INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Analyse von Hotplug Mechanismen für Linuxsysteme zur Erhöhung der Performanz in virtuellen Maschinen

Michael Goldberger

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Analyse von Hotplug Mechanismen für Linuxsysteme zur Erhöhung der Performanz in virtuellen Maschinen

Michael Goldberger

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Dr. Nils gentschen Felde Tobias Lindinger Hans-Joachim Picht (IBM)
Abgabetermin:	13. Januar 2010

Inhaltsverzeichnis

1 Aufgabenstellung

Auf Grund wirtschaftlicher Faktoren gewinnt die Virtualisierung von IT-Systemen aktuell immer mehr an Bedeutung. Daher stehen sie auch im Fokus vieler wissenschaftlicher Arbeiten. Leider erreichen virtuelle Maschinen oft die Performanz ihrer physischen Gegenstücke nicht. Ein Grund hierfür sind einfach implementierte Schedulingalgorithmen, die Ressourcen wenig intelligent verteilen. Sie teilen virtuellen CPUs Rechenzeit zu, obwohl diese gerade keine benötigen. Um dem entgegen zu wirken hat IBM einen Linux Hotplug Daemon namens *cpuplugd für System z* entwickelt. Er ist in der Lage, die Anzahl der aktiven virtuellen CPUs einer Domäne dynamisch dem jeweiligen Bedarf anzupassen. Da dieser speziell für *z/VM* Hypervisor or *LPAR (PS/SM)* Hypervisor entwickelt wurde und dort zu signifikanten Effizienzsteigerungen geführt hat, ist im Zuge dieses Praktikums zu prüfen, ob eine Portierung auf einen anderen Hypervisor ebenfalls die Performanz der virtualisierten Systeme steigert. Die hier zu untersuchende Virtualisierungslösung ist *Xen*. Zur Evaluierung ist eine *Xen*-Testumgebung samt einer aktuellen Linux Version aufzubauen und mit geeigneten Tools die Leistungsunterschiede bei verschiedenen Belastungen der Domänen, mit und ohne *cpuplugd*, zu vergleichen. Gegebenenfalls soll das Scheduling von *Xen* genauer untersucht werden, um die Ergebnisse deuten zu können. Ein speziell für *Xen* angepasster Daemon wurde von IBM bereitgestellt.

Im weiteren Verlauf dieser Ausarbeitung soll nun, um eine Entscheidung zu treffen, ob der Einsatz des cpuplugd unter Xen zu Empfehlen ist, zuerst im Kapitel „Grundlagen“ ein kurzer Einblick in den Aufbau und die Arbeitsweise des Hypervisor Xen gegeben werden. Genauer wird hier auf den standardmäßig eingesetzten Scheduler, den Credit Based Schedulers, eingegangen. Basis ist hier die Beschreibung auf der Projekthomepage, Xen.org. Im selben Kapitel folgt eine detaillierte Betrachtung des cpuplugd Daemons.
Im darauf folgendem Kapitel „Tests“ wird zunächst die für die anschließenden Benchmarktests eingesetzte Hard- und Software vorgestellt und der Versuchsaufbau skizziert. Die darauf folgend vorgestellten Ergebnisse bieten eine erste Grundlage um die Arbeitsfähigkeit des Daemons unter Xen zu bewerten. Sie stellen die erreichte Performanzsteigerung, also das Zentrale Thema, dar. Auch wird in diesem Kapitel die Arbeitsweise des Credit Based Schedulers auf Basis des Xen Quellcodes etwas genauer betrachtet um eine Entscheidung abseits des reinen Performanzgewinns zu treffen. Auf Basis der vorliegenden Daten folgt hier auch eine erste Übersicht über Vor- und Nachteile durch den Einsatz des cpuplugd. Diese werden im Kapitel „Fazit“ nochmals aufgegriffen und abschießend bewertet.

2 Grundlagen

Um die Grundlagen für die folgenden Tests zu schaffen, soll dieses Kapitel einen kurzen Überblick über den eingesetzten Hypervisor „Xen“ und das Scheduling seiner virtuellen CPUs geben. Auch werden hier die Arbeitsweise und die Konfigurationsmöglichkeiten des cpuplugd genauer betrachtet.

