschließend, solange Änderungen am Hauptspeicher der virtuellen Maschine auf das Ziel repliziert werden, bis nahezu 100% des Arbeitsspeichers auf Quelle und Ziel identisch sind, sind die Migrationskosten abhängig davon, wie groß die zu übertragende Datenmenge ist. Diese ist abhängig von der Größe des Arbeitsspeichers der virtuellen Maschine und der Änderungsfrequenz dessen Inhalts. Maschinen mit kleinem und konstantem Hauptspeicher sind daher die günstigsten zu migrierenden Maschinen. Die Größe des Festplattenimages spielt in der Regel keine Rolle, da diese meist zentral im Netz abgelegt wurden und sowohl von Quell- als auch Zielserver mountbar sind. Da eine Migration ein minimales Pausieren der virtuellen Maschine im Bereich einiger zehntel Sekunden zur Folge hat, ist es auch eine Überlegung Maschinen, auf denen Realtime-Anwendungen wie etwa Streaming Server laufen, nur zu migrieren, wenn unbedingt notwendig, da dieser Typ von Anwendungen in ihrer Dienstgüte beeinträchtigt werden können.

Auswahl des Ziels einer Migration

Bei der Auswahl des Ziels sind zwei Kriterien zu beachten. Der ausgewählte Server muss zum einen genug freie Ressourcen für die zu migrierende virtuelle Maschine besitzen und zum anderen sollten virtuelle Maschinen nicht wahllos miteinander kombiniert werden, sondern nur solche, deren Ressourenanforderungen sich nicht gegenseitig negativ beeinflussen.

6.2 Automatisierung der Ansätze

Anforderungen für automatische Optimierung

Die angegebenen Richtlinien ermöglichen bereits eine manuelle Optimierung der virtuellen Infrastruktur durch einen Administrator. Eine automatisierte Optimierung auf Basis von Algorithmen ist jedoch wünschenswert, um den Administrator zu entlasten und schnell auf Veränderungen des Ressourcenbedarfs reagieren zu können. Dazu ist es notwendig, die im Abschnitt 6.1.2 angegebenen Richtlinien

in eine Form zu bringen, die von Algorithmen automatisiert verarbeitet werden kann. Desweiteren müssen weitere Anforderungen an den Algorithmus spezifiziert werden, die für einen Administrator intuitiv gegeben und selbstverständlich sind, nicht jedoch für einen Automatismus. Im Anschluss daran wird am Ende dieses Kapitels ein Algorithmus spezifiziert und implementiert.

6.2.1 Generische Anforderungen

Skalierbarkeit Eine wesentliche Anforderung an einen Algorithmus ist eine dem Problem angemessene Laufzeit und ein angemessener Ressourcenverbrauch an sich. Dies inkludiert die Skalierbarkeit des Algorithmus bei wachsender Anzahl an Servern, virtuellen Maschinen sowie Regelsätzen, auch wenn intuitiv ersichtlich ist, dass die Komplexität des Problems in größeren Umgebungen überproportional steigt.

Determinismus Eine errechnete Verteilung sollte bei identischen Eingabeparametern an den Algorithmus erneut als Lösung ausgegeben werden. Dies ist notwendig um unnötige Migrationen zu vermeiden, die andernfalls bei einer Neuberechnung der Verteilung entstehen würden.

Vermeidung von Oszillation / Kosteneffizienz Virtuelle Maschinen sollten bei kleineren Schwankungen im Ressourcenverbrauch nicht zwischen zwei oder mehreren Servern oszillieren, sondern erst migriert werden, falls die effizientere Verteilung die Migrationskosten rechtfertigt.

Erweiterbarkeit Die Erweiterbarkeit des Algorithmus im Sinne der Integration weiterer Eigenschaften muss gegeben sein. So könnten zum Beispiel betriebswirtschaftliche oder rechtliche Aspekte eine aus Sicht der Performanz sinnvolle Verteilung verbieten. Das Regelwerk muss daher flexibel anpassbar und erweiterbar gestaltet werden.

6.2.2 Modellierung der Richtlinien

Binäre Modellierung der Richtlinien

Die bisher identifizierten Richtlinien für die globale Optimierung sind das Separieren, Gruppieren und Binden konkreter virtueller Maschinen. Die Operationen Gruppieren und Separieren benötigen als Argumente jeweils zwei virtuelle Maschinen, sofern man die Operationen binär modelliert, während das Binden einer virtuellen Maschine an einen physischen Server jeweils eine Virtuelle Maschine und ein physisches System als Parameter voraussetzt. Die Operationen für das Gruppieren und Separieren könnten ebenso gut auf einer Menge an virtuellen Maschinen realisiert werden, in der die entsprechende Funktion paarweise auf allen Argumenten zu realisieren ist. Eine Abbildung in die binäre Darstellung ist jedoch immer realisierbar und auch notwendig, um Kosten für jede verletzte

Richtlinie modellieren zu können. Die Modellierung dieser Richtlinien könnte daher wie folgt realisiert werden:

- $pin(VM, PM)$
- $separate(VM_1, VM_2)$
- $group(VM_1, VM_2)$

Modellierungsbeispiele

Ein Beispiel für die Gruppierung eines virtuellen Terminalservers und eines zugehörigen, virtuellen Clients auf einen beliebigen Host zur Steigerung des Netz-Durchsatzes und zur Verringerung der Latenz könnte daher durch

$$group(Terminalserver, Terminalclient)$$

realisiert werden. Eine Separierung zweier CPU-intensiver Instanzen, wie etwa Maschinen zum Ray-Tracing $\mathrm{RTVM}_1, \mathrm{RTVM}_2$ könnte hingegen mit der Schreibweise

$$separate(\mathrm{RTVM}_1, \mathrm{RTVM}_2)$$

modelliert werden. Existieren mehr als zwei Ray-Tracing-VMs, die auf mehrere Server verteilt werden sollen, so könnte dies mit der Syntax

$$separate(\mathrm{RTVM}_1, \mathrm{RTVM}_2, ..., \mathrm{RTVM}_\mathrm{n})$$

spezifiziert werden. Die Bedeutung dieses Ausdrucks ist dann gleichzusetzen mit

$$\forall\, x, y \in \{\mathrm{RTVM}_1, \mathrm{RTVM}_2, ..., \mathrm{RTVM}_\mathrm{n}\} \mid x \neq y : separate(x, y),$$

was der Menge an Separierungs-Anweisung aller binär konstruierbaren Paare entspricht. Das Binden einer virtuellen Maschine mit großem Arbeitsspeicher, der einer hohen Änderungsrate unterworfen ist wie etwa bei Crash-Test-Simulationen, an ein physisches System PM_1 kann mit $pin(\mathrm{CrashTestVM}_1, \mathrm{PM}_1)$ realisiert werden. Hierbei wird impliziert, dass die virtuelle Maschine $\mathrm{CrashTestVM}_1$ bereits auf dem physischen System PM_1 ausgeführt wird. Sollte die Zuordnung im Laufe der Zeit aus irgendeinem Grund geändert werden, so muss auch die Regel an den neuen Host angepasst werden.

Instanziierung der Richtlinien

Mit der angegebenen Syntax lassen sich Richtlinien für die Instanziierung einer Verteilung der virtuellen Maschinen auf physische Server in einer virtuellen Infrastruktur realisieren. Die Beschreibung einer gewünschten Verteilung muss durch einen Administrator manuell erstellt oder durch einen Algorithmus auf Basis von höherwertigen Richtlinien automatisiert erzeugt werden. Ein Beispiel für den zweiten Ansatz wäre etwa das automatische Kategorisieren von virtuellen Maschinen auf die von ihnen am meisten benutzte Ressource aufgrund von Monitoringdaten und das anschließende Erstellen von Gruppierungs- und Separierungsrichtlinienn mit dem Ziel, die in 6.1.2.2 beschriebenen Querabhängigkeiten der Ressourcen zu vermeiden.

Eine Beschreibung der Verteilung setzt sich in der Regel aus einer Menge an Separierungs- , Gruppierungs- und Bindeoperationen zusammen, so dass es Ziel

eines zu realisierenden Algorithmus ist, aufgrund der vorliegenden Beschreibung eine Verteilungsinstanz zu finden, die alle angegebenen Richtlinien erfüllt.

6.2.3 Algorithmen

Lösung ist NP-vollständig

Die Lösung des gegeben Optimierungs-Problems ist alles andere als trivial, da bereits die optimale Verteilung der virtuellen Maschinen auf physische Server ähnlich zum Knappsack-Problem ist und damit ein NP-vollständiges Problem darstellt. Eine optimale Lösung dieses Problems ist daher nur durch vollständiges Ausprobieren aller Permutationen der Verteilung möglich. Da die Anzahl der virtuellen Maschinen und Server in großen Rechenzentren und Serverfarmen sehr groß werden kann und die Anzahl der Permutationen exponentiell mit der Zahl der virtuellen Maschinen steigt, müssen Algorithmen gefunden werden, die keine optimale Lösung des Problems forcieren, sondern lediglich eine gute, dafür jedoch eine drastisch bessere Laufzeit erreichen. Eine Gruppe von Algorithmen, die diesen Ansatz verfolgen sind Constraint Solver, die mit gewichteten Constraints arbeiten. Für jedes Constraint können virtuelle Kosten spezifiziert werden, die, wenn sie nicht erfüllt werden, fällig werden. Eine Lösung des Problems ist genau dann eine valide Gesamtlösung, wenn die Summe der Strafwerte aller verletzten Constraints einen zuvor definierten Schwellwert nicht übersteigt. Dieser Ansatz ermöglicht es, abhängig vom gesetzten Schwellwert, einzelne Constraints zu verletzen, jedoch sinkt mit diesem Verfahren die Wahrscheinlichkeit eine unlösbare Menge an Constraints spezifiziert zu haben. Ein vergleichbares Constraint Satisfaction Problem (CSP) mit harten Constraints, welche in einer Lösung zwangsweise erfüllt sein müssen, weist in diesem Punkt eine weitaus höhere Anfälligkeit auf. Weiterhin erlaubt der gewichtete Ansatz (WCSP) weitere Eingriffsmöglichkeiten, um die Anforderungen Stabilität und Vermeidung von Oszillation zu realisieren. Leider können existierende Algorithmen, welche zur Lösung gewichteter CSPs implementiert sind, zum heutigen Stand der Forschung nur unäre und binäre Constraints in Form konkreter Werte effizient verarbeiten [LaSc 04], so dass höherwertige Constraints, wie etwa das Separieren einzelner virtueller Maschinen auf mehrere Hosts zuvor mit dem damit verbundenen Rechenaufwand in entsprechende Form gebracht werden müssen. Im Folgenden werden mehrere konkrete Algorithmen vorgestellt und ihre Vor- und Nachteile diskutiert, um eine Basis für einen Algorithmus zur Verteilung von virtuellen Maschinen mit dem spezifiziertem Regelwerk zu finden. Konkret wird dabei anlehnend an [LaSc 04] die folgende Notationsweise herangezogen:

Constraint Solver

$\mathcal{X}$: eine Menge an Variablen

$\mathcal{D}$: eine Menge an Wertebereichen; konkret ist D_i der Wertebereich für die Variable $i \in \mathcal{X}$

E : eine Menge an Kosten

$\top$: maximale Kosten; entspricht “verboten“

$\bot$: minimale Kosten; entspricht “kostenlos“

$\oplus$: Kostenaggregatsfunktion, $a \oplus b := \min\{a + b, \top\}$

$\mathcal{C}$: eine Menge unärer und binärer Kostenfunktionen;
konkret ist $C_i : D_i \rightarrow E$ die unäre Kostenfunktion der Variablen i
und $C_{ij} : D_i \times D_j \rightarrow E$ eine binäre Kostenfunktion für die Variablen i und j

(i, a) : die Variable i ist mit dem Wert a belegt

Intuitiver Lösungsansatz

Die einfachste Herangehensweise zur Lösung eines WCSP ist, über alle möglichen Variablenbelegungen zu iterieren und so alle möglichen Permutationen auf mögliche Lösungskandidaten zu überprüfen. Formal gesehen durchläuft man bei diesem Verfahren einen Baum, dessen Knoten Variablen entsprechen und dessen Kanten Variablenbelegungen repräsentieren. Ein Beispiel eines solchen Baumes wird in Abbildung 6.1 dargestellt. Ein Algorithmus, der dies realisiert, ist der *Depth-first Branch and Bound-* Algorithmus, dessen Laufzeit allerdings exponentiell mit der durchschnittlichen Größe der Wertemengen der Variablen steigt. Er trägt jedoch zum Verständnis der Problematik bei und ist Grundlage für eine optimierte Variante, die am Ende dieses Teilkapitels entwickelt wird.