2.1 Eine kurze Einführung in Xen

Der Xen Hypervisor läuft direkt auf der nativen Hardware. Mit ihm wird die privilegierte virtuellen Maschinen, Dom0, gestartet. Über diese werden die weiteren, unprivilegierten, Domänen, DomUs, geladen und konfiguriert. Die Dom0 muss dabei besonders für ihre Aufgabe angepasst sein, wohingegen die DomUs für den Einsatz in Xen angepasst, und damit paravirtualisiert, oder nativ, mit entsprechender Hardwareunterstützung, betrieben werden können. Abbildung 2.1 zeigt den Aufbau der Xen Architektur. Das zentrale Xen-Verwaltungsprogramm heißt *xm* (vergl. Manual xm [MXM]) und befindet sich im Domain Management and Control (DM&C). Einen tieferen Einblick über den Aufbau und die Arbeitsweise von Xen gewähren die Dokumentationsseiten des Xen Projekts. [DXP]

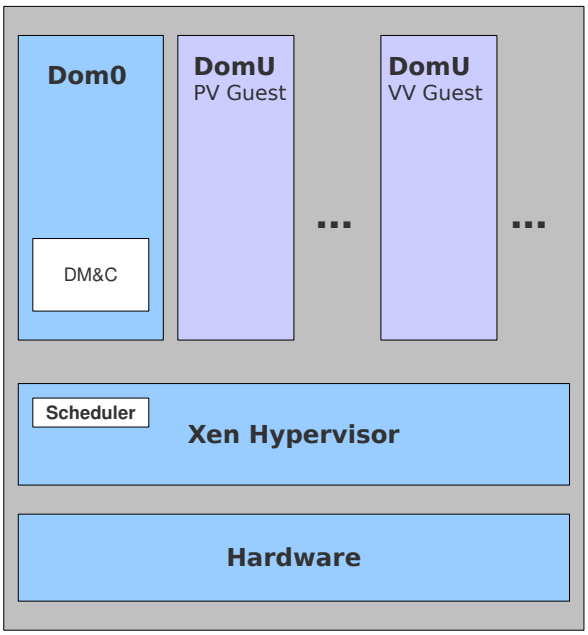

Abbildung 2.1: Xen Architekturbild

Der standardmäßig eingesetzte Scheduler, welcher auch über die Dom0 und das Tool *xm* gesteuert wird, ist der Credit based CPU Scheduler. Er beeinflusst die Ressourcenverteilung

der CPU-Zyklen indem den Domänen verschiedene Gewichtungen (*weight*) und Gewichtsobergrenzen (*cap*) zugeteilt werden. Jede physische CPU, gemeint ist damit ein Kern, managed dabei eine eigene Warteschlange in die sich die zugehörigen virtuellen CPUs einer Domäne gemäß ihrer Priorität einreihen. Es gibt die Prioritäten *under* und *over*. Under bedeutet dabei, dass die virtuelle CPU über sogenannte *Credits* verfügt, welche ihr z.B. beim Start zugewiesen wurden. *Over* heißt, dass diese verbraucht sind. Bekommt eine virtuelle CPU Rechenzeit zugewiesen, ist sie also zum Entscheidungszeitpunk an oberster Stelle der Warteschlange, verbraucht sie dabei *Credits*, muss sie in der Queue warten bekommt sie welche gutgeschrieben. Anzumerken ist, dass falls in einer lokalen Warteschlange keine virtuelle CPU mehr mit dem Status under vorhanden ist, bei Systemen mit mehreren Kernen erst in einer anderen Warteschlange nach einer virtuellen CPU mit diesem Status gesucht wird, bevor einer wartenden virtuellen CPU mit dem Status over Rechenzeit zugeteilt wird. Das Scheduling einer virtuellen CPU stellen Abb.2.2 und Abb. 2.3 vereinfacht dar.
Um einer Domäne mehr Rechenzeit als einer anderen zuzuteilen, kann ihr *weight*-Wert erhöht werden. Voreingestellt ist ein Wert von 256. Wird einer Domäne ein anderer Wert zugewiesen, so berechnet sich die ihr zugeteilte CPU-Zeit als Anteil dieses Werts an der Summe der Gesamtwerte aller Domänen.
Der *cap*-Wert stellt die Obergrenze der nutzbaren CPU-Zyklen einer Domäne dar. Er wird als Prozentwert im Verhältnis zu einer physischen CPU angegeben. Bei einem Single Prozessor System beträgt der Maximalwert also 100, bei einem Multi Prozessor System addieren sich die Werte entsprechend. Ein Wert von 200 würde daher bedeuten, dass die Domäne die kompletten Zyklen von 2 CPUs nutzen darf.
Die Parameter des Credit-Based CPU Schedulers lassen sich für jede Domäne einzeln mit dem Kommando *xm sched-credit -d <domain >* plus dem Flag -w bzw. -c und den zugehörtigen Werten einstellen (vergl. [CrS]).

Alternativ steht der Simple Earliest Deadline First (SEDF) Scheduler zu Verfügung. Er soll jedoch in kürze komplett vom Credit based Scheduler abgelöst werden und wird deshalb hier nicht näher erläutert.

2.2 CPU Hotpluging mit dem cpuplugd

Grundgedanke bei der Entwicklung des cpuplugd waren naive Schedulingalgorithmen, die bei einigen gängigen Virtualisierungslösungen zum Einsatz kommen. Dabei wird die zu Verfügung stehende Rechenleistung gleichmäßig auf alle vorhandenen virtuellen CPUs einer virtuellen Maschine verteilt. Nutzt diese jedoch, wie bei single threaded Anwendungen, nur einen virtuellen Prozessor, so geht die Rechenzeit, die für die anderen evtl. vorhanden virtuellen CPUs bereitgestellt wird, verloren. Diese Leistung könnte einer anderen virtuellen Maschine zugewiesen werden.
Der cpuplugd bietet nun die Möglichkeit die Anzahl der aktiven virtuellen CPUs dynamisch nach einem Satz vorzugebender Regeln anzupassen. Er wird direkt in den jeweils zu überwachenden Domänen installiert, konfiguriert und gestartet. Als Grundlage für das Hotpluging der virtuellen CPUs dienen der „Load Average“, welcher der Datei */proc/loadavg* entnommen wird, sowie die „Idle Percentage“, welche aus */proc/stat* ermittelt wird. Weiter Informationen, wie die Anzahl der laufenden virtuellen CPUs, kommen aus den Dateien unter dem Verzeichnis */sys/devices/system/cpu*. Als Parameter können in der Konfigurati-

onsdatei die minimale und maximale Anzahl der virtuellen CPUs, das Updateintervall sowie diverse Regeln für das Hotpluging und Hotunpluging angegeben werden. Die Untergrenze der virtuellen CPUs ist 1 die Obergrenze ist die vom Hypervisor vorgegebene Anzahl. Ein Eintrag in dem configuration File könnte wie folgt aussehen:

CPU_MIN=“1“
CPU_MAX=“2“
UPDATE=”10“
HOTPLUG = „(loadavg>onumcpus + 0.75) & (idle<10.0))“
HOTUNPLUG = „(loadavg<onumcpus - 0.25) |(idle>50))“

Das Updateintervall wird in Sekunden angegeben, die Voreinstellung beträgt 10 Sekunden. Dies macht Sinn, da z.B. in */proc/loadavg* der Durchschnitt über den kleinsten Zeitraum über 60 Sekunden berechnet wird. Die Hotplug Regel in diesem Beispiel gibt an, dass eine virtuelle CPU online geschallten wird, wenn der aktuelle „load average“ (loadavg) größer als die Anzahl der der momentan online geschalteten virtuellen CPUs (onumcpus) plus 0.75 ist und die aktuelle „idle percentage“ (idle) unter 10 % liegt. Die Hotunplug Regel deaktiviert hier eine virtuelle CPU, falls der „load average“ unter die Anzahl der momentan online geschalteten virtuellen CPUs minus 0.25 fällt oder die „idle percentage“ über 50% liegt.
Der Daemon wird über das Kommando *cpuplugd -c <config file >* gestartet. Das eigentliche Hotpluging geschieht, nach Auswertung der angegebenen Regeln, dann durch eintragen einer 0 für offline oder einer 1 für online in der Datei */sys/devices/system/cpu/cpu[CpuNumber]/online*. (vergl. [MCp09]Seite. 411f. und Manual PROC [PRO])