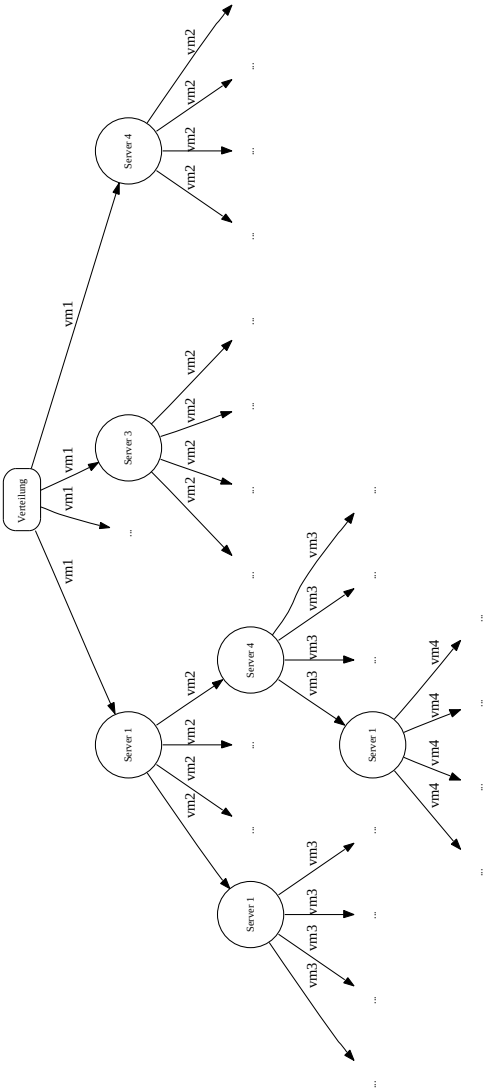


Abbildung 6.1: Durchlaufener Baum beim Algorithmus Depth-first Branch and Bound

6.2.3.1 Depth-first Branch and Bound

Pfade entsprechen Belegungen

Der Depth-first Branch and Bound Algorithmus realisiert eine Tiefensuche in einem Baum, bei der im schlimmsten Fall alle Äste durchlaufen werden müssen. Jeder Knoten entspricht dabei einer Variablen, jeder Ast einer konkreten Variablenbelegung. Jeder Pfad von der Wurzel zum äußersten Variablenknoten enthält jede Variable des CSP exakt einmal. Durchläuft man einen Pfad von der Wurzel bis zum äußersten Ast, so erhält man auf diesem Weg eine eindeutige und vollständige Variablenbelegung des CSPs. Jede einzelne Variablenbelegung kann mit Kosten verbunden sein, wenn die vorgenommene Belegung nicht erwünscht ist. Der Algorithmus terminiert die Suche auf einem Pfad, wenn die summierten Kosten über eine konkrete Variablenbelegung auf dem Suchpfad die obere Schranke der spezifizierten Kosten überschreiten oder bereits auf einem zuvor durchlaufenen Pfad eine Belegung mit geringeren Kosten gefunden wurde. Der Ablauf des Algorithmus ist wie folgt: Zunächst wird eine beliebige Variable i ausgewählt, die nacheinander mit allen möglichen Werten ihres Wertebereichs belegt wird. Für jede Belegung werden die Kosten der unären Kostenfunktion für die konkrete Belegung zu den mitgeführten Gesamtkosten addiert und die Belegung gespeichert. Anschließend werden in der Funktion `LookAhead` die Kosten aller binärer Constraints C_{ij} auf die Kosten der unären Constraints C_j übertragen und eliminiert. Die Funktion `LocalConsist` überprüft, ob das entstandene Subproblem noch lösbar ist, und entfernt gegebenenfalls Werte mit zu hohen Kosten aus dem Wertebereich, ehe eine rekursiver Aufruf für das Subproblem erfolgt.

Abbruch der Suche bei zu teuren Pfaden

Der Algorithmus ist in der Lage potentielle Lösungen des CSPs zu verwerfen, auch wenn ein Lösungspfad noch nicht in seiner vollen Tiefe durchlaufen wurde, da bereits ganze Teilbäume ausgeschlossen werden können, wenn an einem inneren Knoten eine Variablenzuweisung die Kosten für das Gesamtproblem die obere Schranke durchbrechen oder bereits eine Lösung mit niedrigeren Kosten gefunden wurde. Es ist jedoch erst nach Durchlaufen aller Pfade bekannt, ob eine gefundene Lösung bereits die bestmögliche darstellt oder ob bessere Lösungen existieren. Ebenso ist im schlimmsten Fall erst nach einem vollständigen Durchlauf des Algorithmus eine Aussage möglich, ob ein CSP überhaupt eine Lösung besitzt. Selbst wenn im Schnitt eine Lösung schon nach der Hälfte aller möglichen Belegungen gefunden wurde, so muss der restliche Bereich dennoch durchlaufen werden, um die Optimalität dieser Belegung zu beweisen. Die Laufzeit des Algorithmus ist daher von der Komplexität

$$\Theta(\prod_{i \in \mathcal{X}} \|D_i\|)$$

und somit exponentiell. Dieses Verhalten kann auch dadurch nicht verbessert werden, dass man von einer optimalen Lösung absieht, sondern die erste Lösung greift, deren Kosten eine untere Schranke unterbietet. Dies kann die Laufzeit zwar um einen gewissen Faktor verbessern, ändert jedoch nichts an der exponentiellen

Laufzeit des Algorithmus und verschlechtert zumindest im Schnitt das Ergebnis.

Optimierungsansätze

Das Verhalten dieses Algorithmus kann in geeigneten Situationen durch die Integration weiterer Algorithmen, wie zum Beispiel W-AC3 oder W-AC*3 verbessert werden. Daher werden diese beiden Algorithmen im Folgenden kurz erläutert, ehe auf Basis des Depth-First Branch and Bound Algorithmus ein Lösungsansatz für das konkrete Verteilungsproblem entwickelt wird.

Algorithm 1: Depth-first Branch and Bound [LaSc 04]

Function `BranchAndBound`$(t, v_t, k, \mathcal{X}, \mathcal{D}, \mathcal{C})$
1. **if** $(X = \varnothing)$ **then**
2. $\quad$ **return** lb;
3. **else**
4. $\quad i :=$ `ChooseVar`$(\mathcal{X})$;
5. $\quad$ **foreach** $a \in D_i$ **do**
6. $\quad\quad \mathcal{DD} := \mathcal{D}; \mathcal{CC} := \mathcal{C}; Nt := t + (i, a); v_{Nt} := v_t \oplus C_i(a);$
7. $\quad\quad$ `LookAhead`$(i, a, \mathcal{CC})$;
8. $\quad\quad$ **if** (`LocalConsist`$(k, \mathcal{X} - \{i\}, \mathcal{DD}, \mathcal{CC})$) **then**
9. $\quad\quad\quad k :=$ `BranchAndBound`$(Nt, v_{Nt}, k, \mathcal{X} - \{i\}, \mathcal{DD}, \mathcal{CC})$;
10. **return** k;

Procedure `LookAhead`$(i, a, \mathcal{CC})$
11. $\mathcal{CC} := \mathcal{CC} - \{C_i\}$;
12. **foreach** $C_{ij} \in \mathcal{CC}$ **do**
13. $\quad$ **foreach** $b \in D_j$ **do**
14. $\quad\quad C_j(b) := C_j(b) \oplus C_{ij}(a, b)$;
15. $\quad \mathcal{CC} := \mathcal{CC} - \{C_{ij}\}$;

6.2.3.2 W-AC3

Übertragen binärer auf unäre Kosten

Der W-AC3 Algorithmus basiert auf einem anderem Ansatz als der Depth-first Branch and Bound Algorithmus. Er versucht binäre Constraints aufzulösen und deren Kosten auf unäre Constraints abzuwälzen. Dies wird realisiert, indem für jeden Wert a einer Variable die minimalen Kosten α zu allen Werten b einer anderen Variable gesucht werden und diese zu den Kosten des unären Constraints des Wertes a addiert werden. Im Gegenzug werden die minimalen Kosten α von allen Kosten der überprüften binären Constraints subtrahiert, so dass das Constraint mit dem ermittelten minimalen Kosten die Kosten $\perp$ erhält und damit gelöscht werden kann. Sollten durch diesen Schritt die Kosten eines Wertes a größer als die maximal zulässigen Kosten werden, so kann dieser Wert aus dem Wertebereich der zugehörigen Variablen entfernt werden. Wird für eine Variable

der Wertebereich verändert, so muss sie das gesamte Verfahren wiederholt durchlaufen. Das Verfahren wird initialisiert mit der kompletten Variablenmenge des WCSP. Die Laufzeit des Algorithmus beträgt $O(ed^3)$ und ist daher polynomiell. Im Gegensatz zum Depth-first Branch and Bound Algorithmus ist der W-AC3 Algorithmus ein Verfahren mit einer um eine Klasse besseren Laufzeit, berechnet jedoch keine konkrete Belegung der Variablen, sondern vereinfacht lediglich die Problemstellung, indem er offensichtlich unzulässige Belegungen aus der Datenstruktur entfernt. Das Ergebnis des W-AC3 Algorithmus ist daher ebenfalls ein WCSP, das äquivalent zum ursprünglichen Problem ist, jedoch in seiner Struktur vereinfacht ist. Je dichter das Problem mit binären Constraints belegt ist, desto rentabler wird der Einsatz des W-AC3 Algorithmus, denn in diesem Fall können viele binäre Constraints aufgelöst und Wertemengen beschnitten werden. Ein anschließender Lauf des Depth First Branch and Bound Algorithmus kann folglich schneller und effektiver ausgeführt werden, denn die Anzahl der enthaltenen Elemente aller Wertemengen beeinflussen die Laufzeit des Algorithmus jeweils linear. Bereits wenige beschnittene Wertemengen rechtfertigen daher den zusätzlichen Lauf des W-AC3 Algorithmus und könnten die Zeit, die für das Finden einer optimalen Lösung des Problems erforderlich ist, reduzieren. Der Algorithmus basiert auf Definitionen von Knoten- und Kantenkonsistenz, die, sofern ein Problem in eine Knoten und Kantenkonsistente Form überführt wurde, ein effizienteres Lösen des WCSP ermöglicht. Als knotenkonsistent wird ein Wert im Falle von W-AC3 bezeichnet, wenn sein zugehöriger Kostenwert kleiner als die obere Schranke des Kostenwertes für das gesamte WCSP ist. Analog ist eine Variable knotenkonsistent, wenn all ihre möglichen Werte knotenkonsistent sind, und ein WCSP ist knotenkonsistent, wenn all seine Variablen knotenkonsistent sind. Eine Belegung der Variable i mit dem Wert a heißt kantenkonsistent in Bezug auf das binäre Constraint C_{ij}, wenn die Belegung knotenkonsistent ist und ein Wert $b \in D_j$ existiert, so dass $C_{ij}(a, b) = \bot$. Eine Variable ist kantenkonsistent, wenn all ihre Werte bezogen auf ihre assoziierten Constraints kantenkonsistent sind. Ein WCSP ist kantenkonsistent, wenn all seine Variablen kantenkonsistent sind. Die Kantenkonsistenz im W-AC3 Algorithmus wird sichergestellt, indem nach und nach möglichst viele binäre Constraints aufgelöst und deren Kosten auf unäre Constraints übertragen werden. Verletzen dabei einzelne Werte das Kriterium der Knotenkonsistenz, so werden die Werte aus dem Wertebereich der zugehörigen Variable entfernt. Es kann jedoch nicht ausgeschlossen werden, dass die Summe aller minimalen Strafwerte der belegten Variablen kleiner als die obere Schranke ist. Dieses Problem lässt sich zum Teil durch eine stärkere Definition der Knotenkonsistenz erreichen, welche im W-AC*3 Algorithmus zum Einsatz kommt.

Ploynomielle Laufzeit

Knoten- vs. Kantenkonsistenz

6.2.3.3 W-AC*3

Der W-AC*3 Algorithmus arbeitet nahezu identisch zum W-AC3 Algorithmus mit dem Unterschied, dass eine strengere Definition der Knotenkonsistenz implementiert wird. Hierzu wird eine globale Variable namens $C_\emptyset$ eingeführt, die die Kosten

Erweiterung des W-AC*3 Algorithmus

Algorithm 2: W-AC3 [LaSc 04]

Procedure `FindSupportsAC3(`i, j`)`
1. **foreach** $a \in D_i$ **do**
2. $\quad \alpha := min_{b \in D_j}\{C_{ij}(a,b)\}$;
3. $\quad C_i(a) := C_i(a) \oplus \alpha$;
4. $\quad$ **foreach** $b \in D_j$ **do**
5. $\quad\quad C_{ij}(a,b) := C_{ij}(a,b) \ominus \alpha$;

Function `PruneVar(`i`)`
6. $change := false$;
7. **foreach** $a \in D_i s.t. C_i(a) = \top$ **do**
8. $\quad D_i := D_i - \{a\}$;
9. $\quad change := true$;
10. **return** $change$;

Procedure `W-AC3(`$\mathcal{X}, \mathcal{D}, \mathcal{C}$`)`
11. $Q := \{1, 2, ..., n\}$;
12. **while** $Q \neq \varnothing$ **do**
13. $\quad j :=$ `pop(`Q`)`;
14. $\quad$ **foreach** $C_{ij} \in \mathcal{C}$ **do**
15. $\quad\quad$ `FindSupportsAC3(`i, j`)`;
16. $\quad\quad$ **if** `PruneVar(`i`)` **then**
17. $\quad\quad\quad Q := Q \cup \{\text{i}\}$;

der besten noch erreichbaren Lösung enthält. Ihr Wert wird berechnet, indem für jede Variable der minimale Kostenwert ihrer unären Constraints bestimmt, auf den bestehenden Wert von $C_\emptyset$ addiert und von den einzelnen Kostenwerten der zur Berechnung herangezogenen unären Constraints subtrahiert wird. Eine Variable i heißt knotenkonsistent, wenn für all ihre Werte a

$$C_i(a) \oplus C_\emptyset < \top$$

gilt und mindestens ein Wert a existiert, so dass $C_i(a) = \bot$. Die Komplexität des Algorithmus steigt durch die Anpassung der Definition von Knotenkonsistenz auf

$$O(e * d^3 + n^2 d^2).$$

Die Variable $C_\emptyset$ enthält nach einem Durchlauf die untere Schranke der Kosten, die für die Lösung des WCSP anfallen werden. Mit diesem Wert lassen sich analog zum W-AC3 Algorithmus weitere Äste vom Depth-First Branch and Bound Algorithmus ausschließen und damit komplexe Probleme effizienter lösen. Im Gegensatz zu W-AC3 bringt der W-AC*3 Algorithmus eine höhere Laufzeitkomplexität mit sich, die sich, falls das Problem dicht genug mit Constraints besetzt ist, jedoch durch weiter beschnittene Domains amortisieren kann.