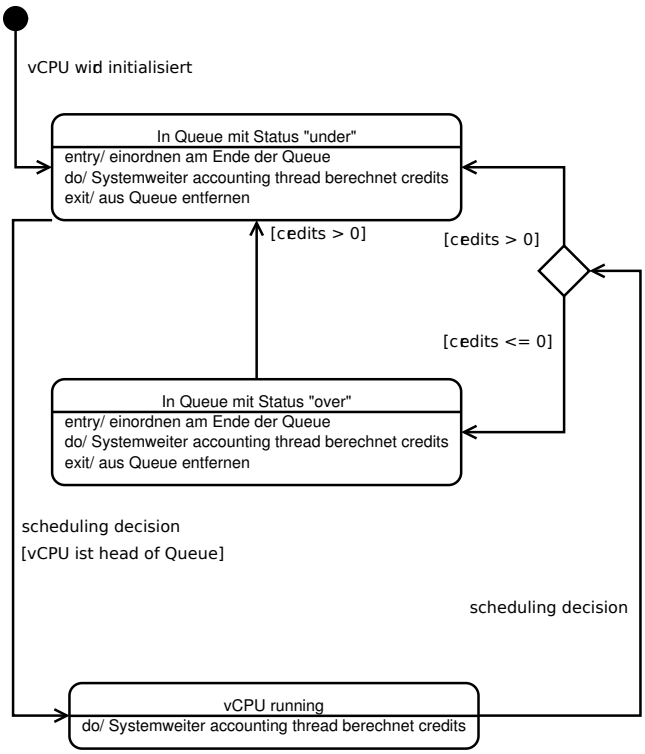

Abbildung 2.2: Scheduling einer vCPU

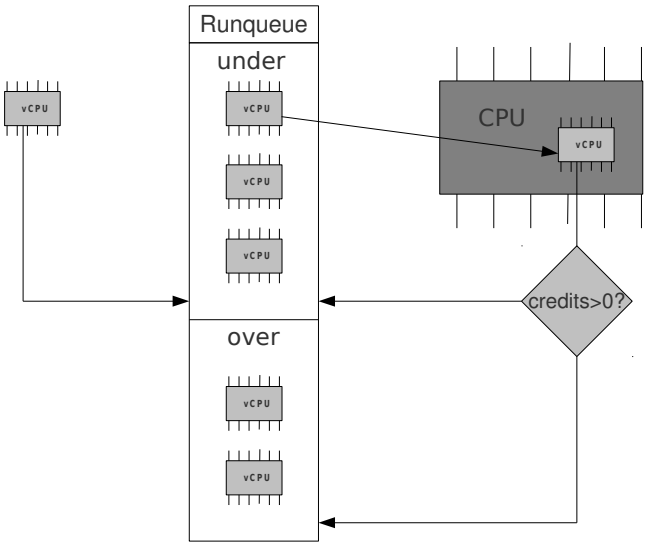

Abbildung 2.3: vCPU im Credit based Scheduler

3 Tests

3.1 Vorgehensweise

Im folgenden Kapitel soll nun zuerst die eingesetzte Hard- und Software vorgestellt werden, die zur Ausführung der Test zum Einsatz kamen. Benötigt wurden dabei eine Programm um single und multi threaded Prozesse darzustellen, hier die Software *stress*, sowie ein Benchmarktool um den Performanzunterschied der Durchläufe mit und ohne eingesetztem Daemon zu messen. Anschließend folgt eine Darstellung des Versuchsaufbaus samt geplantem Versuchsablauf, hier wurde ein Xen System mit zwei Domänen benötigt. Eine Domäne mit einer virtuellen CPU in der die Performance gemessen wurde sowie eine mit dem cpuplugd Daemon überwachte und der Software *stress* belastete Domäne mit zwei virtuellen CPUs. Darauf folgt eine Präsentation der Ergebnisse der einzelnen Benchmarktests. Abschließend werden diese, unter Einbeziehung von Teilen des Xen 3.2 Quellcodes, bewertet und damit die Frage aufgegriffen, ob der Einsatz des cpuplugd Daemons auf die Hypervisorplattform Xen Sinn macht.

3.2 Planung

3.2.1 eingesetzte Hardware

Bei der nachfolgend vorgestellten Versuchsreihe kam ein AMD Athlon64 X2 4800+ Chipsatz auf einem ASUS M2N-SLI Deluxe Mainboard, bestückt mit 4 mal SASMUNG 1024MB DDR2-800 PC2-6400 RAM, zum Einsatz.

3.2.2 eingesetzte Tools

Die eingesetzten Software-Tools wurden ausgewählt, da sie leicht zu beschaffen, relativ weit verbreitet und einfach in der Handhabung sind.

stress

Mit Hilfe des Tools „stress“ aus dem Ubuntu Repositorium ist es möglich, eine beliebige Anzahl von Threats (N) zu starten, welche durch zufällige Wurzelberechnungen die entsprechenden CPUs belasten. Dadurch ist es möglich, eine Belastung der entsprechenden Domäne mit single bzw. multi threaded Anwendungen darzustellen. Der Aufruf geschieht mit dem Kommando:
stress –cpu N
(vergl. Manual stress [MSt])

Linpack

Die eigentlichen Benchmarktests wurden mit Hilfe eines angepassten Linpack Benchmarktools durchgeführt. Es löst lineare Gleichungssysteme vorgegebener Größen und gibt die dafür benötigte Zeit an (vergl. [Lin]). Die eingesetzten Arraygrößen sind 1000, 2000 und 4000 mit jeweils 10 Durchgängen. Um der fehlerhaften Zeitgebung in virtualisierten Maschinen (vergl. [Sti07]) entgegenzuwirken wurde das Tool um eine Server/Client Architektur ergänzt, welche über eine dedizierte Netzwerkverbindung die aktuellen Zeiten von einem unabhängigen Rechner erhält.

3.2.3 Versuchsaufbau und Messstrategie

Um zu entscheiden, ob ein Einsatz des Daemons unter dem Hypervisor Xen sinnvoll ist, wurde ein Xen 3.2 System aufgesetzt, welches aus einer Dom0 sowie einer DomU besteht (Vergl. Abb.3.1). Als Distribution für die Dom0 wurde Linux Ubuntu mit Kernel 2.6.24-11-Server installiert. Als DomU, der zum Versuchszeitpunkt aktuellste, für paravirtualisierung angepasste, erhältliche Kernel aus dem Ubuntu Universum, 2.6.24-24-xen. Der Dom0 wurde 1, der DomU 2 virtuelle CPUs zugeteilt, wobei alle virtuellen CPUs auf die gleiche physische gebunden wurden. Durch das Binden können einzelne Kerne gezielt belastet werden. Dies ist nötig um zu vermeiden, dass bei den Tests die beiden Domänen in verschiedenen lokalen Warteschlangen laufen. Wäre dies der Fall würde die virtuelle CPU der Dom0, in der wir die Benchmarktests laufen lassen, nur Rechenzeit von der Warteschlange in der die virtuelle CPU der DomU läuft zugeteilt bekommen falls alle virtuellen CPUs der DomU den Status over hätten. Hätte das System aber beiden Domänen die selbe CPU zugeteilt, würden alle virtuellen CPUs in der selben Warteschlange laufen und das Scheduling würde sich damit anders verhalten. Zusätzlich wäre dann auch nicht ersichtlich auf welcher CPU die Xen eigenen Aufgaben laufen und welche Domäne damit weniger Rechenzeit zugeteilt bekommt. Dies könnte bei jedem Test auf einem andern Kern geschehen und die Ergebnisse der Benchmarktests in der Dom0 könnten sich bei jedem Durchlauf unterscheiden. Die Ergebnisse wären so nicht aussagekräftig. Die Einstellungen des Credit-Scheduler wurde auf den Standartwerten belassen, alle Domänen sind also gleichgestellt (Vergleiche hierzu auch Kapitel 2.1). Die Benchmarktests mit Linpack erfolgten in der Dom0, da diese zwar die privilegierte Domäne darstellt, aber im Scheduling den unprivilegierten gleichgestellt ist. Benötigt Linpack für die Berechnung der Arrays eine geringer Zeit, so steht der virtuellen CPU in der Dom0 mehr Rechenzeit zu Verfügung, die Performanz ist damit höher. Der eigentliche Dämon wurde in der DomU installiert und diese daraufhin abwechselnd mit einer bzw. zwei Instanzen von *stress* belastet, um so abwechselnd Belastungen von single bzw. multi threaded Anwendungen darzustellen. Dabei wurde darauf geachtet, eine genügend lange Zeit bis zum Start der Benchmarktest zu warten, da die Updaterate von cpuplugd auf 10 Sekunden eingestellt war und dies schon einen Minimalwert darstellt, da der Dämon auch hier mehrere Updatedurchgänge brauchte um sich an die neue Belastung anzupassen, also virtuelle CPUs online oder offline zu setzen.
Strategisch teilen sich die Tests in Durchläufe mit einer und zwei belasteten virtuellen CPUs, wobei die Testreihen mit einer belasteten virtuellen CPU jeweils mit und ohne Dämon durchgeführt werden. Mittels der eventuell ermittelten Performanzgewinne der Reihen mit einer belasten virtuellen CPU im Vergleich zu den Reihen mit zwei belasteten virtuellen CPUs wird zum einen ersichtlich wie der Xen eigene Scheduler mit unbeschäftigten virtuellen CPUs um-

geht und zum anderen inwieweit sich die Ergebnisse durch den Einsatz des cpuplud Dämons verbessern. Durch den Vergleich der Performanzgewinne der Durchläufe mit und ohne cpuplug Dämon kann der Einsatz des Dämons dann aufgrund des reinen Perfomanzgewinns bewertet werden.