6.2.3.4 Integration weiterer Anforderungen in den Algorithmus

Server haben beschränkte Kapazitäten

Technisch gesehen ist bereits der einfache Depth-first Branch and Bound Algorithmus in der Lage WCSP-Probleme zu lösen. Ein Problem in der Praxis ist jedoch seine exponentielle Laufzeit, die bereits bei mittelgroßen Szenarien, wie z.B. fünf Servern und 100 virtuellen Maschinen, sehr schnell in den Stunden- und Tage-Bereich wächst. Zusätzlich berücksichtigt der Algorithmus in seiner nativen Form nicht, dass Server, die hier formal als Elemente dargestellt werden, eine Kapazität besitzen und dass virtuelle Maschinen - formal dargestellt als Variablen - Ressourcen beanspruchen. Berücksichtigt man letzteres Kriterium, welches formal gesehen eine weitere Art Constraint darstellt, die sich jedoch nur sehr schwer als Kombination von unären und binären Constraints darstellen lässt, so lässt sich der Lösungsraum des Problems weiter einschränken. Damit können weitere Teilbäume im Algorithmus zum Teil sehr früh abgeschnitten werden und die Anzahl der vollständig zu überprüfenden Kombinationen des Lösungsraumes sinkt stark, was eine reduzierte Laufzeit zur Folge hat. Die Funktion `BranchAndBound` im Algorithmus 4 implementiert diese Erweiterung, indem für jeden Wert einer Domain geprüft wird, ob für die durchzuführende Variablenbelegung noch Ressourcen vorhanden sind oder nicht. Im positiven Fall werden die Ressourcen allokiert und die Berechnung der Lösung rekursiv fortgesetzt. Kehrt die Rekursion in die Aufrufende `BranchAndBound`-Instanz zurück, so werden die reservierten Ressourcen wieder freigegeben. Für diese Erweiterung ist es notwendig, ein neues Symbol für Ressourcen einzuführen. Im Algorithmus werden von einem Server bereitgestellte Ressourcen daher mit R_a bezeichnet, während der Ressourcenbedarf virtueller Maschinen mit R_i bezeichnet wird. Da es sich bei Ressourcen um

Algorithm 3: W-AC*3 [LaSc 04]

Procedure `FindSupportsAC*3(`i, j`)`
1. **foreach** $a \in D_i$ **do**
2. $\quad \alpha := min_{b \in D_j}\{C_{ij}(a,b)\}$;
3. $\quad v := Nil$;
4. $\quad C_i(a) := C_i(a) \oplus \alpha$;
5. $\quad$ **foreach** $b \in D_j$ **do**
6. $\quad\quad C_{ij}(a,b) := C_{ij}(a,b) \ominus \alpha$;
7. $\alpha := min_{a \in D_i}\{C_i(a)\}$;
8. $C_\emptyset := C_\emptyset \oplus \alpha$;
9. **foreach** $a \in D_i$ **do**
10. $\quad C_i(a) := C_i(a) \ominus \alpha$;
11. **return** $\alpha \neq \bot$;

Function `PruneVar(`i`)` **:** `Boolean`
12. $change := false$;
13. **foreach** $a \in D_i$ *s.t.* $C_i(a) \oplus C_\emptyset = \top$ **do**
14. $\quad D_i := D_i - \{a\}$;
15. $\quad change := true$;
16. **return** $change$;

Procedure `W-AC*3(`$\mathcal{X}, \mathcal{D}, \mathcal{C}$`)`
17. $Q := \{1, 2, ..., n\}$;
18. **while** $Q \neq \emptyset$ **do**
19. $\quad j :=$ `pop(`Q`)`;
20. $\quad flag := false$;
21. $\quad$ **foreach** $C_{ij} \in \mathcal{C}$ **do**
22. $\quad\quad flag := flag \vee$ `FindSupportsAC*3(`i, j`)`;
23. $\quad\quad$ **if** `PruneVar(`i`)` **then**
24. $\quad\quad\quad Q := Q \cup \{i\}$;
25. $\quad$ **if** $flag$ **then**
26. $\quad\quad$ **foreach** $i \in \mathcal{X}$ *s.t.* `PruneVar(`i`)` **do**
27. $\quad\quad\quad Q := Q \cup \{i\}$;

mehrdimensionale Objekte handelt, werden sie als Vektor definiert. Die einzelnen Dimensionen dieser Vektoren stehen für eine definierte Ressource eines Servers, wie zum Beispiel CPU, RAM, Netz oder Disk. Additionen und Subtraktionen von Vektoren können auf diese Weise mit den bekannten Rechenregeln durchgeführt werden.

Beispiel für die Darstellung von Ressourcen

Ein physisches System mit zwei Prozessorsockeln und je vier Kernen mit 3.0 GHz Taktfrequenz, 32 GByte Hauptspeicher, 1 GBit/s Ethernet NIC und 250 MBit/s maximalem Disk-I/O kann beispielsweise wie folgt als Vektor modelliert werden:

$$\mathrm{Server}_1 = \begin{pmatrix} 24\ \mathrm{GHz} \\ 32\ \mathrm{GB} \\ 1\ \mathrm{GBit/s} \\ 250\ \mathrm{Mbit/s} \end{pmatrix}$$

Eine virtuelle Maschine, die beispielsweise einen Webserver beinhaltet, kann anaolog wie folgt modelliert werden:

$$\text{Webserver-VM} = \begin{pmatrix} 1{,}5\ \mathrm{GHz} \\ 2\ \mathrm{GB} \\ 100\ \mathrm{MBit/s} \\ 1\ \mathrm{Mbit} \end{pmatrix}$$

Die nach Zuweisung der Webserver-VM auf Server_1 verbleibende Restkapazität beträgt die Differnz der Vektoren. In diesem Fall:

$$\mathrm{Rest} : \mathrm{Server}_1 = \begin{pmatrix} 22{,}5\ \mathrm{GHz} \\ 30\ \mathrm{GB} \\ 900\ \mathrm{MBit/s} \\ 249\ \mathrm{Mbit} \end{pmatrix}$$

Ist jeder Eintrag der Restkapazität größer als Null, so ist die Zuweisung der virtuellen Maschine an den entsprechenden Server grundsätzlich möglich. Da die Ressourcen eines Servers jedoch nicht zu 100% an virtuelle Maschinen vergeben werden sollten, sondern ein Teil der Rechenleistung und des Hauptspeichers für den Hypervisor reserviert werden sollte, ist es in der Regel sinnvoll von der vorhandenen Kapazität des Servers ein bis zwei Kerne, in diesem Fall also 6 GHz sowie je nach Hypervisor etwas Hauptspeicher vom initialen Vektor abzuziehen. Eine Zuweisung einer virtuellen Maschine an ein physisches System ist genau dann nicht mehr möglich, wenn bei der Differenzbildung mindestens eine Komponente negativ wird. Die in diesem Beispiel angegebene Vektoren sind einfach gehalten und können bei Bedarf beliebig um weitere Ressourcen oder Eigenschaften erweitert werden. Beispielsweise wäre es möglich zwischen lesendem und schreibendem

Netz- / Disk-IO zu unterscheiden und die Modellierung auf diese Weise feingranularer zu gestalten.

Abbruch bei Überschreiten einer besten bisher gefunden Lösung

Die Funktion `LocalConsist` wurde erweitert um den Aufruf des W-AC3 beziehungsweise W-AC*3 Algorithmus, wann immer ein neues Subproblem berechnet wird. Entsteht durch den Aufruf eine leere Wertemenge oder sind die Kosten des bisher berechneten Teilpfades bereits größer als die beste bisher gefundene Lösung, so würde die Berechnung aller Subpfade zu Lösungen mit höheren Kosten führen und kann daher an dieser Stelle abgebrochen werden.

Local Repair

Sortiert man weiterhin alle Domains aufsteigend nach den Kosten ihrer assoziierten unären Constraints, so wird der Lösungsbaum entlang des Astes mit den niedrigsten unären Kosten zuerst durchlaufen. Unter der Annahme, dass es sich nicht um eine initiale Optimierung der Verteilung handelt, sondern um eine Optimierung der Verteilung im laufenden Betrieb, kann man davon ausgehen, dass weitestgehend alle binären Constraints realisiert sind. Belegt man zusätzlich alle Zuordnungen von virtuellen Maschinen auf Hosts, die Migrationen erfordern, also kostenintensiv sind, automatisiert mit Kosten, so entspricht der Pfad mit den minimalen unären Kosten im Idealfall der aktuellen Belegung oder ist dieser zumindest sehr ähnlich. In beiden Fällen wird eine Lösung durch den Algorithmus sehr schnell gefunden. Diese Lösung muss nicht zwangsläufig die beste existierende Lösung sein, auf jeden Fall jedoch eine sehr kostengünstige Lösung im Bezug auf Migrationskosten.

Begrenzung der Laufzeit

Ein strategischer Ansatz zur Optimierung der Infrastruktur wäre es daher, den Algorithmus tagsüber beziehungsweise zu Zeiten höherer Last nur so lange laufen zu lassen, bis ein globales Optimum gefunden wurde und zu Zeiten niedrigerer Last - zum Beispiel nachts - längere Laufzeiten zu erlauben. Da die Lösung des Verteilungsproblems nach längerer Laufzeit, sofern eine bessere Lösung gefunden wurde, mit Migrationskosten belegt sein wird, kann diese ebenfalls nachts realisiert werden. Ein störender Einfluss des Optimierungsverfahrens auf den Produktivbetrieb ist somit ausgeschlossen. Durch die Spezifikation von Kosten für Migrationsvorgänge und eines oder mehrerer Kosten-Nutzen-Quotienten für die Migrationen lassen sich zum Beispiel abhängig von der Tageszeit individuell Schwellwerte festlegen, bis zu denen eine oder mehrere Migrationen erfolgen oder unterbunden werden sollen. Ist die Kostendifferenz der bestehenden und der berechneten Verteilung positiv, so ist die neu gefundene Belegung besser als die bestehende. Ist der Quotient aus Migrationskosten und dieser Differenz kleiner als ein vorgegebener Schwellwert, so ist eine Anpassung der Verteilung auch wirtschaftlich.

Heuristiken

Ein Aspekt, der bei der Berechnung von Verteilungen zu Zeiten niedriger Last beachtet werden muss, ist, dass der Algorithmus nicht mit den aktuellen Werten aus dem Monitoring gestartet werden darf, sondern, dass Heuristiken notwendig sind, die das Verhalten zu Hochzeiten approximieren. Diese Heuristiken können sowohl musterbasiert als auch tendenzbasiert sein. Im ersten Fall würde eine zukünftige Auslastung auf Basis von periodisch wiederkehrenden Lastmustern berechnet, im zweiten Fall eine Last auf Basis von approximierten Werten des Monitorings über

einen definierten Zeitraum in der nahen Vergangenheit. Einen möglichen zeitlichen Ablauf eines gesamten Optimierungsdurchlaufes stellt Abbildung 6.2 dar.

6.2.3.5 Datenstrukturen

Reduzieren komplexer Constraints auf maximal binäre Constraints

Die vorgestellten und weiterentwickelten Algorithmen können maximal Constraints binären Grades verarbeiten, höherwertige Constraints können von ihnen zumindest nicht nativ verarbeitet werden. Die zur globalen Optimierung notwendigen Constraint Typen

- Binden einer VM an einen dedizierten Server (*pin*)
- Gruppieren zweier VMs auf einem beliebigen Server (*group*)
- Separieren zweier VMs auf unterschiedliche Server (*separate*)

müssen daher auf Constraints mit maximal binärer Komplexität abgebildet werden. Die hierfür notwendigen Relationen für die Constraints vom Typ *pin*, *group* und *separate* sind intuitiv:

- Für eine bevorzugte Platzierung der virtuellen Maschine i auf dem Server a, also $pin(i, a, costs)$ sollen für die Belegung $i = a$ keine Kosten anfallen, für alle anderen Belegungen von i mit $a \in D_i \setminus \{a\}$ fallen Kosten in der Höhe von *costs* an. Dies lässt sich mit unären Constraints für die Variable i modellieren.
- Für ein bevorzugtes Separieren der virtuellen Maschinen i und j mittels $separate(i, j, costs)$, sollen für jede Belegung, für die $i \neq j$ gilt, keine Kosten anfallen, anderenfalls die Kosten *costs*. Dies lässt sich mittels binärer Constraints für die Variablen i und j modellieren.
- Für ein bevorzugtes Gruppieren der virtuellen Maschinen i und j mittels $group(i, j, costs)$ sollen für jede Belegung, für die $i = j$ gilt, keine Kosten anfallen, anderenfalls die Kosten *costs*. Dies lässt sich mittels binärer Constraints für die Variablen i und j modellieren.

Übersetzung vor der Laufzeit

Die Übersetzung der höherwertigen Constraints in unäre und binäre Constraints mit dem angegebenen Algorithmus kann initial vor dem Aufruf des Branch and Bound Algorithmus erfolgen. Sie besitzt die Komplexität

$O(\sum_{C_i \in pin(i,a,costs)} |D_i|)$ für Pin Anweisungen und

$O(\sum_{C_{ij} \in separate(i,j,costs)} \cdot |D_i| \cdot |D_j|)$ für Separate Anweisungen bzw.

$O(\sum_{C_{ij} \in group(i,j,costs)} \cdot |D_i| \cdot |D_j|)$ für Group Anweisungen.