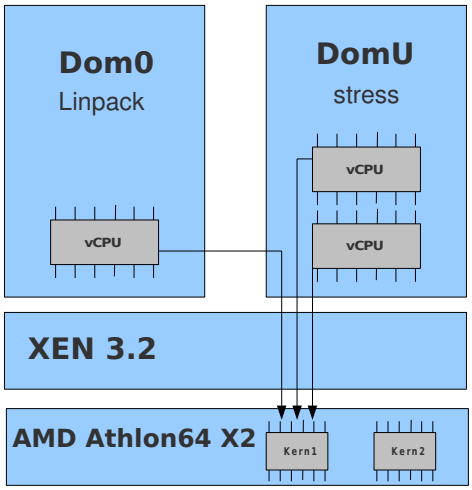

Abbildung 3.1: Versuchsaufbau

3.3 Ergebnisse

Nachfolgende Tabellen geben die Ergebnisse der Benchmarktests wieder. Ausgehend von den unterschiedlichen Arraygrößen werden die von Linpack benötigten Zeiten für 10 Durchgänge angegeben. Die Tabellen 3.1 und 3.2 stellen Tests mit aktiviertem Daemon und einer bzw. zwei „gestressten“ virtuellen CPUs in DomU dar. Tabelle 3.3 zeigt die Ergebnisse bei deaktiviertem cpuplugd Daemon und einer belasteten virtuellen CPU in DomU. Abbildung 3.2 bildet die Durchschnittswerte der einzelnen Tests nochmals grafisch ab.

Arraygröße	**Lauf 1**	**Lauf 2**	**Lauf 3**	**Mittelwert**
1000	6,39	6,43	6,39	6,40
2000	54,04	54,07	54,07	54,06
4000	420,38	420,16	420,17	420,24

Tabelle 3.1: 1 gestresste vCPU mit cpuplugd

Arraygröße	**Lauf 1**	**Lauf 2**	**Lauf 3**	**Mittelwert**
1000	8,93	9,02	9,13	9,03
2000	75,86	72,76	75,16	74,59
4000	578,5	586,41	590,73	585,21

Tabelle 3.2: 2 gestresste vCPUs mit cpuplugd

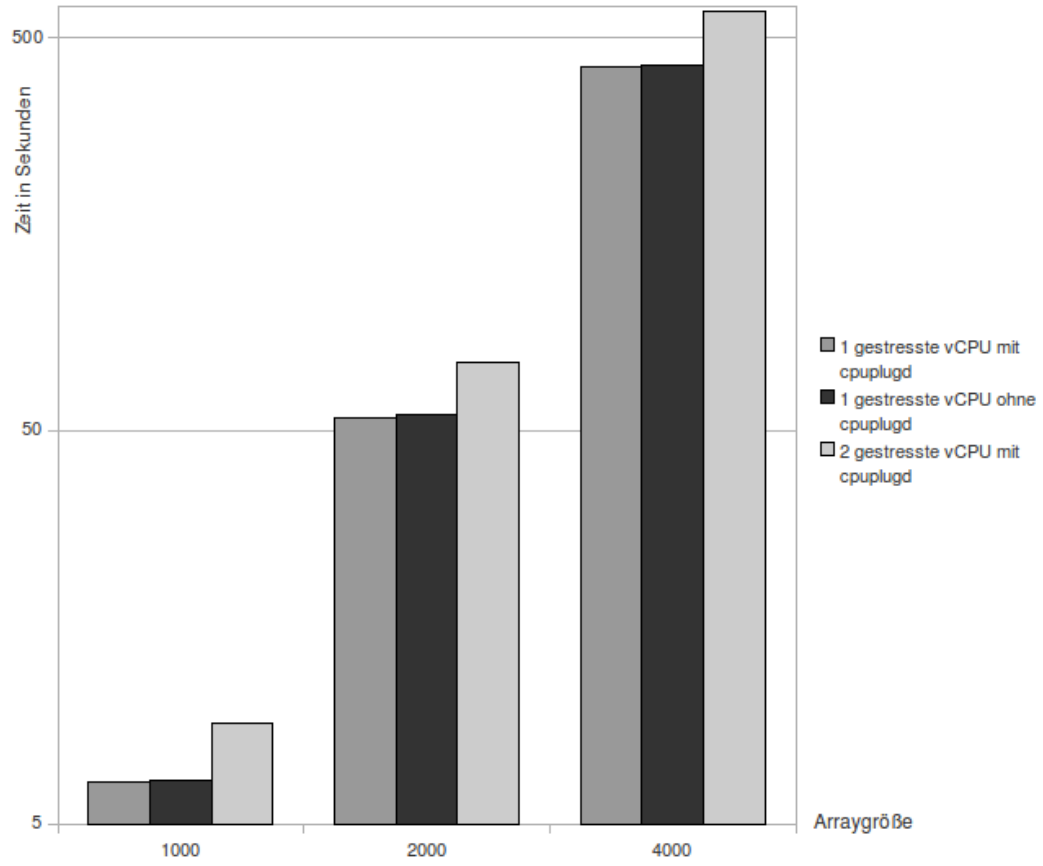

Abbildung 3.2: Grafische Gegenüberstellung

Arraygröße	Lauf 1	Lauf 2	Lauf 3	Mittelwert
1000	6,38	6,44	6,46	6,43
2000	54,78	54,78	54,64	54,73
4000	426,63	426,02	426,63	426,43

Tabelle 3.3: 1 gestresste vCPU ohne cpuplugd

3.4 Auswertung

Wie man aus den Tabellen 3.1 und 3.2 entnehmen kann, erhöht sich die Performanz bei Einsatz des Daemons und Belastung der DomU mit nur einer Instanz von stress, im Vergleich zu zwei Instanzen von stress, im Mittel um etwa 28%. Dies entspricht auch der Ausgabe von *xentop* (vergl. Manual xm [MXM]). In der Ausgabe von *xentop*, ein Tool welches Echtzeitinformationen über das Xen-System und seine Domänen ausgibt (vergl. [Mxt]), ist auch zu sehen, dass in diesem Fall in der DomU nur noch eine virtuelle CPU vorhanden ist. Cpuplugd verrichtet seine Arbeit also wie erwartet.
Vergleicht man aber die mittlere Laufzeit bei einer belasteten virtuellen CPU mit und ohne Daemon (Tabelle 3.3), so unterscheiden sich die Laufzeiten im Mittel nur um etwas mehr als 1 %. Auch die Ausgabe von *xentop* gibt bei den Tests ohne cpuplugd die Lastverteilung nun mit etwa 50:50 an, wobei in der DomU weiterhin 2 virtuelle CPUs angezeigt werden. Dies führt zu dem Schluss, dass das eingesetzte Scheduling von Xen keineswegs zu den, bei der Entwicklung des cpuplugd Daemons angenommenen, naiven Schedulingalgorithmen zählt. Eine genauere Betrachtung des Schedulings im Quellcodes von Xen bestätigt diese Vermutung. Bevor die entsprechende virtuelle CPU dispatcht wird, wird geprüft, ob diese lauffähig, also nicht idle, ist. Dies ist im folgenden Code-Ausschnitt zu erkennen (aus Xen:[root]/xen/common/sche_credit.c).

```
1173      /*
1174       * Select next runnable local VCPU (ie top of local runq)
1175       */
1176      if ( vcpu_runnable(current) )
1177          __runq_insert(cpu, scurr);
1178      else
1179          BUG_ON( is_idle_vcpu(current) || list_empty(runq) );
1180
1181      snext = __runq_elem(runq->next);
```

Die Funktionen *vcpu_runnable()* und *is_idle_vcpu()* werden im gesamten Scheduling von Xen mehrfach eingesetzt um die Lauffähigkeit einer virtuellen CPU zu testen. Dies geschieht auch schon im älteren Simple Earliest Deadline First Algorithmus.
Die Tatsache, dass der Einsatz des Daemons eine geringe Verbesserung der Performanz bringt, ist darauf zurückzuführen, dass die entsprechende virtuelle CPU nicht mehr gescheduled werden muss und der Schedulingturnus wesentlich kleiner ist als das Updateintervall von cpuplugd. Dieser Gewinn wird bei der Belastung aller virtuellen CPUs aber geschmälert, da der Daemon hier quasi unnütz im Hintergrund läuft.
Alles in Allem liegt der gemessene Leistungsgewinn weit hinter den Erwartungen zurück. Ein relevanter Gewinn ist nur bei vermehrtem Auftreten von single threaded Anwendungen, welche in ihrer Laufzeit das Updateintervall des Daemons überragen, zu erwarten. Ein