Erweiterung des Algorithmus vermeidet Übersetzung

Auch wenn sich die Übersetzungskosten in Grenzen halten, so ist es doch von Nachteil, dass Änderungen an den höherwertigen Constraints eine komplette Neuübersetzung zur Folge haben, da die ursprünglich erzeugten unären und binären

Algorithm 4: Depth-first Branch and Bound Extended

Function `BranchAndBound`$(t, v_t, k, \mathcal{X}, \mathcal{D}, \mathcal{C})$

1. **if** $(X = \varnothing)$ **then**
2. $\quad$ **return** lb;
3. **else**
4. $\quad$ $i :=$ `ChooseVar`$(\mathcal{X})$;
5. $\quad$ **foreach** $a \in D_i$ **do**
6. $\quad\quad$ **if** $R_i < R_a$ **then**
7. $\quad\quad\quad$ $R_a = R_a - R_i$;
8. $\quad\quad$ **else**
9. $\quad\quad\quad$ **continue**;
10. $\quad\quad$ $\mathcal{DD} := \mathcal{D}$; $\mathcal{CC} := \mathcal{C}$; $Nt := t + (i, a)$; $v_{Nt} := v_t \oplus C_i(a)$;
11. $\quad\quad$ `LookAhead`$(i, a, \mathcal{CC})$;
12. $\quad\quad$ **if** $($`LocalConsist`$(k, \mathcal{X} - \{i\}, \mathcal{DD}, \mathcal{CC}))$ **then**
13. $\quad\quad\quad$ $k :=$ `BranchAndBound`$(Nt, v_{Nt}, k, \mathcal{X} - \{i\}, \mathcal{DD}, \mathcal{CC})$;
14. $\quad\quad$ $R_a = R_a + R_i$;
15. **return** k;

Procedure `LocalConsist`$(k, \mathcal{X}, \mathcal{DD}, \mathcal{CC})$

16. **if** $(k >= lb)$ **then**
17. $\quad$ **return False**;
18. **else**
19. $\quad$ `W-AC*3`$(\mathcal{X}, \mathcal{DD}, \mathcal{CC})$;
20. $\quad$ **if** $\exists D_i \in \mathcal{DD} s.t. D_i = \varnothing$ **then**
21. $\quad\quad$ **return False**;
22. $\quad$ **return True**;

Procedure `LookAhead`$(i, a, \mathcal{CC})$

23. $\mathcal{CC} := \mathcal{CC} - \{C_i\}$;
24. **foreach** $C_{ij} \in \mathcal{CC}$ **do**
25. $\quad$ **foreach** $b \in D_j$ **do**
26. $\quad\quad$ $C_j(b) := C_j(b) \oplus C_{ij}(a, b)$;
27. $\quad$ $\mathcal{CC} := \mathcal{CC} - \{C_{ij}\}$;

Constraints durch die Algorithmen W-AC3 / W-AC*3 inzwischen nicht nachvollziehbar geändert worden sein können. Eine elegantere Herangehensweise ist es daher, die Algorithmen so zu modifizieren, dass sie die benötigten Constraints nativ verarbeiten können. Projektionen von binären auf unäre Kosten können bei diesem Ansatz jedoch nicht mehr in der ursprünglichen Form auf der originalen Datenstruktur durchgeführt werden, sondern Änderungen müssen in einer separaten Datenstruktur mitgeführt werden. Schreibt man $C^0_{ij}(a,b)$ für die initialen binären Kosten und puffert Dekrementierungen an diesem Wert in einer zusätzlichen Datenstruktur $F(i,j,a)$, so kann man alle lesenden Zugriffe auf $C_{ij}(a,b)$ innerhalb der Algorithmen ersetzen durch

$$C^0_{ij}(a,b) \ominus F(i,j,a) \ominus F(j,i,b).$$

Schreibzugriffe auf $C_ij(a,b)$ ersetzt man stattdessen durch Schreibzugriffe auf die neue Datenstruktur $F(i,j,a)$. Die initialen binären Constraints $C^0_{ij}(a,b)$ werden nun nur mehr gelesen und können folglich auch in funktionaler statt lediglich in relationaler Art und Weise eingelesen werden. Im Falle eines Gruppierungs-Constraints $group(i,j)$ entsprächen beispielsweise die Kosten von $C^0_{ij}(a,b)$ einem Aufruf der Funktion $group_{i,j}(a,b,costs)$. Analog kann mit unären Constraints verfahren werden. Der einzige Unterschied besteht darin, dass alle $F(i,j,a)$ mit $\perp$ initialisiert werden, während alle $F_i(a)$ mit $C^0_i(a)$ initialisiert werden. Exemplarisch wurde die Ersetzung am Algorithmus 4 durchgeführt. Das Ergebnis ist der nachstehende Algorithmus 5.

Modellierung des WCSP in XML

Der Algorithmus benötigt als Eingabe eine komplette Beschreibung des WCSP Problems. Hierfür eignet sich die in [RoLe 09] spezifizierte XML-Syntax XCSP, welche CSP, QCSP und WCSP Instanzen repräsentieren kann. Das XML-Dokument zur Repräsentation besteht aus mehreren Abschnitten, in denen nacheinander Wertemengen (Domains), Variablen (Variables), Relationen (Relations), Funktionen (Functions) und Constraints (Constraints) definiert werden. Während die Bedeutung von Variablen und Wertemengen intuitiv klar ersichtlich ist, ist die Art und Weise der Spezifikation von Constraints sehr flexibel gelöst. Jeder Eintrag im Block Constraints besitzt unter anderem einen Gültigkeitsbereich (Scope), der angibt, auf welche Variablen sich das Constraint bezieht, und eine Referenz auf entweder eine Relation oder eine Funktion, welche zuvor spezifiziert wurden. Relationen erlauben die explizite Angabe von Werten oder Wertepaaren samt ihrer Kosten, während Funktionen Kosten aufgrund ihrer Eingabeparameter errechnen können. In der angegebenen Beispieldatei wurden exemplarisch die drei Funktionen für das Separieren, Gruppieren und Binden von VMs spezifiziert. Der Vorteil dieser Lösung liegt in ihrer Flexibilität, da benötigte Funktionen in der Eingabedatei spezifiziert werden können und nicht hart an ein Programm gebunden sind. Sollte in einem konkreten Fall eine zusätzliche Art von Constraint benötigt werden, so kann diese ohne Änderung am Algorithmus implementiert werden. Lediglich die Kapazitäten für virtuelle Maschinen und Server können in diesem Format nicht ohne Erweiterungen spezifiziert werden. Dies stellt jedoch keine Einschränkung dar, da man diese Daten sinnvollerweise aus dem Monitoring oder

Algorithm 5: Depth-first Branch and Bound Extended +

Function `BranchAndBound` $(t, v_t, k, \mathcal{X}, \mathcal{D}, \mathcal{C})$

1. **if** $(X = \varnothing)$ **then**
2. $\quad$ **return** lb;
3. **else**
4. $\quad i :=$ `ChooseVar` $(\mathcal{X})$;
5. $\quad$ **foreach** $a \in D_i$ **do**
6. $\quad\quad$ **if** $R_i < R_a$ **then**
7. $\quad\quad\quad R_a = R_a - R_i$;
8. $\quad\quad$ **else**
9. $\quad\quad\quad$ **continue**;
10. $\quad\quad \mathcal{DD} := \mathcal{D}; \mathcal{CC} := \mathcal{C}; Nt := t + (i, a); v_{Nt} := v_t \oplus F_i(a)$;
11. $\quad\quad$ `LookAhead` $(i, a, \mathcal{CC})$;
12. $\quad\quad$ **if** $($`LocalConsist` $(k, \mathcal{X} - \{i\}, \mathcal{DD}, \mathcal{CC}))$ **then**
13. $\quad\quad\quad k :=$ `BranchAndBound` $(Nt, v_{Nt}, k, \mathcal{X} - \{i\}, \mathcal{DD}, \mathcal{CC})$;
14. $\quad\quad R_a = R_a + R_i$;
15. **return** k;

Procedure `LocalConsist` $(k, \mathcal{X}, \mathcal{DD}, \mathcal{CC})$

16. **if** $(k >= lb)$ **then**
17. $\quad$ **return False**;
18. **else**
19. $\quad$ `W-AC*3` $(\mathcal{X}, \mathcal{DD}, \mathcal{CC})$;
20. $\quad$ **if** $\exists D_i \in \mathcal{DD} s.t. D_i = \varnothing$ **then**
21. $\quad\quad$ **return False**;
22. $\quad$ **return True**;

Procedure `LookAhead` $(i, a, \mathcal{CC})$

23. $\mathcal{CC} := \mathcal{CC} - \{C_i\}$;
24. **foreach** $C_{ij} \in \mathcal{CC}$ **do**
25. $\quad$ **foreach** $b \in D_j$ **do**
26. $\quad\quad F_j(b) := F_j(b) \oplus (C^0_{ij}(a, b) \ominus F(i, j, a) \ominus F(j, i, b))$;
27. $\quad \mathcal{CC} := \mathcal{CC} - \{C_{ij}\}$;

den erwähnten Heuristiken bezieht. Ein Beispiel einer Eingabedatei zeigt Listing 6.1.

```
<instance>
  <presentation name="virtual infrastructure" format="XCSP 2.1" type="WCSP">
    This is a WCSP instance for optimizing virtual infrastructures.
  </presentation>

  <domains nbDomains="1">
    <domain name="Servers" nbValues="3">
      <list> Server1 Server2 Server3 </list>
    </domain>
  </domains>

  <variables nbVariables="4">
    <variable name="VM0" domain="Servers"/>
    <variable name="VM1" domain="Servers"/>
    <variable name="VM2" domain="Servers"/>
    <variable name="VM3" domain="Servers"/>
  </variables>

  <relations nbRelations="0">
  </relations>

  <functions nbFunctions="3">

    <function name="separate" return="int">
      <parameters> int X int Y int costs </parameters>
      <expression>
        <functional> if(eq(X,Y),costs,initialCost) </functional>
      </expression>
    </function>

    <function name="group" return="int">
      <parameters> int X int Y </parameters>
      <expression>
        <functional> if(eq(X,Y),initialCost,costs) </functional>
      </expression>
    </function>

    <function name="pin" return="int">
      <parameters> int X int Y int costs</parameters>
      <expression>
        <functional> if(eq(X,Y),initialCost,costs) </functional>
      </expression>
    </function>

  </functions>

  <constraints nbConstraints="4" initialCost="0" maximalCost="10">
    <constraint name="C0" arity="3" scope="VM0 VM1 <i>5</i>" reference="separate" />
    <constraint name="C1" arity="3" scope="VM0 VM2 <i>3</i>" reference="group" />
    <constraint name="C2" arity="3" scope="VM1 "Server1" <i>1</i>" reference="pin" />
    <constraint name="C3" arity="3" scope="VM3 "Server2" <i>3</i>" reference="pin" />
  </constraints>
</instance>
```

Listing 6.1: Beispiel einer XCSP Eingabedatei

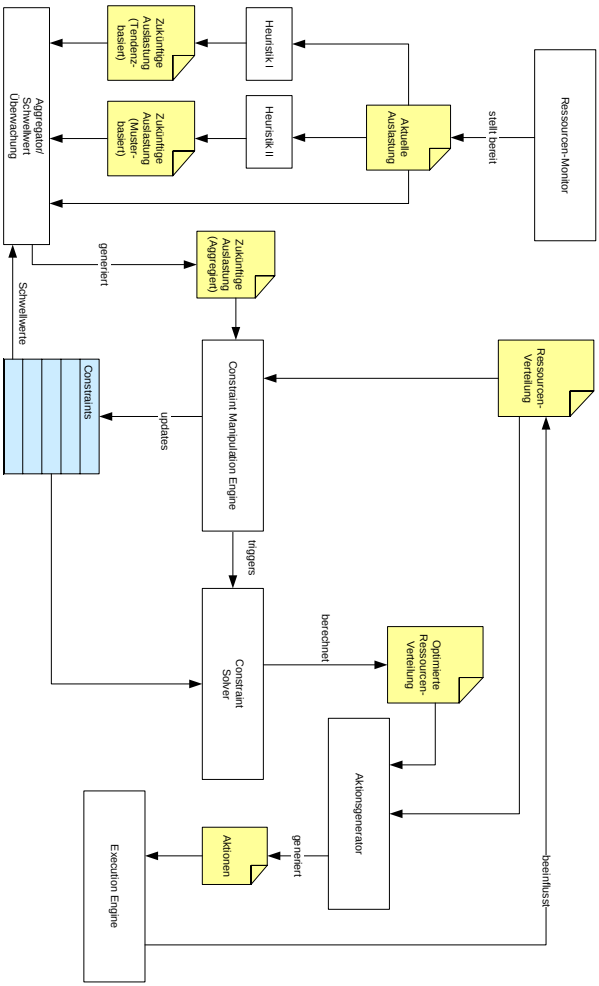


Abbildung 6.2: Architektur eines Ressourcen-Schedulers

6.2.4 Proof-of-Concept

Verhalten des entwickelten Algorithmus in der Praxis

Bereits der einfache Branch and Bound Algorithmus, wie er in [LaSc 04] beschrieben ist, vermeidet einen kompletten Durchlauf des Lösungsbaumes, indem Teilbäume, die aufgrund ihrer Kosten nicht mehr zu einer besseren Lösung führen können, abgeschnitten werden und somit der Raum für potentielle Lösungskandidaten unter Umständen stark eingeschränkt werden kann. Die Erweiterung dieses Ansatzes und die Integration von Kapazitätswerten von Servern und virtuellen Maschinen, welche implizit weitere harte Constraints für das WCSP darstellen, verbessert die Laufleistung des Algorithmus erneut, da aufgrund ihrer Existenz sehr viele weitere Teilbäume ausgeschlossen werden können. Die Zeit zum Finden der optimalen Belegung des WCSPs sinkt damit deutlich, während die Zeit zum Finden einer ersten Lösung zunächst zunimmt, da Lösungen aufgrund der höheren Anzahl an Constraints seltener werden. Diesem Nachteil wird entgegengewirkt, indem zunächst die Pfade mit den niedrigsten assoziierten unären Kosten durchlaufen werden. Dies entspricht einem Suchen von Lösungen, die der aktuellen Belegung möglichst ähnlich sind, also wenige Migrationskosten verursachen und aufgrund früherer Optimierungen bereits nahezu optimale Lösungen sind. Selbst wenn zuvor noch keine Optimierung durchgeführt wurde liefert dieser Greedy-Ansatz sehr schnell sehr gute Lösungskandidaten. Exemplarisch wurde der entwickelte Algorithmus an drei Verteilungsproblemen unterschiedlicher Größe getestet, die wie folgt spezifiziert waren:

- Szenario 1: 5 virtuelle Maschinen auf 3 Server verteilen
- Szenario 2: 25 virtuelle Maschinen auf 5 Server verteilen
- Szenario 3: 100 virtuelle Maschinen auf 5 Server verteilen

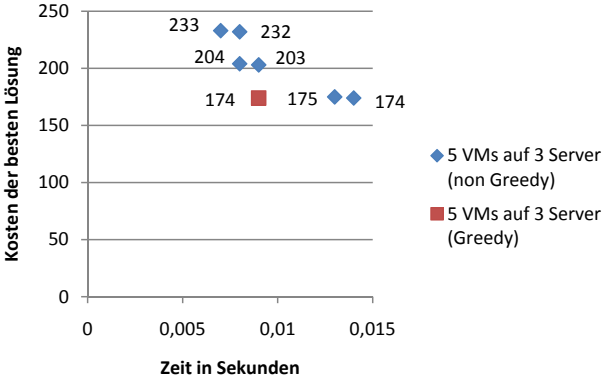


Abbildung 6.3: Vergleich der Algorithmen 5 VMs auf 3 Server

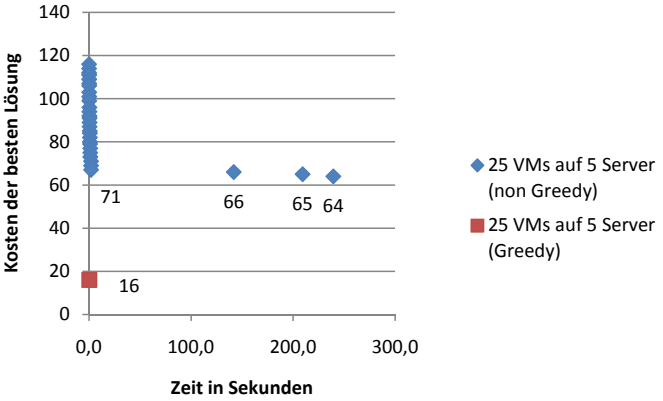


Abbildung 6.4: Vergleich der Algorithmen 25 VMs auf 5 Server

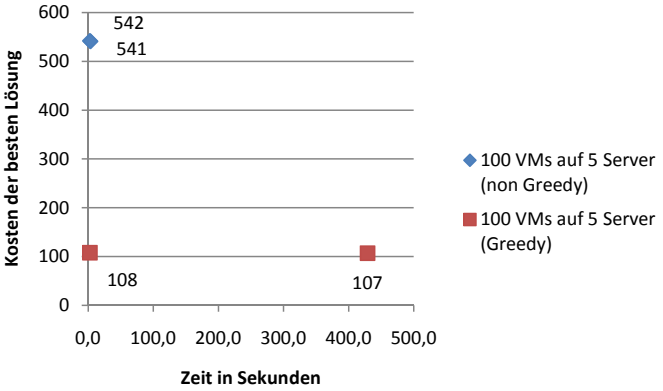


Abbildung 6.5: Vergleich der Algorithmen 100 VMs auf 5 Server

Spezifikation von drei Testszenarien für den Algorithmus

Jede der virtuellen Maschinen hatte eine vollständig spezifizierte Menge unärer Constraints sowie mindestens ein binäres Constraint zugeordnet. Die Constraints vom Typ Pin wurden mit zufälligen Kosten im Bereich 1 und 100 belegt. Platziert der Algorithmus eine virtuelle Maschine auf dem durch den Algorithmus präferierten Server, so fallen keine Kosten an. Wird die Maschine auf einen anderen als den bevorzugten Server gesetzt, so fallen die durch das Constraint spezifizierten Kosten an. Die Constraints vom Typ Group und Separate wurden ebenfalls mit

Kostenwerten zwischen 1 und 100 belegt, die, falls zwei Maschinen trotz Separate Anweisung auf demselben Server oder trotz Group Anweisung auf unterschiedlichen Servern platziert wurden, die Kosten der Gesamtlösung erhöhten. Für die Ermittlung der Kosten einer Lösung wurde der im Abschnitt 6.2.3 spezifizierte $\oplus$ - Operator (Kostenaggregatsfunktion) eingesetzt. Die obere Schranke der Gesamtkosten $\top$ wurde für das Szenario 1 im Experiment auf 300 festgelegt. Für die beiden größeren Szenarien wurde dieser Wert erhöht und testweise variiert. Dies konnte jedoch die Laufzeit in den ersten zehn Minuten nicht beeinflussen. Die Rate für die Anzahl der zufällig spezifizierten binären Group und Separate Constraints wurde im kleinsten Szenario auf einen Wert von 30% aller möglichen Constraints gesetzt. Für die beiden größeren Szenarien wurde die Constraint-Rate auf 10% reduziert, da die Modellierung von zehntausend Constraints im Falle des dritten Szenarios als unrealistisch erschien. Alle drei Szenarien wurden sowohl mit als auch ohne Greedy-Strategie getestet, wobei die beiden größeren Szenarien nach jeweils zehn Minuten abgebrochen wurden.

Verhalten des Algorithmus in den Testszenarien

Das kleine Szenario wurde durch beide Algorithmen innerhalb von 15 Millisekunden gelöst. Die folgenden Diagramme 6.3, 6.4, 6.5 zeigen die innerhalb der Laufzeit gefundenen Lösungskandidaten in Gegenüberstellung zur benötigten Rechenzeitzeit. Die Tests der Algorithmen zeigen, dass die Greedy-Strategie insbesondere bei größeren Szenarien sehr viel schneller bessere Lösungen findet. Dieser Vorteil wächst mit der Tatsache, dass durch die gefundene kostengünstige Lösung mehr Teilbäume ausgeschlossen werden können als dies ohne die Greedy-Strategie möglich ist. Die Integration der W-AC3 beziehungsweise W-AC*3 Algorithmen in den Branch and Bound Algorithmus wirkt sich hingegen negativ auf die Performanz des Algorithmus aus. Ein einzelner initialer Lauf des W-AC3 Algorithmus vor dem Start des Branch and Bound Algorithmus hingegen kann das Finden von Lösungen beschleunigen. Ein Test dieser Konstellation im Szenario 3 reduzierte die Zeit, die für das Finden der ersten Lösung erforderlich war, von 2,5 auf 1,8 Sekunden. Die Integration des W-AC*3 Algorithmus brachte keine weiteren Vorteile hinsichtlich der Laufzeit mehr, jedoch kann durch den Algorithmus sehr schnell eine untere Schranke an Kosten abgeschätzt werden, unter der das Problem nicht lösbar ist. Dies kann einen Vorteil für den Administrator darstellen, da dieser anhand dieses Wertes eventuell Inkonsistenzen in der Spezifikation seiner Verteilungsrichtlinien finden kann. Im Folgenden (Listing 6.2) werden die wesentlichen Abschnitte der Implementierung des Branch and Bound Algorithmus abgebildet. Auf ein Codelisting der Hilfsklassen sowie der ebenfalls implementierten W-AC3 und W-AC*3 Algorithmen wird aus Platzgründen verzichtet.

Sinnvolle Spezifikation von Kostenwerten

Die Belegung der Kosten in den drei beschriebenen Szenarien mit zufälligen Werten zeigt, das der Algorithmus an sich funktioniert. Die erzeugten Eingaberichtlinien in Form von Pin, Group und Separate Constraints an sich sind jedoch sinnlos. Welche Werte ein ein Administrator für die Kosten der einzelnen Constraints spezifiziert, bleibt letztlich ihm selbst überlassen. Im Folgenden werden jedoch noch einige Erfahrungswerte angegeben, welche die Belegung der Constraints mit

Kosten vereinfacht.

Spezifikation einer oberen Schranke

Die Experimente in den drei Szenarien haben ergeben, dass die Spezifikation einer oberen Schranke durch einen Administrator in größeren Szenarien nicht trivial, aber auch nicht notwendig ist. Vor dem Start des Algorithmus besitzt man in der Regel keine Information über die Größenordnung der Kosten der Lösungen, so dass man den Parameter $\top$ auf einen hinreichend großen Wert setzen sollte. Anderenfalls läuft man Gefahr, dass der Algorithmus gar keine Lösung für das spezifizierte Problem findet, da eventuell alle existierenden Lösungen größer als $\top$ sind. Der Algorithmus kann auf diese Weise zu Anfang nur wenige Lösungsbäume aufgrund der angegebenen oberen Schranke abschneiden. Dies stellt jedoch kein Problem dar, da der Algorithmus, wie die Beispiele zeigen, selbst sehr schnell eine gute Lösung findet und diese für das Ausschließen zu teurer Pfade von der Berechnung verwendet.

Verhältnis von Migrationskosten zu Effizienzgewinn

In der Praxis sollte weiter darauf geachtet werden, dass die Kosten der angegebenen Group- und Separate-Anweisungen in sinnvollem Verhältnis zu Pin-Anweisungen spezifiziert werden. Wählt man die Kosten der Constraints zum Binden von virtuellen Maschinen zu hoch, so besteht die Gefahr, dass gute Lösungen für die Verteilung der virtuellen Maschinen in der Infrastruktur nicht umgesetzt werden, weil die damit verbundenen Migrationskosten zu hoch spezifiziert sind. Migrationskosten treten jedoch nur einmal auf, von einer effizienteren Verteilung der Maschinen profitiert man längerfristig. Aus diesem Grund sollten die Kosten für Pin-Anweisungen kleiner gewählt werden als die Kosten der beiden anderen Constraint-Typen. Zudem empfiehlt es sich mehrere Kostenstufen für eine Operation zu verwenden, so dass eine Einteilung in Klassen wie ”Kann migriert werden”, ”Migration vermeiden” und ”Migration dringend vermeiden” möglich wird. Analoges gilt für Gruppierungs- und Separierungsanweisungen. Belegt man die Klassen beispielsweise mit den Werten 1, 3, und 5 für Pin-Anweisungen und 4, 6, und 8 für Group- beziehungsweise Separate-Anweisungen, so erreicht man das Gewünschte.

```
1  public static costs BranchAndBound(Hashtable t, costs vt, costs k, LinkedList Q){
   if(Q.size()==0){
3    if (vt.isSmaller(lb)){
       lb = vt.clone();
5      System.out.println(hashtableToString(t) + " ::: Kosten : " + lb);
     }
7    System.out.println(t + " ::: Kosten der aktuellen Belegung: " + vt + " ::: Kosten
         der bisher besten Belegung: " + lb);
     return lb;
9  }
   else {
11   variable i = (variable) Q.getFirst();
     domain d_i = i.getDomain();
13   Q.remove(i);
     for(int l=0; l<d_i.getValues().size(); l++){
15     element a = (element) d_i.getValues().get(l);
       if (!(a.addVM(i))){
17       System.out.println(a.leftCapacity);
         System.out.println("Kapazität von " + a + "erschöpft!");
19       continue;
       }
21     LinkedList QQ = deepClone(Q);
       Hashtable Nt = (Hashtable) t.clone();
23     Nt.put(i, a);

25     costs vNt = vt.clone();
       unaryConstraint c_i = i.getuConst();
27     vNt.oplus(c_i.getCosts(a), top);
       LookAhead(i,a,QQ);
29     System.out.println("Setze Variable " + i + " auf Wert " + a);
       if (LocalConsist(k, vNt, QQ)){
31       k = BranchAndBound(Nt, vNt, k, QQ);
       }
33     a.removeVM(i);

35   }
   }
37 return k;

39 }

41 public static void LookAhead(variable i,element a, LinkedList QQ){
       LinkedList bConstLst = i.getbConstLst();
43     for(int l=0; l<bConstLst.size(); l++){
         binaryConstraint c_ij= (binaryConstraint) bConstLst.get(l);
45       variable j = c_ij.getVar2();
         domain d_j = (domain) j.getDomain();
47       for(int m=0; m<d_j.getValues().size(); m++){
           element b = (element) d_j.getValues().get(m);
49         unaryConstraint c_j = (unaryConstraint) j.getuConst();
           costs c = c_j.getCosts(b);
51         c.oplus(c_ij.getCosts(a, b), top);
         }
53
       }
55 }

57 public static boolean LocalConsist(costs k, costs v, LinkedList QQ){
       //return true; //temporär!
59     if (lb.isLessOrEqual(v) && !(lb.equals(top))){
         System.out.println("Die entstehenden Kosten sind größer als die bisherige
             untere Schranke " + v + ">=" + lb + " : Abbruch des aktuellen Pfades");
61         return false;
       }
63     else {
         System.out.println("lb > k");
65   // An dieser Stelle kann bei Bedarf zyklisch der W-AC3 bzw. W-AC*3 Algorithmus
         aufgerufen werden
       //wac3(QQ, k);
67     return true;
       }
69 }
```

Listing 6.2: Auszug aus dem implementierten Constraint Solver

6.2.5 Zusammenfassung

Auswahl der Klasse an Algorithmen zur Lösung des Problems

Die Automatisierung der Umsetzung der im Absatz 6.1 vorgeschlagenen Verteilungsrichtlinien durch einen Algorithmus war das Hauptziel dieses Kapitels. Hierzu wurden zunächst zusätzliche Anforderungen an einen Algorithmus spezifiziert, die eine Verteilung der virtuellen Maschinen neben der Konformität zu den aufgestellten Richtlinien erfüllen muss. Diese waren neben Skalierbarkeit und Erweiterbarkeit, welche durch nahezu jeden Algorithmus realisiert werden sollten, Kosteneffizienz, Determinismus und die Fähigkeit, Server in ihrer Kapazität zu begrenzen, so dass diese durch virtuelle Maschinen gleichmäßig belegt werden können. Die Wahl einer Gruppe an Algorithmen, die für die Lösung dieses Problems geeignet ist, fiel auf die Gruppe der Constraint-Solver. Es existieren jedoch eine Menge weiterer Algorithmen beziehungsweise Klassen an Algorithmen, die für die Lösung des spezifizierten Problems durchaus in Frage kommen und im Vorfeld dieser Arbeit evaluiert wurden. Hierunter fallen unter anderem Binpacking-Algorithmen und Data-Mining-Ansätze, die ähnlich wie die Constraint-Solver einen Teil der Anforderungen erfüllen können, nicht jedoch alle. Die Klasse der Constraint-Solver stellte sich als die am einfachsten erweiterbare Klasse der erwähnten Ansätze dar. Analog zur Wahl der Klasse der Algorithmen war die Wahl des Depth-first Branch and Bound Algorithmus keine von Anfang an zwingende Entscheidung. Auch hier stellte sich zunächst die Wahl der Subklasse der gewichteten Constraint-Solver als am passendsten heraus, ehe der konkrete Algorithmus gefunden wurde und es sich zeigte, dass er um die fehlenden Eigenschaften erweitert werden kann.