vorausschauendes Zuteilen der Rechenressourcen würde in den meisten Fällen ausreichen. Eine Domäne mit vielen single threaded Anwendungen könnte mit nur einer virtuellen CPU ausgestattet werden und über eine höhere Gewichtung die gleiche Rechenzeit wie eine entsprechende mit zwei virtuellen CPUs erhalten. Nicht zu vergessen ist, dass der Daemon einen nicht unerheblichen Einfluss auf den Credit Scheduler nimmt, da virtuelle CPUs im idle Modus während ihres Verbleibs in der Warteschlange Credits zugeteilt bekommen, aus- und wieder eingeschaltete virtuelle CPUs sich aber mit ihren Initialisierungswerten in die Queue einreihen. Auch muss bedacht werden, dass bei einigen Änderungen an einer virtuellen Maschine die Konfiguration des cpuplugd mit angepasst werden muss und dies eine weitere Fehlerquelle darstellt. Da die Parameter des Daemons in der unprivilegierten Domäne (DomU) eingestellt werden müssen, entzieht man der privilegierten Domäne (Dom0) dadurch die Kontrolle über diesen. Dies widerspricht zum einen der Idee von privilegierten und unprivilegierten Domänen, zum anderen gibt man bei kommerziell genutzten Systemen den Kunden unnötige Rechte. Zusätzlich muss bei Tests in den überwachten Domänen, bedacht werden, ob das Hotpluging evtl. Einfluss auf diese haben könnte.

4 Fazit

Aufgrund der Ergebnisse der Tests und der darauf folgenden Auswertung im vorhergehenden Kapitel komme ich zu dem Schluss, dass der cpuplugd, falls überhaupt, mit Bedacht und dem Bewusstsein, dass man die Kontrolle an die unprivilegierte Domain abgibt und man dadurch den Einsatz den Credit Scheduler beeinflusst, eingesetzt werden sollte. Eine Lösung um eventuell eine ähnliche Performanzsteigerung zu erziehen und einigen der Nachteilen des cpuplugd (vergl. Tabelle 4.1) zu entgehen, könnte es sein, die Funktion des Daemons in der Dom0 abzubilden und statt die virtuellen CPUs on- und offline zu stellen, diese bei der offline Bedingung aus der Queue zu nehmen und beim „online schalten“ so in die Queue einfügen, als hätte sie diese im Modus idle durchlaufen und die entsprechenden Credit zugeteilt bekommen. Die Kontrolle wäre in der Dom0, und in der DomU wären die virtuellen CPUs immer online, was bei eventuellen Test in dieser Bedingung sein könnte.

Vorteile	etwas performanter bei single threaded Anwendungen
Nachteile	Mehraufwand bei der Konfiguration Kontrolle liegt in der DomU beeinflusst den credit Scheduler evtl. Einfluss auf Tests in DomU

Tabelle 4.1: Vor-/Nachteile des cpuplugd

Abbildungsverzeichnis

Literaturverzeichnis

[CrS] *CreditScheduler - Xen Wiki.* http://wiki.xensource.com/xenwiki/CreditScheduler.

[DXP] *Xen.org Dokumentationsseiten.*

[Lin] *Linpack Dokumentation.* http://www.netlib.org/linpack/.

[MCp09] *Linux on System z documentation for 'Development stream'- Device Drivers, Features, and Commands (kernel 2.6.31) - SC33-8411-03*, September 2009. http://download.boulder.ibm.com/ibmdl/pub/software/dw/linux390/docu/lk31dd03.pdf.

[MSt] *Manual stress.* http://manpages.ubuntu.com/manpages/hardy/man1/stress.1.html.

[MXM] *Manual xm(1) - Xen management user interface.* http://linux.die.net/man/1/xm.

[Mxt] *xentop(1) - displays real-time information about a Xen system.* http://linux.die.net/man/1/xentop.

[PRO] *Linux Programmer's Manual - PROC(5).* http://www.kernel.org/doc/man-pages/online/pages/man5/proc.5.html.

[Sti07] Stiller, Andreas: *Xen und die virtuelle Zeit.* c't - Magazin für Computertechnik, (26):180, 2007.