Modellierung der Eingabeparameter

Auf Grundlage der Wahl des Algorithmus konnten die fehlenden Eigenschaften integriert und die aufgestellten Richtlinien für die Verteilung der virtuellen Maschinen passend modelliert werden. Bei den wichtigsten fehlenden Eigenschaften der Klasse der gewichteten Constraint-Solver handelt es sich um die Fähigkeit Ressourcen nur begrenzt oft zuzuteilen sowie die Laufzeit zu begrenzen, Ähnlichkeit in der Verteilung zu gewährleisten, wenn die Eingabeparameter ähnlich sind und Migrationskosten zu vermeiden. Eine zentrale Idee war es hier, einen Greedy-Ansatz über die angegebenen unären Constraints zu verfolgen. Für die Modellierung der Richtlinien konnte auf die XML-Sprache XCSP zurückgegriffen werden, die im Wesentlichen für den Austausch von Constraint Satisfaction Problemen geschaffen wurde, sich jedoch auch als Eingabeformat für Constraint-Solver eignet. Hierzu mussten die aufgestellten Verteilungsrichtlinien als XCSP-Funktionen modelliert werden.

Prototypische Implementierung und Evaluierung

Neben den konstruktiven Arbeiten dieses Kapitels war es eine weitere Leistung, die erweiterten Algorithmen mit all ihren Hilfsklassen prototypisch als Java-Klasse zu implementieren und deren Leistungsfähigkeit anhand einiger Szenarien zu evaluieren. Die Implementierung, wie sie im Listing 6.2 zu sehen ist, zeigt lediglich die wichtigsten Routinen, die die Logik des Algorithmus enthalten. Neben diesen Hauptroutinen sind eine Menge an Hilfsklassen notwendig, um den Algorithmus

ausführen zu können, da es zum Beispiel keine Standardklassen für die Modellierung von mathematischen Mengen und Constraints gibt.

6.3 Bewertung der Arbeit

Erhöhung des Wirkungsgrades

Das Ziel dieser Arbeit stellt es dar, Maßnahmen und Ansätze aufzuzeigen, die den Wirkungsgrad von Host-Virtualisierungslösungen im Kontext ganzer Infrastrukturen erhöhen können. Diese Ansätze umfassen sowohl das Aufzeigen von Punkten, die bei der initialen Planung hinsichtlich Performanz von virtuellen Infrastrukturen beachtet werden sollten, als auch Strategien, wie bestehende Infrastrukturen ohne Änderung an der Hardware vorzunehmen durch Beachten von Konfigurationsrichtlinien effektiv ausgelastet werden können, so dass virtualisierte Dienste mit optimaler Leistung bei minimalem Ressourcenverbrauch betrieben werden können. Die vorgestellten Lösungsansätze sparen daher nicht nur Ressourcen, sondern leisten auch einen Beitrag zum Schutz unserer Umwelt im Sinne der Green-IT und schonen das Budget eines Rechenzentrums, indem Strom- und Klimakosten reduziert werden können. Die folgenden beiden Abschnitte haben zum Ziel, die in dieser Arbeit diskutierten Fragestellungen zu bewerten. Der Abschnitt 6.3.1 wiederholt die in Kapitel 1 aufgeworfenen Fragen, während der Abschnitt 6.3.2 die zugehörigen Lösungsansätze bewertet.

6.3.1 Wiederholung der Fragestellungen

Definition eines Leistungsbegriffs

Einen zentralen Punkt dieser Arbeit stellt die Definition eines Leistungsbegriffes im Kontext der Virtualisierung dar. Diese Definition dient in der Arbeit als Metrik für die Optimalität von virtualisierten Infrastrukturen. Alle Ansätze, seien sie theoretischer oder empirischer Natur, basieren auf dieser Metrik. Eng verbunden mit der Definition einer Metrik stellt sich empirisch die Frage der Messbarkeit und nach Messtechniken für die Performanz von virtualisierten Systemen. Als Grundlage für Überlegungen in diesem Bereich existiert nahezu ausschließlich Literatur für die Durchführung von Benchmarks in physischen Systemen. Die Anpassung dieser Verfahren, so dass diese sinnvoll mit virtuellen Maschinen durchgeführt werden können, ist Teil dieser Arbeit.

Analyse von Virtualisierungsansätzen

Die Wahl der richtigen Virtualisierungslösung für einen definierten Einsatzzweck hängt meist, abgesehen von finanziellen Aspekten, von der Virtualisierungstechnik der möglichen Virtualisierungsprodukte ab. Das Verständnis dieser Techniken kann für die Entscheidung für oder gegen einen Lösungsansatz und ein Produkt extrem hilfreich sein. Im Rahmen dieser Arbeit werden daher alle gängigen Virtualisierungsansätze und Techniken beschrieben, verglichen und ihre Vor- und Nachteile hinsichtlich Performanz erläutert. Weiterhin ermöglicht die Kenntnis der eingesetzten Verfahren ein Verständnis für mögliche Konfigurationsparameter und

deren sinnvolle Belegung, so dass ein bisher nicht veröffentlichter Vergleich aller Virtualisierungsansätze einen lohnenden Beitrag darstellt.

Analyse von Konfigurationsparametern

Neben der Kenntnis von Virtualisierungstechniken ist das Wissen über potentielle Konfigurationsparameter und deren sinnvolle Belegung ein wichtiger Aspekt. Eine Fragestellung in dieser Arbeit war daher, theoretisch und empirisch zu analysieren, welche Parameter den Wirkungsgrad von Virtualisierung beeinflussen können.

Richtlinien zum Design virtueller Infrastrukturen

Basierend auf den Erkenntnissen der bereits aufgeführten Fragestellungen wurde im Kapitel 6.1.1 dieser Arbeit ein Katalog erstellt, der die Möglichkeiten zur Optimierung kompakt zusammenfasst und als Hilfe für Systemadministratoren dienen kann, um sich für eine Virtualisierungslösung zu entscheiden und initiale Konfigurationen vorzunehmen. Darüber hinaus ermöglicht die Betrachtung des absoluten Wirkungsgrades eine Abschätzung der notwendigen Ressourcen, die für die Virtualisierung einer gegebenen Infrastruktur notwendig sind beziehungsweise eine Beurteilung ermöglichen, ob eine Virtualisierung von einzelnen Systemen überhaupt sinnvoll erscheint.

Regelwerk für Ressourcenscheduler

Weiterhin können Erfahrungen über den Ressourcenverbrauch der Virtualisierungsschicht bei der Virtualisierung beziehungsweise Emulation bestimmter Aktionen Querabhängigkeiten von virtuellen Maschinen hilfreich bei der Vermeidung ebensolcher Situationen sein. Ein Scheduler kann, ein entsprechendes Regelwerk vorausgesetzt, virtuelle Maschinen mit extremen Querabhängigkeiten im Ressourcenbedarf gezielt auf verschiedenen physischen Systemen platzieren und negative Auswirkungen einer virtuellen Maschine auf die Performanz eines anderen virtuellen Systems vermeiden. Die Entwicklung eines solchen Regelwerkes ist mit den durchgeführten theoretischen Analysen und empirischen Untersuchen möglich und wird im Abschnitt 6.1.2 dieser Arbeit durchgeführt.

Algorithmus als Basis für einen Ressourcen-Scheduler

Bestehende Ressourcen-Scheduler erkennen zurzeit lediglich akute Engpässe verfügbarer Ressourcen auf einzelnen physischen Systemen. Diese Engpässe werden, sofern auf anderen Systemen Ressourcen zur Verfügung stehen, meist zuverlässig durch das Anstoßen geeigneter Migrationen gelöst. Nicht beachtet werden hierbei jedoch die oben beschriebenen Querabhängigkeiten von virtuellen Maschinen und deren Kommunikationsbeziehungen, wie sie das entwickelte Regelwerk spezifiziert. Die Entwicklung eines Algorithmus zur Verteilung beziehungsweise Umverteilung virtueller Maschinen unter Berücksichtigung der modellierten Abhängigkeiten ist Ziel der letzten Fragestellung dieser Arbeit.

6.3.2 Bewertung der Lösungsansätze

Im Folgenden werden die in dieser Arbeit gefundenen Lösungsansätze für die anfangs spezifizierten wissenschaftlichen Fragestellungen zusammengefasst und bewertet. Im Anschluss daran wird ein Ausblick auf noch offene Teilfragestellungen gegeben und es werden Anknüpfungspunkte für weitere wissenschaftliche Arbeiten diskutiert.

6.3.2.1 Leistungsbegriff

Der Wirkungsgrad als Leistungsbegriff einer virtuellen Maschine

Der Leistungsbegriff im Kontext Virtualisierung, wie er im Kapitel 3 dieser Arbeit definiert wurde, wurde anders als klassische Metriken zur Leistungsmessung nicht als absolute, sondern als relative Metrik definiert. Da die Betrachtung von Leistung in virtuellen Maschinen nicht nur von der virtuellen Maschine selbst, dem Hypervisor und der eigentlichen Hardware bestimmt wird, sondern auch von nebenläufigen virtuellen Maschinen, ist ein Vergleich von absoluten Metriken nur bedingt sinnvoll, zumal durch Migrationen auf andere physische Systeme das Bezugssystem wechseln kann. Zudem ist ein absolutes Ergebnis eines Benchmarks in einer virtuellen Maschine ohne Information über die Leistungsfähigkeit des physischen Systems wertlos. Interessanter und auch aussagekräftiger ist hier der Quotient aus den absoluten Messungen von virtueller und physischer Maschine, welcher dem Wirkungsgrad der virtuellen Maschine nach der klassischen Definition entspricht. Der Wirkungsgrad einer virtuellen Maschine ist abhängig von der Anwendung, die als Benchmark gewählt wurde. Da es unmöglich ist, alle existierenden Anwendungen als Benchmarks auszuführen und deren Wirkungsgrad aufzustellen, wurden exemplarisch synthetische Benchmarks für die einzelnen Hardwarekomponenten durchgeführt und der Wirkungsgrad dieser Komponenten bestimmt. Je nach Anwendungsprofil lassen sich mit diesen Ergebnissen analog zur Basis eines Erzeugendensystems zumindest Abschätzungen über die Leistungsfähigkeit einer gegebenen Anwendung treffen.

Entwicklung von Messverfahren

Neben der Definition von Leistung in einer virtuellen Maschine wurden in dieser Arbeit Messverfahren erarbeitet, die Messungen in einer virtuellen Maschine erlauben. Als kritisch stellen sich hier Zeitmessungen in der virtuellen Maschine heraus, da die Time Counter der virtuellen Maschine zum Teil nur schlecht synchronisiert sind. Als sichere Lösung wurden daher Verfahren entwickelt, die die benötigten Zeiten von externen Zeitquellen beziehen.

Bewertung

Ein Vergleich von Messergebnissen, die mit den erarbeiteten Messverfahren erstellt wurden, über die spezifizierte Metrik wurde im Kapitel 4.4 dieser Arbeit durchgeführt und konnte die theoretisch erarbeiteten Vermutungen großteils bestätigen. Ein wissenschaftlicher Beitrag dieser Arbeit liegt daher in der Spezifikation einer Leistungsmetrik, ohne die ein Großteil der weiteren Arbeit nicht möglich gewesen wäre.

6.3.2.2 Analyse von Virtualisierungsansätzen

Ein weiterer wissenschaftlicher Beitrag dieser Arbeit stellt die Analyse von Virtualisierungstechniken und eine Kategorisierung von Produkten dar. Hierzu wurden Produkte untersucht, deren Virtualisierungsansätze komponentenbezogen kategorisiert und deren verwendeten Verfahren hinsichtlich ihrer Leistungsfähigkeit analysiert wurden. Aufbauend auf dieser qualitativen Analyse und der Definition eines Leistungsbegriffes wurden im Kapitel 4 Messungen durchgeführt, um die

Ergebnisse der theoretischen Untersuchung zu bestätigen und zu quantifizieren. Ein vor den eigentlichen Messungen zu lösendes Problem war die Spezifikation eines systematischen Messplans, der allgemeine Gültigkeit besitzt und es erlaubt weitere Virtualisierungslösungen oder Nachfolgeversionen der getesteten Produkte zu untersuchen und mit vorhandenen Messergebnissen zu vergleichen. Ein objektiver Vergleich von Virtualisierungstechniken ist in der Literatur bisher nicht zu finden, so dass sowohl die theoretische als auch die empirische Analyse einen echten Mehrwert bedeutet.

6.3.2.3 Konfigurationsparameter

Ein Nebenprodukt der Analyse von Virtualisierungsansätzen ist die Gewinnung einer Menge an Konfigurationsparametern. Diese Menge an Parametern wurde den Schichten Hardware, Virtual Machine Monitor und virtueller Maschine beziehungsweise deren Betriebssystem zugeordnet. Aufgrund dieser Zuordnung ließ sich verhältnismäßig einfach eine Einteilung in statische und dynamische Parameter vornehmen. Die entstandenen Listen an Konfigurationsparameter wurden anschließend ergänzt und Kandidaten ausgewählt, welche empirisch weiter untersucht wurden. Ein noch nicht zufriedenstellender Aspekt in diesem Zusammenhang besteht darin, dass keine Möglichkeit gefunden wurde, systematisch alle möglichen Einflussparameter zu erfassen, sondern lediglich die Kernkomponenten betrachtet wurden. Parameter, die zum Beispiel den Zugriff auf entfernten Storage beeinflussen und damit einen direkten Einfluss auf den Durchsatz von Disk-IO besitzen, können auch den Wirkungsgrad der Speichervirtualisierung beeinflussen. Zu nennen sind hier exemplarisch Dateisysteme, Blockgrößen eines Dateisystems und die Art der Anbindung an den entfernten Speicher, welche zwar noch als Parameter identifiziert, aber aufgrund der Komplexität nicht weiter untersucht wurden. Die Kategorisierung der gefundenen Parameter in die Kategorien statisch und dynamisch hingegen ist allgemeingültig und beeinflusste die weitere Gliederung der Vorgehensweise in statische und dynamische Optimierungsansätze.

6.3.2.4 Richtlinien zum Design virtueller Infrastrukturen

Erstellen eines Leitfadens zum Aufbau virtueller Infrastrukturen

Ein Grund für die Motivation dieser Arbeit waren Probleme bei der Migration von existierenden Infrastrukturen auf virtuelle Infrastrukturen. Aufgrund des unbekannten Wirkungsgrades existierender Virtualisierungslösungen und den initial unbekannten Querabhängigkeiten von virtuellen Maschinen war vor dieser Arbeit weder eine Dimensionierung der Hardwarekomponenten zum Aufbau der virtuellen Infrastruktur möglich noch war es möglich gezielt grundlegende Konfigurationseinstellungen vorzunehmen und somit einen performanten Betrieb der virtualisierten Maschinen zu gewährleisten. Durch den aus den qualitativen und quantitativen Analysen gewonnenen Katalog aus Komponenten-bezogenen Wirkungsgraden (Kapitel 6.1.1) und Querabhängigkeiten (Kapitel 6.1.2)kann nun

jeder Administrator zumindest grob abschätzen, wie viele Server für den Betrieb einer virtuellen Infrastruktur benötigt werden, welche Ausstattungsmerkmale diese besitzen sollten und welche Konfigurationen für einen geordneten Betrieb notwendig sind. Aufgrund der unvollständigen Menge an untersuchten Konfigurationsparametern ist auch der aufgestellte Katalog nicht als final zu betrachten. Weitere untersuchte Parameter, die Auswirkungen auf einzelne virtualisierte oder emulierte Komponenten besitzen, werden voraussichtlich auch den Wirkungsgrad dieser Komponenten in speziellen Situationen beeinflussen, wenn auch keine allzu großen Abweichungen von den ermittelten Wirkungsgraden mehr zu erwarten sind.

Erstellen eines Regelwerks für ein globales Scheduling

Weiterhin wurde in der Arbeit aufgezeigt, dass unter bestimmten Bedingungen das Gruppieren und Separieren von virtuellen Maschinen auf ein beziehungsweise mehrere physische Systeme die Performanz der Systeme steigern kann. Das Zusammenstellen dieser Bedingungen in ein Regelwerk erlaubt weitere Konfigurationseinstellungen durch einen Administrator oder ist Grundlage für einen globalen Ressourcen-Scheduler. Bislang existierte weder ein Scheduler mit der Funktionalität virtuelle Maschinen bevorzugt zu gruppieren oder separieren noch existierte ein Regelwerk, dass diese Funktionalität unterstützen würde. Die Erstellung eines entsprechenden Regelwerkes war Bestandteil dieser Arbeit. Die Grundlage für einen globalen Ressourcen-Scheduler ist damit gelegt.

6.3.2.5 Entwicklung eines globalen Ressourcen-Schedulers

Regelwerk als Grundlage zur globalen Verteilung

Die Entwicklung eines Algorithmus zur globalen Verteilung von virtuellen Maschinen auf physische Server auf Basis des bereitgestellten Regelwerkes ist ein weiterer Beitrag dieser Arbeit. Die Entwicklung eines Algorithmus für einen globalen Ressourcen-Scheduler ist ein notwendiger Schritt für die Automatisierung von Lastausgleichsmechanismen unter Beachtung der aufgestellten Regeln. Neben den zu beachtenden Verteilungsregeln wurden weitere Kriterien bei der Verteilung von virtuellen Maschinen identifiziert und berücksichtigt. Ein Mechanismus zur Erweiterung des Regelwerks um weitere Anforderungen wurde ebenfalls spezifiziert.

Weitere berücksichtigte Anforderungen

Als eine wichtige Erweiterung des Regelwerks sind Kapazitäten von Servern und Ressourcenanforderungen von virtuellen Maschinen zu nennen, die die Verteilung in ihrer Komplexität, aber auch in ihren Möglichkeiten einschränken. Weiterhin sind Migrationen Vorgänge, die Betriebsressourcen kosten und daher selbst die Performanz von virtuellen Maschinen negativ beeinflussen können. Die neu errechnete Verteilung sollte damit eine möglichst hohe Ähnlichkeit zur bestehenden Verteilung besitzen, um nicht unnötig viele Migrationen durchführen zu müssen. Als Lösungsansatz für dieses Problem wurden für Migrationen virtuelle Kosten eingeführt, die sich durch das Erreichen einer besseren Verteilung, welche ebenfalls mit virtuellen Kosten anhand des Regelwerks versehen werden kann, amortisieren müssen, ehe eine Migration durchgeführt wird. Ein Problem bei der Lösung des spezifizierten Problems durch einen Algorithmus stellt die erwartete maximale

Laufzeit

Laufzeit dar. Da es sich bei dem aufgezeigten Problem um ein gewichtetes Constraint Satisfaction Problem handelt, das nur mit exponentiellem Aufwand optimal gelöst werden kann, müssen Heuristiken entwickelt werden, die die exponentielle Laufzeit des Problems begrenzen. Die in dieser Arbeit eingesetzte Greedy Strategie erfüllt diesen Zweck hervorragend und liefert bereits nach wenigen Millisekunden gute Ergebnisse, die alle spezifizierten Kriterien erfüllen.

7 Zusammenfassung und Ausblick

Zusammenfassung

Ein bislang wenig objektiv betrachtetes Gebiet im Themenbereich der Virtualisierung ist eine vergleichende Gegenüberstellung von Virtualisierungstechniken und Produkten. Bisher waren Administratoren, die eine Migration ihrer vorhandenen Infrastruktur in eine virtuelle Infrastruktur anstrebten, auf die Aussagen von Herstellern angewiesen, was Kapazitätsplanung betraf. Vergleiche von Produkten verschiedener Hersteller musste jeder Betreiber selbst aufwändig und kostenintensiv mit Teststellungen durchführen. Ziel dieser Arbeit war es, einen allgemeingültigen Katalog zu entwickeln, auf dessen Basis die Wahl einer Lösung vereinfacht und Kapazitätsplanung transparenter wird. Weiterhin wurden Konfigurationsvorschläge erarbeitet, die ein Einbrechen der Performanz im virtualisierten Zustand verhindern können. Um eine Automatisierung dieser Konfigurationen zu ermöglichen, wurde auf Basis eines Teils dieser Regeln ein globaler Ressourcen-Scheduler entwickelt, der eine Verteilung der virtuellen Maschinen auf physische Systeme der Infrastruktur gemäß der spezifizierten Richtlinien realisiert.

Weitere identifizierte Arbeiten

Der Fokus dieser Arbeit lag auf der Analyse von Virtualisierungstechniken für die verbreitete PCArchitektur, da diese zurzeit einem regen Interesse von Wissenschaft und Praxis unterliegt. Viele Erkenntnisse dieser Arbeit lassen sich voraussichtlich auf weitere Architekturen übertragen, gelten jedoch nicht pauschal für alle existierenden Architekturen. Um die Gültigkeit der wissenschaftlichen Erkenntnisse auf anderen Architekturen gewährleisten zu können, sind daher weitere Arbeiten notwendig. Zudem wurde der Fokus dieser Arbeit auf Host-Virtualisierung gelegt und die Untersuchung von Parametern auf dieses Themenfeld begrezt. Ähnliche Arbeiten in benachbarten Anwendungsfällen wie etwa der Speichervirtualisierung und der Netzvirtualisierung könnten interessante Ergebnisse liefern und von Bedeutung für die Effizienz von virtuellen Infrastrukturen sein. Weiterhin wurden Fragen im Themenumfeld dieser Arbeit identifiziert, die sich nicht oder nur unvollständig im Rahmen dieser Arbeit beantworten ließen. Eine offene Fragestellung zielt auf die mangelnde Synchronisation der Scheduler im Betriebssystem der virtuellen Maschine und des Hypervisors. Bedingt durch diese Asynchronität der beiden Scheduler können Echtzeitanwendungen derzeit nicht zuverlässig in virtuellen Maschinen ausgeführt werden. Die Synchronisation der Scheduler über eine Schnittstelle nach dem Vorbild der Paravirtualisierung könnte dieses Problem lösen, ist aber zum aktuellen Zeitpunkt noch nirgends implementiert oder spezifiziert. Der Algorithmus zur globalen Verteilung von Ressourcen wurde in dieser Arbeit nur mit Constraints belegt, deren Ziel es war, die Performanz virtueller Infrastrukturen zu erhöhen. Eine Grundidee beim Entwurf des Algorithmus war es jedoch, auf Flexi-

bilität und Erweiterbarkeit zu achten, so dass die Spezifikation weiterer Constraints aus verschiedenen Forschungsbereichen, wie etwa dem Security-Bereich, möglich ist und einen Gegenstand zukünftiger Arbeiten darstellen könnte. Im genannten Beispiel könnten mit Hilfe des etablierten Mechanismus virtuelle Maschinen konkurrierender Unternehmen in einer Cloud gezielt voneinander isoliert und auf unterschiedlichen physischen Systemen ausgeführt werden, so dass Industriespionage erschwert wird. Das Ausbrechen aus einer virtuellen Maschine aufgrund einer potentiellen Schwachstelle im Hypervisor kann auf diese Weise den Schaden begrenzen [DHLg 08]. Ein weiterer Anknüpfungspunkt liegt in der Erweiterung des Algorithmus um Bedingungen, deren Darstellung in binärer Form nicht ohne weiteres möglich ist. An einer entsprechenden Version des Branch-And-Bound-Algorithmus wird bereits gearbeitet [LaSc 04]. Ob die Anpassung auf das spezielle Problem der Verteilung von virtuellen Maschinen basierend auf dem neuen Basisalgorithmus möglich ist, bleibt zu untersuchen. Ein für die Praxis wichtiger Punkt ist die noch ausstehende Integration des entwickelten Algorithmus in die Management-Komponenten von Virtualisierungslösungen. Die Schwierigkeit, die mit dieser Aufgabe verbunden ist, ist dass nahezu alle existierenden Management-Lösungen kommerzielle Produkte sind und damit in der Regel auch nicht im Quellcode verfügbar sind. Eine Integration des Algorithmus in solche Produkte ist daher extrem aufwändig oder erfordert die Kooperation der Hersteller. Auch für die Entwickler von PC-Komponenten lassen sich weitere Aufgaben ableiten. Das Beispiel der iSCSI-HBAs zeigt deutlich, wie Hardware Virtualisierungsprozesse unterstützen kann. Für die noch aufwändigere Netzvirtualisierung existieren jedoch nur eingeschränkt Produkte, die den Hypervisor bei der Netzvirtualisierung unterstützen können. Hier müssen langfristig Produkte entwickelt werden, die ähnlich wie die Modi Intel VT-x und AMD-V kontextsensitiv und mandantenfähig Netzpakete senden und empfangen können und damit die Prozessoren der physischen Systeme sowie den Hypervisor entlasten.

A Linpack

Im Folgenden ist der Quellcode des Linpack-Benchmarks mit den in dieser Arbeit vorgenommenen Änderungen abgebildet. Ausgegraute Abschnitte wurden durch die auf sie folgenden Routinen ersetzt, um die Problematik der Zeitmessung im Virtuellen durch externe Zeitquellen zu korrigieren.

```
1  static REAL second(void)
       {
3      REAL help;
       bzero(buffer,256);
5      sprintf(buffer,"time\n");
       n = write(sockfd,buffer,strlen(buffer));
7      if (n < 0)
     error("ERROR writing to socket");
9      bzero(buffer,256);
       n = read(sockfd,buffer,255);
11     if (n < 0)
     error("ERROR reading from socket");
13     help = atof(buffer);
       help /= 1000000;
15
       bzero(buffer,256);
17     return help;
       }
```

Listing A.1: TimeClient für den Linpack-Benchmark (Auszug)

```
  /* Linpack Time-Server
2 **
  ** To compile:  cc -O -o linpackserver linpackserver.c
4 */

6 #include <stdio.h>
  #include <sys/types.h>
8 #include <sys/socket.h>
  #include <netinet/in.h>
10 #include <time.h>
  #include <sys/timeb.h>
12 #include <float.h>

14 typedef float REAL;

16 static struct timeval second   (void);
```

```
18
   void error(char *msg)
20 {
       perror(msg);
22     exit(1);
   }
24
   int main(int argc, char *argv[])
26 {
       int sockfd, newsockfd, portno, clilen;
28     char buffer[256];
       struct sockaddr_in serv_addr, cli_addr;
30     int n;
       struct timeval tv1, starttime;
32
       if (argc < 2) {
34         fprintf(stderr,"ERROR, no port provided\n");
           exit(1);
36     }
       sockfd = socket(AF_INET, SOCK_STREAM, 0);
38     if (sockfd < 0)
          error("ERROR opening socket");
40     bzero((char *) &serv_addr, sizeof(serv_addr));
       portno = atoi(argv[1]);
42     serv_addr.sin_family = AF_INET;
       serv_addr.sin_addr.s_addr = INADDR_ANY;
44     serv_addr.sin_port = htons(portno);
       if (bind(sockfd, (struct sockaddr *) &serv_addr,
46              sizeof(serv_addr)) < 0)
                error("ERROR on binding");
48     printf("Linpack Time-Server started\n");
       listen(sockfd,5);
50     clilen = sizeof(cli_addr);
       newsockfd = accept(sockfd,
52                  (struct sockaddr *) &cli_addr,
                    &clilen);
54     if (newsockfd < 0)
           error("ERROR on accept");
56
       printf("Client connected\n");
58
       while (1)
60     {
        bzero(buffer,256);
62      n = read(newsockfd,buffer,255);
        if (n < 0) error("ERROR reading from socket");
64
    if (strcmp(buffer,"start\n") == 0)
66        {
      bzero(buffer,256);
68          starttime = second();
      n = write(newsockfd,"start",7);
70    if (n < 0) error("ERROR writing to socket");
```

```
            }
72
74
    if (strcmp(buffer,"quit\n") == 0)
76  {
            bzero(buffer,256);
78    printf("Goodbye\n");
      n = write(newsockfd,"quit",6);
80    if (n < 0) error("ERROR writing to socket");
      bzero(buffer,256);
82    shutdown (sockfd,2);
      shutdown (newsockfd,2);
84    return 0;
    }
86
    if (strcmp(buffer,"time\n") == 0)
88        {
          bzero(buffer,256);
90    tv1 = second();
    //  printf("Timefunction message: %d\n",((tv1.tv_sec -
        starttime.tv_sec)*1000000)+(tv1.tv_usec-starttime.
        tv_usec));
92    sprintf(buffer,"%d\n",((tv1.tv_sec - starttime.tv_sec)
          *1000000)+(tv1.tv_usec-starttime.tv_usec));
          n = write(newsockfd,buffer,strlen(buffer));
94    if (n < 0) error("ERROR writing to socket");
        }
96
    bzero(buffer,256);
98    }
      shutdown (sockfd,2);
100   shutdown (newsockfd,2);
      return 0;
102 }
104
  static struct timeval second(void)
106 {
    struct timeval tv1;
108 gettimeofday(&tv1);
    return tv1;
110 }
```

Listing A.2: TimeServer für den Linpack-Benchmark

Literaturverzeichnis

[AdAg 06] ADAMS, KEITH und OLE AGESEN: *A comparison of software and hardware techniques for x86 virtualization*. In: *ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, Seiten 2–13, New York, NY, USA, 2006. ACM.

[AMD 05] AMD: *Secure Virtual Machine Architecture Reference Manual*, Mai 2005, `http://enterprise.amd.com/downloadables/Pacifica_Spec.pdf` .

[AMD 08] AMD: *Quad-Core AMD Opteron Processor with Direct Connect Architecture*, Juli 2008, `http://www.amd.com/us-en/assets/content_type/DownloadableAssets/44226A_4P_Arch.pdf` .

[ARC 07] *A brief architecture overview of VMware ESX, XEN and MS Viridian*, April 2007, `http://it20.info/blogs/main/archive/2007/06/17/25.aspx` . VMworld 2005.

[ARC 08] *Review: Microsoft's Hyper-V puts VMWare and Linux on notice*, Februar 2008, `http://blogs.zdnet.com/microsoft/?p=1182` . Blog on ZDNet.

[AW 08] ADDISON-WESLEY: *Funktionsweise des Paging in der x86-Architektur*, 2008, `http://images.tecchannel.de/images/tecchannel/bdb/339476/E10F5E11ABF452F80797EE6B79A1A68B_1000x700.jpg` . Schematische Darstellung.

[BDF+ 03] BARHAM, PAUL, BORIS DRAGOVIC, KEIR FRASER, STEVEN HAND, TIM HARRIS, ALEX HO, ROLF NEUGEBAUER, IAN PRATT und ANDREW WARFIELD: *Xen and the art of virtualization*. In: *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, Seiten 164–177, New York, NY, USA, 2003. ACM.

[Dani 09] DANIEL SCHEIBLI, MING ZHANG: *IOMeter Projekt*, September 2009, `http://www.iometer.org/` .

[DHLg 08] DANCIU, V., W. HOMMEL, T. LINDINGER und N. GENTSCHEN FELDE: *Adaptive defense measures against the security hazards induced by systems virtualisation*. In: *Proceedings of the 2008*

Workshop of HP Software University Association (HP–SUA), Band 2008, Marrakech, Morocco, Juni 2008. Hewlett–Packard, Infonomics–Consulting.

[Doro 07] DOROFEEV, ANDREI: *ESX Server CPU Scheduling*, September 2007, `http://www.vmworld.com/vmworld/static/sessions/2007/IO19.html` . Vortrag IO19.

[Drum 08] DRUMMONDS, SCOTT: *Time-based Measurements in Virtual Machines*, Mai 2008, `http://communities.vmware.com/docs/DOC-5581` .

[Gebh09] GEBHARDT, BASTIAN: *Empirische Identifikation von Parametern mit Einfluss auf die Effizienz von Virtualisierung am Beispiel von VMware ESXi*. Fortgeschrittenenpraktikum, Ludwig-Maximilians-Universität München, November 2009.

[GSS 08] GERHOLD, S., M. SONNENFROH und P. SCHULTHESS: *Seminar Betriebssysteme SS07*, 2008, `http://www-vs.informatik.uni-ulm.de/wiki/pub/Main/BetriebsSystemeSS07/BS_Kapitel13-4.pdf` . Virtualisierung, Speicherzugriffsverfahren.

[HAN 99] HEGERING, H.-G., S. ABECK und B. NEUMAIR: *Integrated Management of Networked Systems – Concepts, Architectures and their Operational Application*. Morgan Kaufmann Publishers, ISBN 1-55860-571-1, 1999. 651 p.

[HePa 94] HENNESSY, JOHN L. und DAVID A. PATTERSON: *Rechnerarchitektur*. vieweg, ISBN 3-528-05173-6, 1994.

[INTE 05] INTEL®: *Virtualization Technology Specification for the IA-32 Intel® Architecture*, April 2005, `ftp://download.intel.com/technology/computing/vptech/C97063-002.pdf` .

[KBKK 06] KHANNA, G., K. BEATY, G. KAR und A. KOCHUT: *Application Performance Management in Virtualized Server Environments*. Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP, Seiten 373–381, 0-0 2006.

[LaSc 04] LARROSA, JAVIER und THOMAS SCHIEX: *Solving weighted CSP by maintaining arc consistency*. Artif. Intell., 159(1-2):1–26, 2004.

[Lemb09] LEMBERGER, GÜNTER: *Empirische Identifikation von Parametern mit Einfluss auf die Effizienz von Virtualisierung am Beispiel von Xen*. Fortgeschrittenenpraktikum, Ludwig-Maximilians-Universität München, November 2009.

[Lind 06] LINDINGER, TOBIAS: *Virtualisierung einer Praktikumsinfrastruktur zur Ausbildung im Bereich vernetzter Systeme*, Mai 2006, `http://www.nm.ifi.lmu.de/pub/Diplomarbeiten/lind06` . Diplomarbeit.

[LJL+ 09] LIU, HAIKUN, HAI JIN, XIAOFEI LIAO, LITING HU und CHEN YU: *Live migration of virtual machine based on full system trace and replay*. In: *HPDC '09: Proceedings of the 18th ACM international symposium on High performance distributed computing*, Seiten 101–110, New York, NY, USA, 2009. ACM.

[Lo 05] LO, JACK: *VMware and CPU Virtualization Technology*, Oktober 2005, `http://download3.vmware.com/vmworld/2005/pac346.pdf` . VMworld 2005.

[Math 07] MATHAI, JACOB: *Xen Scheduling*, Juni 2007, `http://wiki.xensource.com/xenwiki/Scheduling` .

[MBT 08] MCCREADY, BRUCE, KENNETH C. BARR und KIRAN TATI: *ESX Server Best Practices for Performance*, September 2008, `http://mylearn.vmware.com/courseware/26148/TA2550.PDF` .

[MCZ 06] MENON, ARAVIND, ALAN L. COX und WILLY ZWAENEPOEL: *Optimizing Network Virtualization in Xen*. In: *USENIX Annual Technical Conference*, 2006. Best paper award.

[OVZW08] PROJECT, OPENVZ: *Virtual Ethernet device*, Mai 2008, `http://wiki.openvz.org/Virtual_Ethernet_device` .

[PoGo 73] POPEK, GERALD J. und ROBERT P. GOLDBERG: *Formal requirements for virtualizable third generation architectures*. In: *SOSP '73: Proceedings of the fourth ACM symposium on Operating system principles*, Band 7, New York, NY, USA, October 1973. ACM Press, `http://portal.acm.org/citation.cfm?id=808061` .

[Prat 05] PRATT, IAN: *Xen Status Report*, Dezember 2005, `http://www.cl.cam.ac.uk/netos/papers/ian-status.ppt` .

[Rhet 09] RHETT M. HOLLANDER: *IOMeter Projekt*, Oktober 2009, `http://www.alasir.com/software/ramspeed/` .

[RoLe 09] ROUSSEL, O. und C. LECOUTRE: *XML Representation of Constraint Networks: Format XCSP 2.1*. ArXiv e-prints, Februar 2009.

[Roma09] ROMANYUK, ANTON: *Empirische Identifikation von Parametern mit Einfluss auf die Effizienz von Virtualisierung am Beispiel von MS Hyper-V*. Fortgeschrittenenpraktikum, Ludwig-Maximilians-Universität München, November 2009.

[Rudo 08] RUDOLPH, JOHANNES: *Windows parametrisieren*, 2008, http://virtual-void.net/projects/bootpgm/wiki/Manual . Ringkonzept der x86 Architektur, Systemaufrufe.

[spe 08] *Standard Performance Evaluation Corporation*, Juni 2008, http://www.spec.org .

[Stil 07] STILLER, ANDREAS: *Magazin für Computertechnik: Xen und die virtuelle Zeit*, März 2007, http://www.heise.de . Seite 180.

[Stra 59] STRACHEY, CHRISTOPHER: *Time sharing in large, fast computers*. In: *IFIP Congress*, Seiten 336–341, 1959.

[Stra 06] STRANDBERG, KEN: *Performance Impacts with Optimized Virtual Environments on Intel Virtualization Technology-based Platforms*, 2006, http://oss.intel.com/pdf/virtualization_performance_impacts.pdf . Intel Whitepaper.

[Tane 02] TANENBAUM, ANDREW S.: *Moderne Betriebssysteme*. Pearson Studium, ISBN 3-8273-7019-1, 2002.

[Tsio09] TSIOUPROU, EVANGELIA: *Empirische Identifikation von Parametern mit Einfluss auf die Effizienz von Virtualisierung am Beispiel von OpenVZ/Virtuozzo*. Fortgeschrittenenpraktikum, Ludwig-Maximilians-Universität München, November 2009.

[VMwa 06] VMWARE, INC.: *Performance Benchmarking Guidelines for VMware Workstation 5.5*, April 2006, http://www.vmware.com/pdf/WS55_Benchmarking_Guidelines.pdf .

[VMwa 07] VMWARE, INC.: *Performance Tuning Best Practices for ESX Server3*, Januar 2007, https://www.vmware.com/pdf/vi_performance_tuning.pdf .

[VMwa 08] VMWARE INC.: *Performance Comparison of Virtual Network Devices*, März 2008, http://www.vmware.com/files/pdf/perf_comparison_virtual_network_devices_wp.pdf .

[Wald] WALDSPURGER, CARL A.: *Memory Resource Management in VMware ESX Server*. , http://www.usenix.org/events/osdi02/tech/waldspurger.html .

[Wiki 08] WIKIPEDIA: *Wikipedia - Die freie Enzyklopädie*, 2008, http://de.wikipedia.org . Online Enzyklopädie.

[ZhDo 08] ZHANG, XIANTAO und YAOZU DONG: *Optimizing Xen VMM Based on Intel® Virtualization Technology*. Internet Computing in Science and Engineering, 2008. ICICSE '08. International Conference on, Seiten 367–374, Jan. 2008.

Abkürzungsverzeichnis

ABI Application Binary Interface

AMD Advanced MicroDevices inc.

BIOS Basic Input/Output System

CPU Central Processing Unit

CSP Constraint Satisfaction Problem

DDR Double Data Rate

DEV Device Exclusion Vector

DMA Direct Memory Access

EAX Extended Accumulator

FLOPS Floating Point Operations Per Second

FSB Front Side Bus

GB Giga Byte

HA High Availability

HPC High Performance Computing

HVM Hardware-Virtual-Machine

IBM International Business Machines

IDE Integrated Drive Electronics

ISA Instruction Set Architecture

KB Kilo Byte

LAN Local Area Network

MB Mega Byte

MIPS Million Instructions Per Second

MMC Microsoft Management Console

MMU Memory Management Unit

NFS Network File System

NUMA Non Uniform Memory Access

PCI Peripheral Component Interconnect

PM Physical Machine

PSW Programm Status Wort

PV Para-Virtualization

QCSP Quantified Constraint Satisfaction Problem

RAM Random Access Memory

SATA Serial Advanced Technology Attachment

SBE Scan Before Execution

SCSI Small Computer Systems Interface

SDRAM Synchronous Dynamic Random Access Memory

SPEC System Performance Evaluation Corporation

SVM Secure Virtual Machine

SYSCALL System-Call

TCP Transmission Control Protocol

TLB Translation Lookaside Buffer

TPM Trusted Platform Module

VM Virtual Machine

VMCS (Virtual Machine Control Structure

VMM Virtual Maschine Monitor

VMPTRLD Load Pointer to Virtual-Machine Control Structure

VMPTRST Store Pointer to Virtual-Machine Control Structure

VMX Virtual Machine eXtension

VT Virtualization Technology

WCSP Weighted Constraint Satisfaction Problem

XML eXtensible Markup Language

Index